



Protocol++ (ProtocolPP) 3.0.1

Generated by Doxygen 1.8.5

Thu Sept 5 2019 06:21:31

Copyright 2017-2019 to John Peter Greninger
US Copyrights at <https://www.copyright.gov/>

TXu002059872 (Version 1.0.0)
TXu002059872 (Version 1.2.7)
TXu002059872 (Version 1.4.0)
TXu002059872 (Version 2.0.0)

All Rights Reserved
All Derivatives the property of
John Peter Greninger until the year 2113

Contents

1	Protocol++ (ProtocolPP) Protocol, Encryption, and Authentication Library with Testbench and Drivers	1
1.1	Protocol++ Use Cases	2
1.2	Ciphers Interface	2
1.3	ProtocolPP Interface	4
1.4	DriverPP Interface	5
1.5	UDP	8
1.6	TCP	9
1.7	ICMP	10
1.8	IP	11
1.9	TLS	14
1.10	IKEv2	17
1.11	IPsec	18
1.12	Macsec	20
1.13	SRTP	21
1.14	Wifi/WiGig	23
1.15	WiMax	25
1.16	Long Term Evolution (LTE)	28
1.17	Digital Signature Algorithm (DSA)	30
1.18	Elliptic Curve Digital Signature Algorithm (ECDSA)	30
1.19	RSA CryptoSystem	30
1.20	W.A.S.P	31
2	Namespace Index	35
2.1	Namespace List	35
3	Hierarchical Index	37
3.1	Class Hierarchy	37
4	Class Index	41
4.1	Class List	41
5	File Index	45

5.1	File List	45
6	Namespace Documentation	49
6.1	DriverPP Namespace Reference	49
6.1.1	Variable Documentation	49
6.1.1.1	jsgt	49
6.2	InterfacePP Namespace Reference	49
6.3	option Namespace Reference	50
6.3.1	Detailed Description	51
6.3.2	Typedef Documentation	51
6.3.2.1	CheckArg	51
6.3.3	Enumeration Type Documentation	51
6.3.3.1	ArgStatus	51
6.3.4	Function Documentation	51
6.3.4.1	printUsage	51
6.3.4.2	printUsage	54
6.3.4.3	printUsage	54
6.3.4.4	printUsage	54
6.3.4.5	printUsage	54
6.4	PlatformPP Namespace Reference	54
6.4.1	Function Documentation	54
6.4.1.1	jsgt	55
6.5	ProtocolPP Namespace Reference	55
6.5.1	Detailed Description	69
6.5.2	Typedef Documentation	72
6.5.2.1	jpoly1305_state_internal_t	72
6.5.2.2	w128_t	72
6.5.3	Enumeration Type Documentation	72
6.5.3.1	auth_method_t	72
6.5.3.2	auth_t	73
6.5.3.3	cfg_attr_t	74
6.5.3.4	cfg_resp_t	76
6.5.3.5	cipher_t	76
6.5.3.6	dh_id_t	77
6.5.3.7	dir_id_t	79
6.5.3.8	direction_t	79
6.5.3.9	encoding_t	80
6.5.3.10	encr_id_t	80
6.5.3.11	endian_t	82
6.5.3.12	err_t	83

6.5.3.13 <code>esn_id_t</code>	86
6.5.3.14 <code>exchg_t</code>	86
6.5.3.15 <code>field_t</code>	87
6.5.4 IKEv2 fields	87
6.5.5 IPsec fields	88
6.5.6 ICMP fields	89
6.5.7 IP fields	89
6.5.8 LTE/3GPP fields	90
6.5.9 MacSEC fields	90
6.5.10 SRTP fields	91
6.5.11 TCP fields	91
6.5.12 TLS fields	92
6.5.13 UDP fields	92
6.5.14 Wifi/Wigig fields	92
6.5.15 WiMax fields	93
6.5.16 RSA CryptoSystem	93
6.5.17 Digital Signature Algorithm (DSA)	93
6.5.18 Elliptic Curve Digital Signature Algorithm (ECDSA)	94
6.5.18.1 <code>iana_t</code>	98
6.5.18.2 <code>icmpcode_t</code>	102
6.5.18.3 <code>icmpmsg_t</code>	104
6.5.18.4 <code>id_type_t</code>	105
6.5.18.5 <code>ike_gateway_t</code>	106
6.5.18.6 <code>ike_hash_t</code>	107
6.5.18.7 <code>ike_ipcomp_t</code>	107
6.5.18.8 <code>ike_pyld_t</code>	107
6.5.18.9 <code>ike_secpass_t</code>	109
6.5.18.10 <code>ike_ts_t</code>	110
6.5.18.11 <code>integ_id_t</code>	110
6.5.18.12 <code>ip_proto_t</code>	111
6.5.18.13 <code>ipmode_t</code>	112
6.5.18.14 <code>ipsec_mode_t</code>	112
6.5.18.15 <code>macseemode_t</code>	112
6.5.18.16 <code>notify_err_t</code>	113
6.5.18.17 <code>notify_status_t</code>	114
6.5.18.18 <code>pad_t</code>	116
6.5.18.19 <code>platform_t</code>	116
6.5.18.20 <code>prf_id_t</code>	117
6.5.18.21 <code>protocol_id_t</code>	117
6.5.18.22 <code>protocol_t</code>	118

6.5.18.23 replay_t	119
6.5.18.24 rohc_attr_t	120
6.5.18.25 role_id_t	121
6.5.18.26 rsapadtype_t	121
6.5.18.27 srtpcipher_t	121
6.5.18.28 tls_ciphersuite_t	123
6.5.18.29 tls_error_t	153
6.5.18.30 tls_handshake_t	155
6.5.18.31 tlslvl_t	156
6.5.18.32 tlstype_t	156
6.5.18.33 tlsver_t	156
6.5.18.34 transform_attr_t	157
6.5.18.35 transform_type_t	158
6.5.18.36 wifictext_t	158
6.5.18.37 wifisub_t	159
6.5.18.38 wifitype_t	160
6.5.18.39 wimaxmode_t	160
6.5.19 Function Documentation	161
6.5.19.1 Begin_Enum_String	161
6.5.19.2 Begin_Enum_String	165
6.5.19.3 Begin_Enum_String	165
6.5.19.4 Begin_Enum_String	165
6.5.19.5 Begin_Enum_String	165
6.5.19.6 Begin_Enum_String	165
6.5.19.7 Begin_Enum_String	167
6.5.19.8 Begin_Enum_String	167
6.5.19.9 Begin_Enum_String	167
6.5.19.10 Begin_Enum_String	167
6.5.19.11 Begin_Enum_String	167
6.5.19.12 Begin_Enum_String	167
6.5.19.13 Begin_Enum_String	167
6.5.19.14 Begin_Enum_String	167
6.5.19.15 Begin_Enum_String	167
6.5.19.16 Begin_Enum_String	167
6.5.19.17 Begin_Enum_String	167
6.5.19.18 Begin_Enum_String	167
6.5.19.19 Begin_Enum_String	167
6.5.19.20 Begin_Enum_String	167
6.5.19.21 Begin_Enum_String	167
6.5.19.22 Begin_Enum_String	167

6.5.19.23 Begin_Enum_String	167
6.5.19.24 Begin_Enum_String	167
6.5.19.25 Begin_Enum_String	167
6.5.19.26 Begin_Enum_String	167
6.5.19.27 Begin_Enum_String	167
6.5.19.28 Begin_Enum_String	168
6.5.19.29 Begin_Enum_String	168
6.5.19.30 Begin_Enum_String	168
6.5.19.31 Begin_Enum_String	168
6.5.19.32 Begin_Enum_String	168
6.5.19.33 Begin_Enum_String	168
6.5.19.34 Begin_Enum_String	168
6.5.19.35 Begin_Enum_String	168
6.5.19.36 Begin_Enum_String	168
6.5.19.37 Begin_Enum_String	168
6.5.19.38 Begin_Enum_String	168
6.5.19.39 Begin_Enum_String	168
6.5.19.40 Begin_Enum_String	168
6.5.19.41 Begin_Enum_String	168
6.5.19.42 Begin_Enum_String	168
6.5.19.43 Begin_Enum_String	168
6.5.19.44 Begin_Enum_String	168
6.5.19.45 Begin_Enum_String	168
6.5.19.46 Begin_Enum_String	168
6.5.19.47 Begin_Enum_String	168
6.5.19.48 Begin_Enum_String	168
6.5.19.49 Begin_Enum_String	168
6.5.19.50 Begin_Enum_String	168
6.5.19.51 Begin_Enum_String	168
6.5.19.52 Begin_Enum_String	168
6.5.19.53 Begin_Enum_String	168
6.5.19.54 Begin_Enum_String	168
6.5.19.55 Begin_Enum_String	168
6.5.19.56 do_recursion	169
6.5.19.57 gen_rand_array	170
6.5.19.58 gen_rand_array	170
6.5.19.59 gen_rand_array	170
6.5.19.60 gen_rand_array	170
6.5.19.61 jpoly1305_blocks	170
6.5.19.62 jpoly1305_blocks	170

6.5.19.63 lshift128	170
6.5.19.64 mm_recursion	171
6.5.19.65 mm_recursion	171
6.5.19.66 neon_recursion	171
6.5.19.67 rshift128	171
6.5.19.68 sfmt_gen_rand_all	172
6.5.19.69 swap	172
6.5.19.70 U16TO8	172
6.5.19.71 U32TO8	172
6.5.19.72 U64TO8	172
6.5.19.73 U8TO16	172
6.5.19.74 U8TO32	172
6.5.19.75 U8TO64	172
6.5.19.76 vec_recursion	172
6.5.20 Variable Documentation	172
6.5.20.1 End_Enum_String	172
6.6 tinyxml2 Namespace Reference	172
6.6.1 Enumeration Type Documentation	173
6.6.1.1 Whitespace	173
6.6.1.2 XMLError	173
7 Class Documentation	175
7.1 option::Parser::Action Struct Reference	175
7.1.1 Member Function Documentation	175
7.1.1.1 finished	175
7.1.1.2 perform	176
7.2 aead_chacha_poly1305 Class Reference	176
7.2.1 Detailed Description	176
7.2.2 ChaCha20 and Poly1305 for IETF protocols	176
7.3 ProtocolPP::aead_chacha_poly1305 Class Reference	179
7.3.1 Member Enumeration Documentation	179
7.3.1.1 dir_t	179
7.3.2 Constructor & Destructor Documentation	179
7.3.2.1 aead_chacha_poly1305	179
7.3.2.2 ~aead_chacha_poly1305	180
7.3.3 Member Function Documentation	180
7.3.3.1 ProcessData	180
7.3.3.2 result	180
7.4 option::Arg Struct Reference	180
7.4.1 Detailed Description	181

7.4.2	Member Function Documentation	182
7.4.2.1	None	182
7.4.2.2	Optional	182
7.4.2.3	Required	182
7.5	chacha20 Class Reference	182
7.5.1	Detailed Description	182
7.6	ProtocolPP::chacha20 Class Reference	183
7.6.1	Constructor & Destructor Documentation	184
7.6.1.1	chacha20	184
7.6.1.2	chacha20	185
7.6.1.3	~chacha20	185
7.6.2	Member Function Documentation	185
7.6.2.1	context	185
7.6.2.2	ProcessData	185
7.7	ciphers Class Reference	185
7.7.1	Detailed Description	185
7.8	ProtocolPP::ciphers Class Reference	187
7.8.1	Member Function Documentation	187
7.8.1.1	get_auth	187
7.8.1.2	get_cipher	187
7.8.1.3	get_crc32	187
7.9	option::Stats::CountOptionsAction Class Reference	188
7.9.1	Constructor & Destructor Documentation	188
7.9.1.1	CountOptionsAction	188
7.9.2	Member Function Documentation	188
7.9.2.1	perform	188
7.10	option::Descriptor Struct Reference	188
7.10.1	Detailed Description	189
7.10.2	Member Data Documentation	189
7.10.2.1	check_arg	189
7.10.2.2	help	189
7.10.2.3	index	190
7.10.2.4	longopt	190
7.10.2.5	shortopt	190
7.10.2.6	type	190
7.11	tinyxml2::DynArray< T, INITIAL_SIZE > Class Template Reference	191
7.11.1	Constructor & Destructor Documentation	191
7.11.1.1	DynArray	191
7.11.1.2	~DynArray	191
7.11.2	Member Function Documentation	191

7.11.2.1 Capacity	191
7.11.2.2 Clear	191
7.11.2.3 Empty	191
7.11.2.4 Mem	191
7.11.2.5 Mem	191
7.11.2.6 operator[]	191
7.11.2.7 operator[]	192
7.11.2.8 PeekTop	192
7.11.2.9 Pop	192
7.11.2.10 PopArr	192
7.11.2.11 Push	192
7.11.2.12 PushArr	192
7.11.2.13 Size	192
7.11.2.14 SwapRemove	192
7.12 EnumString Class Reference	192
7.12.1 Detailed Description	192
7.13 ProtocolPP::EnumString< EnumType > Struct Template Reference	193
7.13.1 Member Function Documentation	194
7.13.1.1 RegisterEnumerators	194
7.14 ProtocolPP::EnumStringBase< DerivedType, EnumType > Class Template Reference	194
7.14.1 Member Typedef Documentation	195
7.14.1.1 AssocMap	195
7.14.2 Constructor & Destructor Documentation	195
7.14.2.1 EnumStringBase	195
7.14.2.2 ~EnumStringBase	195
7.14.3 Member Function Documentation	195
7.14.3.1 Convert	195
7.14.3.2 From	195
7.14.3.3 GetIdx	195
7.14.3.4 RegisterEnumerator	195
7.14.3.5 size	195
7.14.3.6 To	195
7.15 InterfacePP::file_log_policy Class Reference	195
7.15.1 Constructor & Destructor Documentation	196
7.15.1.1 file_log_policy	196
7.15.1.2 ~file_log_policy	196
7.15.2 Member Function Documentation	196
7.15.2.1 close_ostream	196
7.15.2.2 open_ostream	196
7.15.2.3 write	196

7.16 InterfacePP::file_quiet_log_policy Class Reference	196
7.16.1 Constructor & Destructor Documentation	197
7.16.1.1 file_quiet_log_policy	197
7.16.1.2 ~file_quiet_log_policy	197
7.16.2 Member Function Documentation	197
7.16.2.1 close_ostream	197
7.16.2.2 open_ostream	197
7.16.2.3 write	197
7.17 InterfacePP::file_stdout_log_policy Class Reference	198
7.17.1 Constructor & Destructor Documentation	198
7.17.1.1 file_stdout_log_policy	198
7.17.1.2 ~file_stdout_log_policy	198
7.17.2 Member Function Documentation	198
7.17.2.1 close_ostream	198
7.17.2.2 open_ostream	198
7.17.2.3 write	199
7.18 option::PrintUsageImplementation::FunctionWriter< Function > Struct Template Reference	199
7.18.1 Constructor & Destructor Documentation	199
7.18.1.1 FunctionWriter	199
7.18.2 Member Function Documentation	199
7.18.2.1 operator()	199
7.18.3 Member Data Documentation	200
7.18.3.1 write	200
7.19 ikev2 Class Reference	200
7.19.1 Detailed Description	200
7.19.2 IKEv2 Configuration Schema	200
7.20 interfacepp Class Reference	201
7.20.1 Detailed Description	201
7.21 option::PrintUsageImplementation::IStringWriter Struct Reference	202
7.21.1 Member Function Documentation	202
7.21.1.1 operator()	202
7.22 jarray Class Reference	202
7.22.1 Detailed Description	203
7.23 ProtocolPP::jarray< T > Class Template Reference	203
7.23.1 Constructor & Destructor Documentation	204
7.23.1.1 jarray	204
7.23.1.2 ~jarray	204
7.23.1.3 jarray	205
7.23.1.4 ~jarray	205
7.23.1.5 ~jarray	205

7.23.1.6 <code>jarray</code>	205
7.23.1.7 <code>jarray</code>	205
7.23.1.8 <code>jarray</code>	206
7.23.1.9 <code>~jarray</code>	206
7.23.2 Member Function Documentation	206
7.23.2.1 <code>append</code>	206
7.23.2.2 <code>debug</code>	206
7.23.2.3 <code>empty</code>	207
7.23.2.4 <code>erase</code>	207
7.23.2.5 <code>extract</code>	207
7.23.2.6 <code>get_format</code>	207
7.23.2.7 <code>get_ptr</code>	207
7.23.2.8 <code>get_ptr</code>	208
7.23.2.9 <code>get_size</code>	208
7.23.2.10 <code>insert</code>	208
7.23.2.11 <code>operator!=</code>	208
7.23.2.12 <code>operator!=</code>	208
7.23.2.13 <code>operator&=</code>	208
7.23.2.14 <code>operator*=?</code>	209
7.23.2.15 <code>operator+=</code>	209
7.23.2.16 <code>operator==</code>	209
7.23.2.17 <code>operator==</code>	209
7.23.2.18 <code>operator[]</code>	209
7.23.2.19 <code>operator[]</code>	210
7.23.2.20 <code>operator^=</code>	210
7.23.2.21 <code>operator =</code>	210
7.23.2.22 <code>pop_back</code>	210
7.23.2.23 <code>push_back</code>	210
7.23.2.24 <code>resize</code>	210
7.23.2.25 <code>reverse</code>	211
7.23.2.26 <code>split</code>	211
7.23.2.27 <code>to_string</code>	211
7.23.2.28 <code>update</code>	211
7.24 <code>jdata</code> Class Reference	212
7.24.1 Detailed Description	212
7.25 <code>ProtocolPP::jdata</code> Class Reference	213
7.25.1 Constructor & Destructor Documentation	213
7.25.1.1 <code>jdata</code>	213
7.25.1.2 <code>jdata</code>	213
7.25.2 Member Function Documentation	213

7.25.2.1	get_flow	213
7.25.2.2	get_flowcnt	213
7.25.2.3	get_flows	214
7.25.2.4	get_packet	214
7.25.2.5	get_packets	214
7.25.2.6	get_pktnum	214
7.25.2.7	set_flow	214
7.25.2.8	set_packet	214
7.26	jdirectdrive Class Reference	215
7.26.1	Detailed Description	215
7.26.2	Driver for Protocol++	215
7.27	DriverPP::jdirectdrive Class Reference	216
7.27.1	Constructor & Destructor Documentation	216
7.27.1.1	jdirectdrive	216
7.27.1.2	~jdirectdrive	217
7.27.2	Member Function Documentation	217
7.27.2.1	avail	217
7.27.2.2	get_size	217
7.27.2.3	pop	217
7.27.2.4	push	217
7.27.2.5	push	217
7.27.2.6	push	218
7.27.2.7	runrcv	218
7.27.2.8	runsnd	218
7.27.2.9	teardown	218
7.27.2.10	write_mem	218
7.28	jdrive Class Reference	218
7.28.1	Detailed Description	219
7.28.2	Driver for Protocol++	219
7.29	DriverPP::jdrive Class Reference	219
7.29.1	Constructor & Destructor Documentation	220
7.29.1.1	jdrive	220
7.29.1.2	~jdrive	220
7.29.2	Member Function Documentation	220
7.29.2.1	get_size	220
7.29.2.2	pop	221
7.29.2.3	push	221
7.29.2.4	rcv	221
7.29.2.5	send	221
7.29.2.6	send	221

7.29.2.7	teardown	221
7.29.2.8	write_mem	221
7.30	jdriver Class Reference	222
7.30.1	Detailed Description	222
7.30.2	Driver for Protocol++	222
7.31	DriverPP::jdriver< Q > Class Template Reference	223
7.31.1	Constructor & Destructor Documentation	223
7.31.1.1	jdriver	223
7.31.1.2	~jdriver	224
7.31.2	Member Function Documentation	224
7.31.2.1	get_status	224
7.31.2.2	receive	224
7.31.2.3	send	224
7.31.2.4	teardown	224
7.31.2.5	teardown	225
7.31.2.6	write_mem	225
7.32	jdsa Class Reference	225
7.32.1	Detailed Description	225
7.32.2	Digital Signature Algorithm (DSA)	225
7.33	ProtocolPP::jdsa Class Reference	228
7.33.1	Constructor & Destructor Documentation	228
7.33.1.1	jdsa	228
7.33.1.2	jdsa	228
7.33.1.3	~jdsa	228
7.33.2	Member Function Documentation	228
7.33.2.1	gen_keypair	228
7.33.2.2	get_field	228
7.33.2.3	set_field	229
7.33.2.4	sign	229
7.33.2.5	verify	230
7.34	jecdsa Class Reference	230
7.34.1	Detailed Description	230
7.34.2	Elliptic Curve Digital Signature Algorithm (ECDSA)	230
7.35	ProtocolPP::jecdsa Class Reference	233
7.35.1	Constructor & Destructor Documentation	234
7.35.1.1	jecdsa	234
7.35.1.2	~jecdsa	234
7.35.2	Member Function Documentation	234
7.35.2.1	gen_keypair	234
7.35.2.2	get_field	234

7.35.2.3	set_field	234
7.35.2.4	sign	235
7.35.2.5	verify	235
7.36	jenum Class Reference	236
7.36.1	Detailed Description	236
7.37	jexec Class Reference	237
7.37.1	Detailed Description	237
7.37.2	Execution unit for use in testbench for Protocol++	237
7.38	InterfacePP::jexec Class Reference	238
7.38.1	Constructor & Destructor Documentation	238
7.38.1.1	jexec	238
7.38.1.2	~jexec	239
7.38.2	Member Function Documentation	239
7.38.2.1	exec	239
7.38.2.2	get_busy	239
7.38.2.3	get_complat	239
7.38.2.4	get_outlen	239
7.38.2.5	get_readlat	240
7.38.2.6	get_status	240
7.38.2.7	get_writelat	240
7.38.2.8	set_computlat	240
7.38.2.9	set_readlat	240
7.38.2.10	set_writelat	240
7.39	jicmp Class Reference	240
7.39.1	Detailed Description	241
7.39.2	Internet Control Message Protocol (ICMP)	241
7.39.2.1	Internet Control Message Protocol version 6	243
7.40	ProtocolPP::jicmp Class Reference	246
7.40.1	Constructor & Destructor Documentation	246
7.40.1.1	jicmp	246
7.40.1.2	~jicmp	247
7.40.2	Member Function Documentation	247
7.40.2.1	convcode	247
7.40.2.2	convmsg	247
7.40.2.3	decap_packet	247
7.40.2.4	encap_packet	247
7.40.2.5	get_field	248
7.40.2.6	get_hdr	248
7.40.2.7	set_field	248
7.40.2.8	set_hdr	248

7.40.2.9	to_xml	248
7.41	jicmpsa Class Reference	249
7.41.1	Detailed Description	249
7.41.2	Internet Control Message Protocol Security Association (ICMPSA)	249
7.41.2.1	Internet Control Message Protocol version 6	251
7.42	ProtocolPP::jicmpsa Class Reference	252
7.42.1	Constructor & Destructor Documentation	253
7.42.1.1	jicmpsa	253
7.42.1.2	jicmpsa	254
7.42.1.3	jicmpsa	255
7.42.1.4	jicmpsa	255
7.42.1.5	~jicmpsa	255
7.42.2	Member Function Documentation	255
7.42.2.1	get_field	255
7.42.2.2	set_field	256
7.42.2.3	to_xml	257
7.43	ProtocolPP::jikeparse::jikecfg Struct Reference	257
7.43.1	Constructor & Destructor Documentation	259
7.43.1.1	jikecfg	259
7.43.2	Member Data Documentation	259
7.43.2.1	ah_integ	259
7.43.2.2	arwindow	259
7.43.2.3	auth_generator	259
7.43.2.4	auth_verifiers	259
7.43.2.5	bypassdf	259
7.43.2.6	bypassdscp	259
7.43.2.7	ca_certificates	259
7.43.2.8	cert_black_list	259
7.43.2.9	cert_white_list	259
7.43.2.10	certificates	259
7.43.2.11	connection	260
7.43.2.12	cookies_lifetime	260
7.43.2.13	cookies_threshold	260
7.43.2.14	debugclr	260
7.43.2.15	dhcp_interface	260
7.43.2.16	dhcp_retries	260
7.43.2.17	dhcp_server_ip	260
7.43.2.18	dhcp_timeout	260
7.43.2.19	eap_clients	260
7.43.2.20	eap_md5_password	260

7.43.2.21 eap_md5_user_db	260
7.43.2.22 eap_server	260
7.43.2.23 eap_tls_ca_cert	260
7.43.2.24 eap_tls_client_cert	260
7.43.2.25 eap_tls_private_key	260
7.43.2.26 eap_tls_private_key_password	260
7.43.2.27 errclr	260
7.43.2.28 esp_dh	260
7.43.2.29 esp_encr	260
7.43.2.30 esp_integ	260
7.43.2.31 espmode	260
7.43.2.32 fatalclr	260
7.43.2.33 fixed_ipv4_prefix	260
7.43.2.34 fixed_ipv4_prefix_mask	260
7.43.2.35 fixed_ipv6_prefix	260
7.43.2.36 fixed_ipv6_prefix_mask	260
7.43.2.37 flush_before	260
7.43.2.38 format	260
7.43.2.39 generate_allow_policies	261
7.43.2.40 hash_url_support	261
7.43.2.41 hide_mask	261
7.43.2.42 ike_dh	261
7.43.2.43 ike_encr	261
7.43.2.44 ike_integ	261
7.43.2.45 ike_prf	261
7.43.2.46 ikepolicies	261
7.43.2.47 infoclr	261
7.43.2.48 lifetime_hard	261
7.43.2.49 lifetime_soft	261
7.43.2.50 local_id	261
7.43.2.51 local_type	261
7.43.2.52 max_bytes_hard	261
7.43.2.53 max_bytes_soft	261
7.43.2.54 max_idle_time	261
7.43.2.55 max_ike_negotiation	261
7.43.2.56 max_retries	261
7.43.2.57 method	261
7.43.2.58 noncelInitiator	261
7.43.2.59 nonceResponder	261
7.43.2.60 passclr	261

7.43.2.61 peer_addr	261
7.43.2.62 peer_ca_certificates	261
7.43.2.63 peer_preshared_key	261
7.43.2.64 peerid_id	261
7.43.2.65 peerid_type	261
7.43.2.66 preshared_key	261
7.43.2.67 protected_ipv4_subnet	262
7.43.2.68 protected_ipv4_subnet_mask	262
7.43.2.69 protected_ipv6_subnet	262
7.43.2.70 protected_ipv6_subnet_mask	262
7.43.2.71 radius_port	262
7.43.2.72 radius_secret	262
7.43.2.73 radius_server	262
7.43.2.74 randiv	262
7.43.2.75 reauth_time	262
7.43.2.76 rekey_time	262
7.43.2.77 remote_host	262
7.43.2.78 remote_port	262
7.43.2.79 request_configuration	262
7.43.2.80 request_ipv6_suffix	262
7.43.2.81 retransmition_factor	262
7.43.2.82 retransmition_time	262
7.43.2.83 role	262
7.43.2.84 seed	262
7.43.2.85 send_cert_payload	262
7.43.2.86 send_cert_req_payload	262
7.43.2.87 show_extra_info	262
7.43.2.88 show_mask	262
7.43.2.89 tfclen	262
7.43.2.90 use_esn	262
7.43.2.91 use_uname	262
7.43.2.92 vendor_id	262
7.43.2.93 warnclr	262
7.44 ProtocolPP::jikencrypt Class Reference	263
7.44.1 Constructor & Destructor Documentation	263
7.44.1.1 jikencrypt	263
7.44.1.2 ~jikencrypt	263
7.44.2 Member Function Documentation	263
7.44.2.1 encrypt	263
7.44.2.2 get_random	263

7.44.2.3	integrity	264
7.45	jikencrypt Class Reference	264
7.45.1	Detailed Description	264
7.45.2	IKE Encrypt and Integrity Functions	264
7.46	ProtocolPP::jikeparse Class Reference	265
7.46.1	Constructor & Destructor Documentation	266
7.46.1.1	jikeparse	266
7.46.1.2	jikeparse	266
7.46.1.3	jikeparse	266
7.46.1.4	\sim jikeparse	266
7.46.2	Member Function Documentation	266
7.46.2.1	get	266
7.47	jikeparse Class Reference	267
7.47.1	Detailed Description	267
7.47.2	IKEPARSE	267
7.48	ProtocolPP::jikeparse::jikepolicy Struct Reference	268
7.48.1	Constructor & Destructor Documentation	268
7.48.1.1	jikepolicy	268
7.48.2	Member Data Documentation	268
7.48.2.1	dir	268
7.48.2.2	dst_port	268
7.48.2.3	dst_selector	269
7.48.2.4	dst_selector_mask	269
7.48.2.5	dst_tunnel	269
7.48.2.6	ip_proto	269
7.48.2.7	ipsec_proto	269
7.48.2.8	mode	269
7.48.2.9	priority	269
7.48.2.10	src_port	269
7.48.2.11	src_selector	269
7.48.2.12	src_selector_mask	269
7.48.2.13	src_tunnel	269
7.49	ProtocolPP::jikeprf Class Reference	269
7.49.1	Constructor & Destructor Documentation	269
7.49.1.1	jikeprf	269
7.49.1.2	\sim jikeprf	269
7.49.2	Member Function Documentation	270
7.49.2.1	get_prf	270
7.49.2.2	get_prfplus	271
7.50	jikeprf Class Reference	271

7.50.1 Detailed Description	271
7.50.2 IKE PRF and PRF+ functions	271
7.51 ProtocolPP::jikev2 Class Reference	272
7.51.1 Constructor & Destructor Documentation	273
7.51.1.1 jikev2	273
7.51.1.2 jikev2	273
7.51.1.3 ~jikev2	273
7.51.2 Member Function Documentation	273
7.51.2.1 add_sa	273
7.51.2.2 comms	274
7.51.2.3 delete_sa	274
7.51.2.4 findsa	274
7.51.2.5 get_fields	274
7.51.2.6 get_fields	275
7.51.2.7 propose	275
7.51.2.8 reload	275
7.51.2.9 run	275
7.51.2.10 teardown	275
7.52 jikev2 Class Reference	275
7.52.1 Detailed Description	276
7.52.2 Internet Key Exchange Version 2	276
7.52.3 Scenarios	276
7.52.4 Header and Payload Formats	280
7.52.5 Generic Payload Header	281
7.52.6 Security Association Payload	282
7.52.7 Proposal Substructure	282
7.52.8 Transform Substructure	283
7.52.9 Transform Attributes	284
7.52.10 Key Exchange Payload	285
7.52.11 Identification Payload	285
7.52.12 Certification Payload	286
7.52.13 Certification Request Payload	286
7.52.14 Authentication Payload	287
7.52.15 Nonce Payload	288
7.52.16 Notify Payload	288
7.52.17 Delete Payload	289
7.52.18 Vendor ID Payload	289
7.52.19 Traffic Selectors Payload	290
7.52.20 Traffic Selector	290
7.52.21 Encrypted Payload Format	291

7.52.22 Configuration Payload	292
7.52.23 Configuration Attributes	293
7.52.24 EAP Payload	293
7.52.25 EAP Message	293
7.52.26 IKEv2 State Diagram	295
7.53 ProtocolPP::jikev2dh Class Reference	297
7.53.1 Constructor & Destructor Documentation	297
7.53.1.1 jikev2dh	297
7.53.1.2 ~jikev2dh	297
7.53.2 Member Function Documentation	297
7.53.2.1 get_pubkey	297
7.53.2.2 verify	297
7.54 jikev2dh Class Reference	298
7.54.1 Detailed Description	298
7.54.2 Diffie-Hellman Support for IKEv2	298
7.55 jikev2sa Class Reference	300
7.55.1 Detailed Description	300
7.55.2 Internet Key Exchange (IKEv2) Security Association	300
7.55.3 Scenarios	300
7.55.4 Header and Payload Formats	304
7.55.5 Generic Payload Header	305
7.55.6 Security Association Payload	306
7.55.7 Proposal Substructure	306
7.55.8 Transform Substructure	307
7.55.9 Transform Attributes	308
7.55.10 Key Exchange Payload	309
7.55.11 Identification Payload	309
7.55.12 Certification Payload	310
7.55.13 Certification Request Payload	310
7.55.14 Authentication Payload	311
7.55.15 Nonce Payload	312
7.55.16 Notify Payload	312
7.55.17 Delete Payload	313
7.55.18 Vendor ID Payload	313
7.55.19 Traffic Selectors Payload	314
7.55.20 Traffic Selector	314
7.55.21 Encrypted Payload Format	315
7.55.22 Configuration Payload	316
7.55.23 Configuration Attributes	317
7.55.24 EAP Payload	317

7.55.25 EAP Message	317
7.55.26 IKEv2 State Diagram	319
7.56 ProtocolPP::jikev2sa Class Reference	321
7.56.1 Constructor & Destructor Documentation	321
7.56.1.1 jikev2sa	321
7.56.1.2 jikev2sa	321
7.56.1.3 jikev2sa	322
7.56.1.4 jikev2sa	322
7.56.1.5 ~jikev2sa	322
7.56.2 Member Function Documentation	323
7.56.2.1 get_field	323
7.56.2.2 set_field	323
7.56.2.3 to_xml	323
7.57 jip Class Reference	323
7.57.1 Detailed Description	323
7.57.2 Internet Protocol (IP)	323
7.57.3 IPv6 Extension Headers	328
7.58 ProtocolPP::jip Class Reference	335
7.58.1 Constructor & Destructor Documentation	335
7.58.1.1 jip	335
7.58.1.2 jip	336
7.58.1.3 ~jip	336
7.58.2 Member Function Documentation	336
7.58.2.1 decap_packet	336
7.58.2.2 decap_packet	336
7.58.2.3 encapsulate_packet	336
7.58.2.4 encapsulate_packet	337
7.58.2.5 format_exthdr	337
7.58.2.6 get_exthdr	337
7.58.2.7 get_field	337
7.58.2.8 get_field	338
7.58.2.9 get_hdr	339
7.58.2.10 set_exthdr	339
7.58.2.11 set_field	339
7.58.2.12 set_hdr	339
7.58.2.13 to_xml	339
7.59 jipsa Class Reference	340
7.59.1 Detailed Description	340
7.59.2 Internet Protocol (IP) Security Association	340
7.60 ProtocolPP::jipsa Class Reference	345

7.60.1	Constructor & Destructor Documentation	346
7.60.1.1	jipsa	346
7.60.1.2	jipsa	346
7.60.1.3	jipsa	346
7.60.1.4	jipsa	347
7.60.1.5	~jipsa	347
7.60.2	Member Function Documentation	347
7.60.2.1	get_field	347
7.60.2.2	set_field	347
7.60.2.3	to_xml	347
7.61	jipsec Class Reference	348
7.61.1	Detailed Description	348
7.61.2	Encapsulating Security Payload (ESP)	348
7.61.2.1	Encapsulating Security Protocol Processing	354
7.61.2.2	Network Address Translation Transversal (NAT)	356
7.61.2.3	Using Advanced Encryption Standard (AES) Counter Mode With IPsec Encapsulating Security Payload (ESP)	359
7.61.2.4	The Use of Galois/Counter Mode (GCM) in IPsec Encapsulating Security Payload (ESP)	361
7.61.2.5	Using Advanced Encryption Standard (AES) CCM Mode with IPsec Encapsulating Security Payload (ESP)	363
7.61.2.6	ChaCha20, Poly1305, and Their Use in the Internet Key Exchange Protocol (IKE) and IPsec	366
7.61.2.7	IPsec Cipher and Authentication Synopsis	369
7.62	ProtocolPP::jipsec Class Reference	370
7.62.1	Member Enumeration Documentation	371
7.62.1.1	audit_t	371
7.62.2	Constructor & Destructor Documentation	371
7.62.2.1	jipsec	371
7.62.2.2	~jipsec	371
7.62.3	Member Function Documentation	371
7.62.3.1	audit	371
7.62.3.2	decap_packet	372
7.62.3.3	encap_packet	372
7.62.3.4	get_exthdr	372
7.62.3.5	get_field	372
7.62.3.6	get_field	373
7.62.3.7	get_hdr	373
7.62.3.8	set_field	373
7.62.3.9	set_hdr	373
7.62.3.10	to_xml	373

7.63 jipsecsa Class Reference	374
7.63.1 Detailed Description	374
7.63.2 Encapsulating Security Payload (ESP) Security Association	374
7.64 ProtocolPP::jipsecsa Class Reference	381
7.64.1 Constructor & Destructor Documentation	382
7.64.1.1 jipsecsa	382
7.64.1.2 jipsecsa	382
7.64.1.3 jipsecsa	384
7.64.1.4 jipsecsa	384
7.64.1.5 ~jipsecsa	384
7.64.2 Member Function Documentation	384
7.64.2.1 get_field	384
7.64.2.2 set_field	384
7.64.2.3 to_xml	385
7.65 InterfacePP::jlogger Class Reference	385
7.65.1 Detailed Description	386
7.65.2 Member Enumeration Documentation	386
7.65.2.1 asciicolor	386
7.65.2.2 severity_type	387
7.65.3 Constructor & Destructor Documentation	387
7.65.3.1 jlogger	387
7.65.3.2 ~jlogger	388
7.65.4 Member Function Documentation	388
7.65.4.1 get_color	388
7.65.4.2 print	388
7.65.4.3 set_color	388
7.65.4.4 toEnum	388
7.65.4.5 toStr	388
7.66 jlte Class Reference	389
7.66.1 Detailed Description	389
7.66.2 Long Term Evolution (LTE)	389
7.66.2.1 PDCP SN	389
7.66.2.2 Data	390
7.66.2.3 MAC-I	390
7.66.2.4 COUNT	390
7.66.2.5 R	390
7.66.2.6 D/C	390
7.66.2.7 PDU type	390
7.66.2.8 FMS	390
7.66.2.9 Bitmap	390

7.66.2.10 Interspersed ROHC feedback packet	391
7.66.2.11 PGK Index	391
7.66.2.12 PTK Identity	391
7.66.2.13 SDU Type	391
7.66.2.14 KD-sess ID	391
7.66.2.15 NMP	391
7.66.2.16 HRW	391
7.66.2.17 P	391
7.66.2.18 Encryption	391
7.66.2.19 Authentication	391
7.66.2.20 LTE Control Plane Packet formats [4]	391
7.66.2.21 LTE User Plane Packet formats [5]	391
7.67 ProtocolPP::jlte Class Reference	391
7.67.1 Constructor & Destructor Documentation	391
7.67.1.1 jlte	391
7.67.1.2 ~jlte	392
7.67.2 Member Function Documentation	392
7.67.2.1 decap_packet	392
7.67.2.2 encapsulate	392
7.67.2.3 get_field	392
7.67.2.4 get_field	392
7.67.2.5 get_hdr	393
7.67.2.6 set_field	393
7.67.2.7 set_hdr	393
7.67.2.8 to_xml	393
7.68 jltesa Class Reference	393
7.68.1 Detailed Description	394
7.68.2 Long Term Evolution (LTE) Security Association	394
7.68.2.1 PDCP SN	394
7.68.2.2 Data	394
7.68.2.3 MAC-I	394
7.68.2.4 COUNT	395
7.68.2.5 R	395
7.68.2.6 D/C	395
7.68.2.7 PDU type	395
7.68.2.8 FMS	395
7.68.2.9 Bitmap	395
7.69 ProtocolPP::jltesa Class Reference	396
7.69.1 Constructor & Destructor Documentation	397
7.69.1.1 jltesa	397

7.69.1.2	jltesa	397
7.69.1.3	jltesa	398
7.69.1.4	jltesa	398
7.69.1.5	~jltesa	398
7.69.2	Member Function Documentation	398
7.69.2.1	get_field	398
7.69.2.2	set_field	399
7.69.2.3	to_xml	399
7.70	jmacsec Class Reference	399
7.70.1	Detailed Description	399
7.70.2	MACsec protocol	399
7.70.3	Packet Processing	405
7.70.4	Cipher Suites	409
7.70.4.1	Media Access Control (MAC) Security—Amendment 2: Extended Packet Numbering	413
7.70.4.2	Security TAG	414
7.70.4.3	Packet Number (PN)	414
7.70.4.4	PN recovery and preliminary replay check	414
7.70.4.5	verification controls	414
7.70.4.6	Receive SA creation	415
7.70.4.7	Transmit SA creation	415
7.70.4.8	SAK creation	415
7.70.4.9	Cipher Suite use	415
7.70.4.10	Cipher Suite conformance	416
7.71	ProtocolPP::jmacsec Class Reference	420
7.71.1	Constructor & Destructor Documentation	420
7.71.1.1	jmacsec	420
7.71.2	Member Function Documentation	420
7.71.2.1	decap_packet	420
7.71.2.2	encap_packet	421
7.71.2.3	get_field	421
7.71.2.4	get_field	421
7.71.2.5	get_hdr	422
7.71.2.6	set_field	422
7.71.2.7	set_hdr	422
7.71.2.8	to_xml	422
7.72	jmacsecsa Class Reference	422
7.72.1	Detailed Description	423
7.72.2	MACsec Security Association	423
7.73	ProtocolPP::jmacsecsa Class Reference	429
7.73.1	Constructor & Destructor Documentation	429

7.73.1.1	jmacsecsa	429
7.73.1.2	jmacsecsa	429
7.73.1.3	jmacsecsa	430
7.73.1.4	jmacsecsa	430
7.73.2	Member Function Documentation	430
7.73.2.1	get_field	430
7.73.2.2	set_field	430
7.73.2.3	to_xml	431
7.74	InterfacePP::jmmu Class Reference	431
7.74.1	Constructor & Destructor Documentation	431
7.74.1.1	jmmu	431
7.74.1.2	~jmmu	431
7.74.2	Member Function Documentation	431
7.74.2.1	free	431
7.74.2.2	free_mem	432
7.74.2.3	set_mem	433
7.74.2.4	set_mem	433
7.74.2.5	track	433
7.75	jmmu Class Reference	433
7.75.1	Detailed Description	433
7.75.2	Management Unit for Protocol++ testbench	433
7.76	ProtocolPP::jmodes Class Reference	434
7.76.1	Member Enumeration Documentation	435
7.76.1.1	cipher_t	435
7.76.1.2	dir_t	435
7.76.1.3	mode_t	436
7.76.2	Constructor & Destructor Documentation	436
7.76.2.1	jmodes	436
7.76.2.2	jmodes	437
7.76.2.3	~jmodes	437
7.76.3	Member Function Documentation	437
7.76.3.1	context	437
7.76.3.2	ProcessData	437
7.76.3.3	ProcessData	438
7.76.3.4	result	438
7.77	jmodes Class Reference	438
7.77.1	Detailed Description	438
7.77.2	Block Modes of Operation	438
7.77.2.1	Electronic Codebook (ECB)	440
7.77.2.2	Cipher Block Chaining (CBC)	441

7.77.2.3	Counter (CTR)	442
7.77.2.4	Galois Counter Mode (GCM)	443
7.77.2.5	Counter with CBC-MAC (CCM)	445
7.77.2.6	The AES-XCBC-MAC-96 Algorithm and Its Use With IPsec	446
7.77.2.7	The CMAC Mode for Authentication	447
7.78	jpacket Class Reference	452
7.78.1	Detailed Description	452
7.78.2	Packet class for Protocol++	452
7.79	ProtocolPP::jpacket Class Reference	453
7.79.1	Constructor & Destructor Documentation	454
7.79.1.1	jpacket	454
7.79.1.2	jpacket	454
7.79.1.3	~jpacket	455
7.79.2	Member Function Documentation	455
7.79.2.1	get_expect	455
7.79.2.2	get_explen	455
7.79.2.3	get_inlen	455
7.79.2.4	get_input	455
7.79.2.5	get_name	455
7.79.2.6	get_outaddr	456
7.79.2.7	get_outlen	456
7.79.2.8	get_output	456
7.79.2.9	get_outputlen	456
7.79.2.10	get_previous	456
7.79.2.11	get_status	456
7.79.2.12	get_stream	456
7.79.2.13	set_expect	457
7.79.2.14	set_input	458
7.79.2.15	set_name	458
7.79.2.16	set_outaddr	458
7.79.2.17	set_outlen	458
7.79.2.18	set_output	458
7.79.2.19	set_previous	458
7.79.2.20	set_status	459
7.79.2.21	set_stream	459
7.80	ProtocolPP::jpoly1305 Class Reference	459
7.80.1	Constructor & Destructor Documentation	459
7.80.1.1	jpoly1305	459
7.80.1.2	jpoly1305	459
7.80.1.3	~jpoly1305	460

7.80.2 Member Function Documentation	460
7.80.2.1 context	460
7.80.2.2 ProcessData	460
7.80.2.3 result	460
7.81 jpoly1305 Class Reference	460
7.81.1 Detailed Description	460
7.82 ProtocolPP::jpoly1305_state_internal_t Struct Reference	461
7.82.1 Member Data Documentation	462
7.82.1.1 buffer	462
7.82.1.2 final	462
7.82.1.3 h	462
7.82.1.4 h	462
7.82.1.5 h	462
7.82.1.6 h	462
7.82.1.7 leftover	462
7.82.1.8 pad	462
7.82.1.9 pad	462
7.82.1.10 pad	462
7.82.1.11 pad	462
7.82.1.12 r	462
7.82.1.13 r	462
7.82.1.14 r	462
7.82.1.15 r	462
7.83 jproducer Class Reference	462
7.83.1 Detailed Description	463
7.83.2 Producer for Protocol++	463
7.84 InterfacePP::jproducer Class Reference	464
7.84.1 Constructor & Destructor Documentation	464
7.84.1.1 jproducer	464
7.84.1.2 ~jproducer	465
7.84.2 Member Function Documentation	465
7.84.2.1 free_mem	465
7.84.2.2 get_mem	465
7.84.2.3 get_mode	465
7.84.2.4 get_status	465
7.84.2.5 issue	465
7.84.2.6 read	466
7.84.2.7 read	466
7.84.2.8 read	466
7.84.2.9 release	466

7.84.2.10 run	467
7.84.2.11 set_mem	467
7.84.2.12 setup	467
7.84.2.13 write	467
7.84.2.14 write	467
7.84.2.15 write	467
7.84.2.16 write	468
7.85 jprotocol Class Reference	468
7.85.1 Detailed Description	468
7.86 ProtocolPP::jprotocol Class Reference	469
7.86.1 Constructor & Destructor Documentation	471
7.86.1.1 jprotocol	471
7.86.1.2 jprotocol	472
7.86.1.3 jprotocol	472
7.86.1.4 jprotocol	472
7.86.1.5 ~jprotocol	472
7.86.2 Member Function Documentation	472
7.86.2.1 checksum	472
7.86.2.2 currentDateTime	473
7.86.2.3 decap_packet	473
7.86.2.4 decap_packet	473
7.86.2.5 decrypt	473
7.86.2.6 encap_packet	473
7.86.2.7 encap_packet	474
7.86.2.8 encrypt	474
7.86.2.9 get_data	474
7.86.2.10 get_field	474
7.86.2.11 get_field	474
7.86.2.12 get_hdr	475
7.86.2.13 get_status	475
7.86.2.14 hasfile	477
7.86.2.15 pad	477
7.86.2.16 put_data	478
7.86.2.17 roundup	478
7.86.2.18 set_field	478
7.86.2.19 set_hdr	478
7.86.2.20 str_status	478
7.86.2.21 to_array	479
7.86.2.22 to_array	479
7.86.2.23 to_array	479

7.86.2.24 to_array	479
7.86.2.25 to_u16	480
7.86.2.26 to_u32	480
7.86.2.27 to_u64	480
7.86.2.28 to_u8	480
7.86.2.29 to_xml	480
7.86.2.30 update_replay	481
7.86.2.31 update_replay	481
7.86.2.32 update_status	481
7.86.3 Member Data Documentation	482
7.86.3.1 m_dir	482
7.86.3.2 m_endian	482
7.86.3.3 m_file	482
7.86.3.4 m_filename	482
7.86.3.5 m_rand	482
7.86.3.6 m_status	482
7.87 jprotocolpp Class Reference	482
7.88 ProtocolPP::jprotocolpp Class Reference	482
7.88.1 Member Function Documentation	483
7.88.1.1 get_icmp	483
7.88.1.2 get_ip	483
7.88.1.3 get_ip	483
7.88.1.4 get_ipsec	483
7.88.1.5 get_lte	484
7.88.1.6 get_macsec	484
7.88.1.7 get_srtp	484
7.88.1.8 get_tcp	484
7.88.1.9 get_tcp	484
7.88.1.10 get_tls	485
7.88.1.11 get_tls	485
7.88.1.12 get_udp	485
7.88.1.13 get_udp	485
7.88.1.14 get_wifi	485
7.88.1.15 get_wimax	486
7.89 jrand Class Reference	486
7.89.1 Detailed Description	486
7.90 ProtocolPP::jrand Class Reference	487
7.90.1 Constructor & Destructor Documentation	488
7.90.1.1 jrand	488
7.90.1.2 jrand	488

7.90.1.3 <code>jrand</code>	488
7.90.1.4 <code>~jrand</code>	488
7.90.2 Member Function Documentation	488
7.90.2.1 <code>get_crypto</code>	488
7.90.2.2 <code>get_int</code>	488
7.90.2.3 <code>get_int</code>	489
7.90.2.4 <code>get_u16</code>	490
7.90.2.5 <code>get_u32</code>	490
7.90.2.6 <code>get_u64</code>	490
7.90.2.7 <code>get_u8</code>	490
7.90.2.8 <code>get_uint</code>	490
7.90.2.9 <code>getbool</code>	491
7.90.2.10 <code>getbyte</code>	491
7.90.2.11 <code>getbyte</code>	491
7.90.2.12 <code>getchar</code>	491
7.90.2.13 <code>getdouble</code>	491
7.90.2.14 <code>getdouble</code>	492
7.90.2.15 <code>getname</code>	492
7.90.2.16 <code>getstr</code>	492
7.90.2.17 <code>getu16</code>	492
7.90.2.18 <code>getu16</code>	492
7.90.2.19 <code>getword</code>	493
7.90.2.20 <code>getword</code>	493
7.90.2.21 <code>seed</code>	493
7.90.2.22 <code>seed</code>	493
7.90.2.23 <code>tokenizer</code>	493
7.91 <code>jreplay</code> Class Reference	494
7.91.1 Detailed Description	494
7.91.2 Anti-Replay	494
7.92 <code>ProtocolPP::jreplay< T, TE ></code> Class Template Reference	496
7.92.1 Constructor & Destructor Documentation	497
7.92.1.1 <code>jreplay</code>	497
7.92.1.2 <code>jreplay</code>	497
7.92.1.3 <code>~jreplay</code>	497
7.92.2 Member Function Documentation	497
7.92.2.1 <code>antireplay</code>	497
7.92.2.2 <code>get_extseq</code>	498
7.92.2.3 <code>get_seqnum</code>	498
7.92.2.4 <code>get_window</code>	498
7.92.2.5 <code>next</code>	498

7.92.2.6 <code>print</code>	499
7.92.2.7 <code>size</code>	500
7.93 <code>jresponder</code> Class Reference	501
7.93.1 Detailed Description	501
7.93.2 Responder unit for use in testbench for Protocol++	501
7.94 <code>InterfacePP::jresponder</code> Class Reference	501
7.94.1 Constructor & Destructor Documentation	502
7.94.1.1 <code>jresponder</code>	502
7.94.2 Member Function Documentation	502
7.94.2.1 <code>dequeue</code>	502
7.94.2.2 <code>enqueue</code>	502
7.94.2.3 <code>pop</code>	502
7.94.2.4 <code>push</code>	503
7.94.2.5 <code>run</code>	503
7.94.2.6 <code>teardown</code>	503
7.95 <code>jring</code> Class Reference	503
7.95.1 Detailed Description	503
7.95.2 Software ring for use in testbench for Protocol++	503
7.96 <code>InterfacePP::jring< TR ></code> Class Template Reference	504
7.96.1 Constructor & Destructor Documentation	504
7.96.1.1 <code>jring</code>	504
7.96.1.2 <code>~jring</code>	505
7.96.2 Member Function Documentation	505
7.96.2.1 <code>get_avail</code>	505
7.96.2.2 <code>get_proc</code>	505
7.96.2.3 <code>move</code>	505
7.96.2.4 <code>pop</code>	505
7.96.2.5 <code>pop</code>	505
7.96.2.6 <code>push</code>	506
7.96.2.7 <code>reset</code>	506
7.96.2.8 <code>size</code>	506
7.97 <code>jringdrive</code> Class Reference	506
7.97.1 Detailed Description	506
7.97.2 Driver for Protocol++ with input and output driven by a software ring	506
7.98 <code>DriverPP::jringdrive</code> Class Reference	508
7.98.1 Constructor & Destructor Documentation	508
7.98.1.1 <code>jringdrive</code>	508
7.98.1.2 <code>~jringdrive</code>	508
7.98.2 Member Function Documentation	508
7.98.2.1 <code>get_status</code>	509

7.98.2.2 receive	509
7.98.2.3 send	509
7.98.2.4 teardown	509
7.98.2.5 write_mem	509
7.99 jrsa Class Reference	509
7.99.1 Detailed Description	509
7.99.2 RSA CryptoSystem	509
7.100ProtocolPP::jrsa Class Reference	513
7.100.1 Constructor & Destructor Documentation	513
7.100.1.1 jrsa	513
7.100.1.2 ~jrsa	514
7.100.2 Member Function Documentation	514
7.100.2.1 decrypt	514
7.100.2.2 encrypt	514
7.100.2.3 gen_keypair	514
7.100.2.4 get_field	514
7.100.2.5 set_field	514
7.100.2.6 sign	515
7.100.2.7 verify	515
7.101jsec Class Reference	515
7.101.1 Detailed Description	515
7.101.2 Security Encryption CoProcessor (SEC)	515
7.102PlatformPP::jsec Class Reference	516
7.102.1 Constructor & Destructor Documentation	517
7.102.1.1 jsec	517
7.102.1.2 ~jsec	517
7.102.2 Member Function Documentation	517
7.102.2.1 get_desc	517
7.102.2.2 get_shared	517
7.102.2.3 read_sgt	517
7.103ProtocolPP::jsecass Class Reference	518
7.103.1 Constructor & Destructor Documentation	518
7.103.1.1 jsecass	518
7.103.1.2 ~jsecass	519
7.103.2 Member Function Documentation	519
7.103.2.1 to_xml	519
7.104jsecass Class Reference	519
7.104.1 Detailed Description	519
7.105jsecproducer Class Reference	521
7.105.1 Detailed Description	521

7.105.2 Producer for Protocol++ for SEC/CAAM platform	521
7.106InterfacePP::jsecproducer Class Reference	522
7.106.1 Constructor & Destructor Documentation	523
7.106.1.1 jsecproducer	523
7.106.1.2 ~jsecproducer	524
7.106.2 Member Function Documentation	524
7.106.2.1 free_mem	524
7.106.2.2 get_mem	524
7.106.2.3 get_mode	524
7.106.2.4 get_sgt	524
7.106.2.5 get_status	525
7.106.2.6 issue	525
7.106.2.7 read	525
7.106.2.8 read	525
7.106.2.9 read	525
7.106.2.10release	526
7.106.2.11run	526
7.106.2.12set_mem	526
7.106.2.13setup	526
7.106.2.14write	526
7.106.2.15write	526
7.106.2.16write	527
7.106.2.17write	527
7.107jsecresponder Class Reference	527
7.107.1 Detailed Description	527
7.107.2 Responder unit for use in testbench for Protocol++	527
7.108InterfacePP::jsecresponder Class Reference	528
7.108.1 Constructor & Destructor Documentation	529
7.108.1.1 jsecresponder	529
7.108.2 Member Function Documentation	529
7.108.2.1 dequeue	529
7.108.2.2 enqueue	529
7.108.2.3 pop	529
7.108.2.4 push	529
7.108.2.5 run	530
7.109InterfacePP::jsectestbench Class Reference	530
7.109.1 Constructor & Destructor Documentation	530
7.109.1.1 jsectestbench	530
7.109.1.2 ~jsectestbench	531
7.109.2 Member Function Documentation	531

7.109.2.1 <code>get_mem</code>	531
7.109.2.2 <code>read</code>	531
7.109.2.3 <code>read</code>	531
7.109.2.4 <code>read</code>	532
7.109.2.5 <code>write</code>	532
7.109.2.6 <code>write</code>	532
7.109.2.7 <code>write</code>	532
7.109.2.8 <code>write</code>	532
7.110 <code>jsectestbench</code> Class Reference	533
7.110.1 Detailed Description	533
7.110.2 Testbench for Protocol++	533
7.111 <code>jsgt</code> Class Reference	534
7.111.1 Detailed Description	534
7.111.2 Scatter Gather Table (SGT)	534
7.112 <code>PlatformPP::jsgt</code> Class Reference	535
7.112.1 Constructor & Destructor Documentation	535
7.112.1.1 <code>jsgt</code>	535
7.112.1.2 <code>~jsgt</code>	535
7.112.2 Member Function Documentation	535
7.112.2.1 <code>get</code>	536
7.112.2.2 <code>read</code>	536
7.112.2.3 <code>sgt_t</code>	536
7.112.3 Member Data Documentation	536
7.112.3.1 <code>addrsize</code>	536
7.112.3.2 <code>bpid</code>	536
7.112.3.3 <code>chunksize</code>	536
7.112.3.4 <code>entrysize</code>	536
7.112.3.5 <code>extend</code>	536
7.112.3.6 <code>length</code>	536
7.112.3.7 <code>offset</code>	536
7.112.3.8 <code>order</code>	536
7.112.3.9 <code>sgt_t</code>	536
7.112.3.10 <code>sgtfinal</code>	536
7.113 <code>jsnow3g</code> Class Reference	536
7.113.1 Detailed Description	536
7.113.2 Snow3G Protocol	536
7.114 <code>ProtocolPP::jsnow3g</code> Class Reference	537
7.114.1 Member Enumeration Documentation	538
7.114.1.1 <code>dir_t</code>	538
7.114.2 Constructor & Destructor Documentation	538

7.114.2.1 jsnow3g	538
7.114.2.2 jsnow3g	538
7.114.2.3 ~jsnow3g	538
7.114.3 Member Function Documentation	539
7.114.3.1 context	539
7.114.3.2 ProcessData	539
7.114.3.3 ProcessData	539
7.114.3.4 result	539
7.114.3.5 result_size	539
7.115ProtocolPP::jsrtp Class Reference	539
7.115.1 Constructor & Destructor Documentation	540
7.115.1.1 jsrtp	540
7.115.1.2 ~jsrtp	540
7.115.2 Member Function Documentation	540
7.115.2.1 decap_packet	540
7.115.2.2 encapsulate_packet	541
7.115.2.3 get_exthdr	541
7.115.2.4 get_field	541
7.115.2.5 get_field	541
7.115.2.6 get_hdr	542
7.115.2.7 set_exthdr	542
7.115.2.8 set_field	542
7.115.2.9 set_field	542
7.115.2.10 set_hdr	542
7.115.2.11 to_xml	542
7.116jsrtp Class Reference	543
7.116.1 Detailed Description	543
7.116.2 Secure Real-Time Protocol (SRTP)	543
7.116.2.1 SRTP Packet Processing	547
7.116.2.2 Generic AEAD Processing	549
7.116.2.3 Counter Mode Encryption	550
7.116.2.4 AES-GCM processing for SRTP	551
7.116.2.5 AES-CCM for SRTP/SRTCP	552
7.116.2.6 Use of the ARIA Block cipher with SRTP	554
7.117ProtocolPP::jsrtpsa Class Reference	558
7.117.1 Constructor & Destructor Documentation	558
7.117.1.1 jsrtpsa	558
7.117.1.2 jsrtpsa	558
7.117.1.3 jsrtpsa	559
7.117.1.4 jsrtpsa	559

7.117.1.5 ~jsrtpsa	559
7.117.2 Member Function Documentation	559
7.117.2.1 get_field	560
7.117.2.2 set_field	560
7.117.2.3 to_xml	560
7.118jsrtpsa Class Reference	560
7.118.1 Detailed Description	560
7.118.2 Secure Real-Time Protocol Security Association (SRTP)	560
7.119jstream Class Reference	560
7.119.1 Detailed Description	560
7.119.2 Stream class for protocol++	560
7.120ProtocolPP::jstream Class Reference	560
7.120.1 Constructor & Destructor Documentation	560
7.120.1.1 jstream	560
7.120.1.2 jstream	561
7.120.1.3 jstream	561
7.120.1.4 ~jstream	561
7.120.2 Member Function Documentation	561
7.120.2.1 clean	561
7.120.2.2 get_dir	561
7.120.2.3 get_engine	562
7.120.2.4 get_name	562
7.120.2.5 get_post	562
7.120.2.6 get_security	562
7.120.2.7 get_setup	562
7.120.2.8 get_size	562
7.120.2.9 get_type	562
7.120.2.10set_dir	563
7.120.2.11set_engine	564
7.120.2.12set_name	564
7.120.2.13set_post	564
7.120.2.14set_security	564
7.120.2.15set_setup	564
7.120.2.16set_size	564
7.120.2.17set_type	565
7.121jtcp Class Reference	565
7.121.1 Detailed Description	565
7.121.2 Transport Control Protocol (TCP)	565
7.122ProtocolPP::jtcp Class Reference	575
7.122.1 Member Enumeration Documentation	575

7.122.1.1 <code>tcpopt_t</code>	575
7.122.1.2 <code>tcpstate_t</code>	576
7.122.2 Constructor & Destructor Documentation	576
7.122.2.1 <code>jtcp</code>	576
7.122.2.2 <code>jtcp</code>	577
7.122.2.3 <code>~jtcp</code>	577
7.122.3 Member Function Documentation	577
7.122.3.1 <code>add_option</code>	577
7.122.3.2 <code>decap_packet</code>	577
7.122.3.3 <code>encap_packet</code>	577
7.122.3.4 <code>get_field</code>	578
7.122.3.5 <code>get_field</code>	578
7.122.3.6 <code>get_hdr</code>	578
7.122.3.7 <code>set_field</code>	578
7.122.3.8 <code>set_hdr</code>	578
7.122.3.9 <code>to_xml</code>	579
7.123 <code>jtcpsa</code> Class Reference	579
7.123.1 Detailed Description	579
7.123.2 Transport Control Protocol Security Association (TCP)	579
7.124 <code>ProtocolPP::jtcpsa</code> Class Reference	582
7.124.1 Constructor & Destructor Documentation	582
7.124.1.1 <code>jtcpsa</code>	582
7.124.1.2 <code>jtcpsa</code>	582
7.124.1.3 <code>jtcpsa</code>	583
7.124.1.4 <code>jtcpsa</code>	583
7.124.2 Member Function Documentation	583
7.124.2.1 <code>get_field</code>	583
7.124.2.2 <code>set_field</code>	584
7.124.2.3 <code>to_xml</code>	585
7.125 <code>jtestbench</code> Class Reference	585
7.125.1 Detailed Description	585
7.125.2 Testbench for <code>Protocol++</code>	585
7.126 <code>InterfacePP::jtestbench</code> Class Reference	586
7.126.1 Constructor & Destructor Documentation	587
7.126.1.1 <code>jtestbench</code>	587
7.126.1.2 <code>~jtestbench</code>	588
7.126.2 Member Function Documentation	588
7.126.2.1 <code>get_mem</code>	588
7.126.2.2 <code>read</code>	588
7.126.2.3 <code>read</code>	588

7.126.2.4 <code>read</code>	589
7.126.2.5 <code>write</code>	589
7.126.2.6 <code>write</code>	589
7.126.2.7 <code>write</code>	589
7.126.2.8 <code>write</code>	589
7.127 <code>jtls</code> Class Reference	590
7.127.1 Detailed Description	590
7.127.2 Transport Layer Security (TLS)	590
7.127.2.1 Basic TLS handshake	590
7.127.2.2 ChaCha20-Poly1305 Cipher Suites for Transport Layer Security (TLS)	594
7.127.2.3 Addition of Camellia Cipher Suites to Transport Layer Security (TLS)	596
7.127.2.4 Addition of the ARIA Cipher Suites to Transport Layer Security (TLS)	597
7.127.2.5 Addition of SEED Cipher Suites to Transport Layer Security (TLS)	601
7.127.2.6 Encrypt-then-MAC for Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS) [2]	602
7.128 <code>ProtocolPP::jtls</code> Class Reference	605
7.128.1 Constructor & Destructor Documentation	606
7.128.1.1 <code>jtls</code>	606
7.128.1.2 <code>jtls</code>	606
7.128.1.3 <code>~jtls</code>	606
7.128.2 Member Function Documentation	606
7.128.2.1 <code>decap_packet</code>	606
7.128.2.2 <code>encap_packet</code>	606
7.128.2.3 <code>get_field</code>	606
7.128.2.4 <code>get_hdr</code>	607
7.128.2.5 <code>get_prf</code>	607
7.128.2.6 <code>set_field</code>	607
7.128.2.7 <code>set_hdr</code>	607
7.128.2.8 <code>to_xml</code>	608
7.129 <code>ProtocolPP::jtlsa</code> Class Reference	608
7.129.1 Constructor & Destructor Documentation	608
7.129.1.1 <code>jtlsa</code>	608
7.129.1.2 <code>jtlsa</code>	608
7.129.1.3 <code>jtlsa</code>	609
7.129.1.4 <code>jtlsa</code>	609
7.129.1.5 <code>~jtlsa</code>	609
7.129.2 Member Function Documentation	609
7.129.2.1 <code>get_field</code>	609
7.129.2.2 <code>set_field</code>	610
7.129.2.3 <code>to_xml</code>	610

7.130jtlsa Class Reference	610
7.130.1 Detailed Description	610
7.130.2 Transport Layer Security Association (TLSA)	610
7.131judp Class Reference	612
7.131.1 Detailed Description	612
7.131.2 User Datagram Protocol (UDP)	612
7.132ProtocolPP::judp Class Reference	612
7.132.1 Constructor & Destructor Documentation	612
7.132.1.1 judp	612
7.132.1.2 judp	613
7.132.1.3 ~judp	613
7.132.2 Member Function Documentation	613
7.132.2.1 decap_packet	613
7.132.2.2 decap_packet	613
7.132.2.3 encapsulate_packet	613
7.132.2.4 encapsulate_packet	614
7.132.2.5 get_field	615
7.132.2.6 get_field	615
7.132.2.7 get_hdr	615
7.132.2.8 set_field	615
7.132.2.9 set_hdr	615
7.132.2.10 to_xml	616
7.133ProtocolPP::judpsa Class Reference	616
7.133.1 Constructor & Destructor Documentation	616
7.133.1.1 judpsa	616
7.133.1.2 judpsa	617
7.133.1.3 judpsa	618
7.133.1.4 judpsa	618
7.133.1.5 ~judpsa	618
7.133.2 Member Function Documentation	618
7.133.2.1 get_field	618
7.133.2.2 set_field	618
7.133.2.3 to_xml	619
7.134judpsa Class Reference	619
7.134.1 Detailed Description	619
7.134.2 User Datagram Protocol Security Association (UDPSA)	619
7.135ProtocolPP::jwifi Class Reference	621
7.135.1 Constructor & Destructor Documentation	621
7.135.1.1 jwifi	621
7.135.1.2 ~jwifi	622

7.135.2 Member Function Documentation	622
7.135.2.1 decap_packet	622
7.135.2.2 encapsulate_packet	622
7.135.2.3 get_field	622
7.135.2.4 get_field	622
7.135.2.5 get_hdr	623
7.135.2.6 kdf	623
7.135.2.7 pbkdf2	623
7.135.2.8 prf	623
7.135.2.9 set_field	624
7.135.2.10 set_hdr	624
7.135.2.11 to_xml	624
7.136jwifi Class Reference	624
7.136.1 Detailed Description	624
7.136.2 Wifi/WiGig Protocol	624
7.136.3 Broadcast/multicast integrity protocol (BIP)	632
7.136.4 AES-GCM Use in Wifi	635
7.137ProtocolPP::jwifisa Class Reference	651
7.137.1 Constructor & Destructor Documentation	652
7.137.1.1 jwifisa	652
7.137.1.2 jwifisa	652
7.137.1.3 jwifisa	653
7.137.1.4 jwifisa	653
7.137.1.5 ~jwifisa	653
7.137.2 Member Function Documentation	653
7.137.2.1 get_field	653
7.137.2.2 set_field	653
7.137.2.3 to_xml	653
7.138jwifisa Class Reference	654
7.138.1 Detailed Description	654
7.138.2 Wifi/WiGig Protocol Security Association	654
7.138.3 Broadcast/multicast integrity protocol (BIP)	661
7.138.4 AES-GCM Use in Wifi	664
7.139jwimax Class Reference	680
7.139.1 Detailed Description	680
7.139.2 WiMax Protocol (WiMax)	680
7.140ProtocolPP::jwimax Class Reference	690
7.140.1 Constructor & Destructor Documentation	690
7.140.1.1 jwimax	690
7.140.1.2 ~jwimax	691

7.140.2 Member Function Documentation	691
7.140.2.1 decap_packet	691
7.140.2.2 encapsulate_packet	691
7.140.2.3 get_field	691
7.140.2.4 get_hdr	691
7.140.2.5 set_field	692
7.140.2.6 set_hdr	693
7.140.2.7 to_xml	693
7.141 ProtocolPP::jwimaxsa Class Reference	693
7.141.1 Constructor & Destructor Documentation	694
7.141.1.1 jwimaxsa	694
7.141.1.2 jwimaxsa	694
7.141.1.3 jwimaxsa	694
7.141.1.4 jwimaxsa	694
7.141.1.5 ~jwimaxsa	695
7.141.2 Member Function Documentation	695
7.141.2.1 get_field	695
7.141.2.2 set_field	695
7.141.2.3 to_xml	695
7.142 jwimaxsa Class Reference	695
7.142.1 Detailed Description	695
7.142.2 WiMax Protocol Security Association (WiMax SA)	695
7.143 jzuc Class Reference	701
7.143.1 Detailed Description	701
7.143.2 3GPP ZUC Protocol	701
7.144 ProtocolPP::jzuc Class Reference	702
7.144.1 Member Enumeration Documentation	702
7.144.1.1 dir_t	702
7.144.2 Constructor & Destructor Documentation	703
7.144.2.1 jzuc	703
7.144.2.2 ~jzuc	704
7.144.2.3 ~jzuc	704
7.144.3 Member Function Documentation	704
7.144.3.1 ProcessData	704
7.144.3.2 ProcessData	704
7.144.3.3 result	704
7.144.3.4 result_size	705
7.145 option::PrintUsageImplementation::LinePartIterator Class Reference	705
7.145.1 Constructor & Destructor Documentation	705
7.145.1.1 LinePartIterator	705

7.145.2 Member Function Documentation	706
7.145.2.1 column	706
7.145.2.2 data	706
7.145.2.3 length	706
7.145.2.4 line	706
7.145.2.5 next	706
7.145.2.6 nextRow	706
7.145.2.7 nextTable	706
7.145.2.8 restartRow	706
7.145.2.9 restartTable	706
7.145.2.10screenLength	707
7.146option::PrintUsageImplementation::LineWrapper Class Reference	707
7.146.1 Constructor & Destructor Documentation	707
7.146.1.1 LineWrapper	707
7.146.2 Member Function Documentation	707
7.146.2.1 flush	707
7.146.2.2 process	707
7.147InterfacePP::log_policy_interface Class Reference	708
7.147.1 Member Function Documentation	708
7.147.1.1 close_ostream	708
7.147.1.2 open_ostream	708
7.147.1.3 write	708
7.148log_policy_interface Class Reference	709
7.148.1 Detailed Description	709
7.148.2 logger class	709
7.149tinyxml2::MemPool Class Reference	711
7.149.1 Constructor & Destructor Documentation	711
7.149.1.1 MemPool	711
7.149.1.2 ~MemPool	711
7.149.2 Member Function Documentation	711
7.149.2.1 Alloc	711
7.149.2.2 Free	712
7.149.2.3 ItemSize	712
7.149.2.4 SetTracked	712
7.150tinyxml2::MemPoolT< ITEM_SIZE > Class Template Reference	712
7.150.1 Member Enumeration Documentation	713
7.150.1.1 anonymous enum	713
7.150.2 Constructor & Destructor Documentation	713
7.150.2.1 MemPoolT	713
7.150.2.2 ~MemPoolT	713

7.150.3 Member Function Documentation	713
7.150.3.1 Alloc	713
7.150.3.2 Clear	713
7.150.3.3 CurrentAllocs	713
7.150.3.4 Free	713
7.150.3.5 ItemSize	713
7.150.3.6 SetTracked	713
7.150.3.7 Trace	713
7.150.3.8 Untracked	713
7.151 ProtocolPP::MTRand Class Reference	713
7.151.1 Constructor & Destructor Documentation	714
7.151.1.1 MTRand	714
7.151.1.2 MTRand	714
7.151.1.3 MTRand	714
7.151.1.4 ~MTRand	714
7.151.2 Member Function Documentation	714
7.151.2.1 operator()	714
7.152 ProtocolPP::MTRand53 Class Reference	714
7.152.1 Constructor & Destructor Documentation	715
7.152.1.1 MTRand53	715
7.152.1.2 MTRand53	715
7.152.1.3 MTRand53	715
7.152.1.4 ~MTRand53	715
7.152.2 Member Function Documentation	715
7.152.2.1 operator()	715
7.153 ProtocolPP::MTRand_closed Class Reference	715
7.153.1 Constructor & Destructor Documentation	715
7.153.1.1 MTRand_closed	715
7.153.1.2 MTRand_closed	715
7.153.1.3 MTRand_closed	716
7.153.1.4 ~MTRand_closed	716
7.153.2 Member Function Documentation	716
7.153.2.1 operator()	716
7.154 ProtocolPP::MTRand_int32 Class Reference	716
7.154.1 Detailed Description	716
7.154.2 Constructor & Destructor Documentation	717
7.154.2.1 MTRand_int32	717
7.154.2.2 MTRand_int32	717
7.154.2.3 MTRand_int32	717
7.154.2.4 ~MTRand_int32	717

7.154.3 Member Function Documentation	717
7.154.3.1 operator()	717
7.154.3.2 rand_int32	717
7.154.3.3 seed	717
7.154.3.4 seed	717
7.155 ProtocolPP::MTRand_open Class Reference	717
7.155.1 Constructor & Destructor Documentation	718
7.155.1.1 MTRand_open	718
7.155.1.2 MTRand_open	718
7.155.1.3 MTRand_open	718
7.155.1.4 ~MTRand_open	718
7.155.2 Member Function Documentation	718
7.155.2.1 operator()	718
7.156 option::Option Class Reference	718
7.156.1 Detailed Description	719
7.156.2 Constructor & Destructor Documentation	720
7.156.2.1 Option	720
7.156.2.2 Option	720
7.156.2.3 Option	720
7.156.3 Member Function Documentation	720
7.156.3.1 append	720
7.156.3.2 count	720
7.156.3.3 first	720
7.156.3.4 index	720
7.156.3.5 isFirst	721
7.156.3.6 isLast	721
7.156.3.7 last	721
7.156.3.8 next	721
7.156.3.9 nextwrap	721
7.156.3.10 operator const Option *	721
7.156.3.11 operator Option *	722
7.156.3.12 operator=	722
7.156.3.13 prev	722
7.156.3.14 prevwrap	722
7.156.3.15 type	722
7.156.4 Member Data Documentation	723
7.156.4.1 arg	723
7.156.4.2 desc	723
7.156.4.3 name	723
7.156.4.4 namelen	723

7.157option::PrintUsageImplementation::OStreamWriter< OStream > Struct Template Reference	724
7.157.1 Constructor & Destructor Documentation	724
7.157.1.1 OStreamWriter	724
7.157.2 Member Function Documentation	724
7.157.2.1 operator()	724
7.157.3 Member Data Documentation	724
7.157.3.1 ostream	724
7.158option::Parser Class Reference	724
7.158.1 Detailed Description	726
7.158.2 Constructor & Destructor Documentation	726
7.158.2.1 Parser	726
7.158.2.2 Parser	726
7.158.2.3 Parser	727
7.158.2.4 Parser	727
7.158.2.5 Parser	727
7.158.3 Member Function Documentation	727
7.158.3.1 error	727
7.158.3.2 nonOption	728
7.158.3.3 nonOptions	728
7.158.3.4 nonOptionsCount	728
7.158.3.5 optionsCount	728
7.158.3.6 parse	728
7.158.3.7 parse	729
7.158.3.8 parse	729
7.158.3.9 parse	729
7.158.4 Friends And Related Function Documentation	730
7.158.4.1 Stats	730
7.159option::PrintUsageImplementation Struct Reference	730
7.159.1 Member Function Documentation	730
7.159.1.1 indent	730
7.159.1.2 isWideChar	730
7.159.1.3 printUsage	731
7.159.1.4 upmax	731
7.160protocolpp Class Reference	731
7.160.1 Detailed Description	731
7.160.2 Protocol++(ProtocolPP) Configuration Schema	731
7.161InterfacePP::ringflow Class Reference	732
7.161.1 Constructor & Destructor Documentation	732
7.161.1.1 ringflow	732
7.161.1.2 ~ringflow	732

7.161.2 Member Function Documentation	733
7.161.2.1 <code>get_addr</code>	733
7.161.2.2 <code>get_name</code>	733
7.161.2.3 <code>get_size</code>	733
7.161.2.4 <code>set_addr</code>	733
7.161.2.5 <code>set_name</code>	733
7.161.2.6 <code>set_size</code>	733
7.162InterfacePP::ringin Class Reference	734
7.162.1 Constructor & Destructor Documentation	734
7.162.1.1 <code>ringin</code>	734
7.162.1.2 <code>~ringin</code>	734
7.162.2 Member Function Documentation	734
7.162.2.1 <code>get_addr</code>	734
7.162.2.2 <code>get_outaddr</code>	735
7.162.2.3 <code>get_outlen</code>	735
7.162.2.4 <code>get_protect</code>	735
7.162.2.5 <code>get_size</code>	735
7.162.2.6 <code>get_stream</code>	735
7.162.2.7 <code>set_addr</code>	735
7.162.2.8 <code>set_outaddr</code>	735
7.162.2.9 <code>set_outlen</code>	736
7.162.2.10 <code>set_protect</code>	736
7.162.2.11 <code>set_size</code>	736
7.162.2.12 <code>set_stream</code>	736
7.163InterfacePP::ringout Class Reference	736
7.163.1 Constructor & Destructor Documentation	737
7.163.1.1 <code>ringout</code>	737
7.163.1.2 <code>~ringout</code>	737
7.163.2 Member Function Documentation	737
7.163.2.1 <code>get_addr</code>	737
7.163.2.2 <code>get_size</code>	737
7.163.2.3 <code>get_status</code>	737
7.163.2.4 <code>set_addr</code>	737
7.163.2.5 <code>set_size</code>	737
7.163.2.6 <code>set_status</code>	738
7.164InterfacePP::secin Class Reference	738
7.164.1 Constructor & Destructor Documentation	738
7.164.1.1 <code>secin</code>	738
7.164.1.2 <code>~secin</code>	738
7.164.2 Member Function Documentation	739

7.164.2.1 <code>get_addr</code>	739
7.164.2.2 <code>get_size</code>	739
7.164.2.3 <code>set_addr</code>	739
7.164.2.4 <code>set_size</code>	739
7.165 <code>InterfacePP::secout</code> Class Reference	739
7.165.1 Constructor & Destructor Documentation	740
7.165.1.1 <code>secout</code>	740
7.165.1.2 <code>~secout</code>	740
7.165.2 Member Function Documentation	740
7.165.2.1 <code>get_status</code>	740
7.165.2.2 <code>set_status</code>	740
7.166 <code>ProtocolPP::sfmt</code> Class Reference	740
7.166.1 Constructor & Destructor Documentation	741
7.166.1.1 <code>sfmt</code>	741
7.166.1.2 <code>sfmt</code>	741
7.166.1.3 <code>sfmt</code>	741
7.166.1.4 <code>~sfmt</code>	742
7.166.2 Member Function Documentation	742
7.166.2.1 <code>func1</code>	742
7.166.2.2 <code>func2</code>	742
7.166.2.3 <code>gen_rand_array</code>	742
7.166.2.4 <code>idxof</code>	742
7.166.2.5 <code>period_certification</code>	742
7.166.2.6 <code>sfmt_fill_array16</code>	743
7.166.2.7 <code>sfmt_fill_array32</code>	743
7.166.2.8 <code>sfmt_fill_array64</code>	743
7.166.2.9 <code>sfmt_fill_array8</code>	743
7.166.2.10 <code>sfmt_gen_rand_all</code>	743
7.166.2.11 <code>sfmt_genrand_real1</code>	743
7.166.2.12 <code>sfmt_genrand_real2</code>	744
7.166.2.13 <code>sfmt_genrand_real3</code>	744
7.166.2.14 <code>sfmt_genrand_res53</code>	744
7.166.2.15 <code>sfmt_genrand_res53_mix</code>	744
7.166.2.16 <code>sfmt_genrand_uint32</code>	744
7.166.2.17 <code>sfmt_genrand_uint64</code>	744
7.166.2.18 <code>sfmt_get_idstring</code>	745
7.166.2.19 <code>sfmt_get_min_array_size16</code>	745
7.166.2.20 <code>sfmt_get_min_array_size32</code>	745
7.166.2.21 <code>sfmt_get_min_array_size64</code>	745
7.166.2.22 <code>sfmt_get_min_array_size8</code>	745

7.166.2.23sfmt_init_by_array	745
7.166.2.24sfmt_init_gen_rand	745
7.166.2.25sfmt_to_real1	746
7.166.2.26sfmt_to_real2	746
7.166.2.27sfmt_to_real3	746
7.166.2.28sfmt_to_res53	746
7.166.2.29sfmt_to_res53_mix	746
7.167SFMT Class Reference	747
7.167.1 Detailed Description	747
7.168PlatformPP::jsec::sgt_t Struct Reference	748
7.168.1 Detailed Description	748
7.168.2 Constructor & Destructor Documentation	749
7.168.2.1 sgt_t	749
7.168.3 Member Data Documentation	749
7.168.3.1 bufpool	749
7.168.3.2 chunksize	749
7.168.3.3 endianess	749
7.168.3.4 maxentries	749
7.168.3.5 offset	749
7.168.3.6 ptrsize	749
7.168.3.7 sgtsize	749
7.169option::Stats Struct Reference	749
7.169.1 Detailed Description	750
7.169.2 Constructor & Destructor Documentation	750
7.169.2.1 Stats	750
7.169.2.2 Stats	750
7.169.2.3 Stats	751
7.169.2.4 Stats	751
7.169.2.5 Stats	751
7.169.3 Member Function Documentation	751
7.169.3.1 add	751
7.169.3.2 add	751
7.169.3.3 add	751
7.169.3.4 add	751
7.169.4 Member Data Documentation	751
7.169.4.1 buffer_max	751
7.169.4.2 options_max	752
7.170StdCapture Class Reference	752
7.170.1 Detailed Description	752
7.170.2 Member Function Documentation	753

7.170.2.1 BeginCapture	753
7.170.2.2 EndCapture	753
7.170.2.3 GetCapture	753
7.170.2.4 Init	753
7.170.2.5 IsCapturing	753
7.171 option::Parser::StoreOptionAction Class Reference	753
7.171.1 Constructor & Destructor Documentation	754
7.171.1.1 StoreOptionAction	754
7.171.2 Member Function Documentation	754
7.171.2.1 finished	754
7.171.2.2 perform	754
7.172 option::PrintUsageImplementation::StreamWriter< Function, Stream > Struct Template Reference	754
7.172.1 Constructor & Destructor Documentation	755
7.172.1.1 StreamWriter	755
7.172.2 Member Function Documentation	755
7.172.2.1 operator()	755
7.172.3 Member Data Documentation	755
7.172.3.1 fwrite	755
7.172.3.2 stream	755
7.173 tinyxml2::StrPair Class Reference	755
7.173.1 Member Enumeration Documentation	756
7.173.1.1 anonymous enum	756
7.173.2 Constructor & Destructor Documentation	756
7.173.2.1 StrPair	756
7.173.2.2 ~StrPair	756
7.173.3 Member Function Documentation	756
7.173.3.1 Empty	756
7.173.3.2 GetStr	756
7.173.3.3 ParseName	756
7.173.3.4 ParseText	756
7.173.3.5 Reset	756
7.173.3.6 Set	756
7.173.3.7 SetInternedStr	756
7.173.3.8 SetStr	756
7.173.3.9 TransferTo	757
7.174 option::PrintUsageImplementation::SyscallWriter< Syscall > Struct Template Reference	757
7.174.1 Constructor & Destructor Documentation	757
7.174.1.1 SyscallWriter	757
7.174.2 Member Function Documentation	757
7.174.2.1 operator()	757

7.174.3 Member Data Documentation	757
7.174.3.1 fd	757
7.174.3.2 write	757
7.175option::PrintUsageImplementation::TemporaryWriter< Temporary > Struct Template Reference	758
7.175.1 Constructor & Destructor Documentation	758
7.175.1.1 TemporaryWriter	758
7.175.2 Member Function Documentation	758
7.175.2.1 operator()	758
7.175.3 Member Data Documentation	758
7.175.3.1 userstream	758
7.176ProtocolPP::W128_T Union Reference	758
7.176.1 Detailed Description	759
7.176.2 Member Data Documentation	759
7.176.2.1 u	759
7.176.2.2 u64	759
7.177wasp Class Reference	759
7.177.1 Detailed Description	759
7.177.2 W.A.S.P	759
7.177.3 PPP Format	759
7.177.4 PROTPP Format	760
7.177.5 Using W.A.S.P	761
7.178ProtocolPP::wasp Class Reference	763
7.178.1 Constructor & Destructor Documentation	763
7.178.1.1 wasp	763
7.178.1.2 wasp	764
7.178.1.3 wasp	764
7.178.1.4 ~wasp	764
7.178.2 Member Function Documentation	764
7.178.2.1 get	764
7.179tinyxml2::XmlAttribute Class Reference	764
7.179.1 Detailed Description	766
7.179.2 Member Function Documentation	766
7.179.2.1 ArrayValue	766
7.179.2.2 BoolValue	766
7.179.2.3 ByteValue	766
7.179.2.4 DoubleValue	766
7.179.2.5 FloatValue	766
7.179.2.6 GetLineNum	767
7.179.2.7 Int64Value	767
7.179.2.8 IntValue	767

7.179.2.9 Name	767
7.179.2.10Next	767
7.179.2.11QueryArrayValue	767
7.179.2.12QueryBoolValue	767
7.179.2.13QueryByteValue	767
7.179.2.14QueryDoubleValue	767
7.179.2.15QueryFloatValue	767
7.179.2.16QueryInt64Value	767
7.179.2.17QueryIntValue	768
7.179.2.18QueryShortValue	768
7.179.2.19QueryUnsignedU64Value	768
7.179.2.20QueryUnsignedValue	768
7.179.2.21SetAttribute	768
7.179.2.22SetAttribute	768
7.179.2.23SetAttribute	768
7.179.2.24SetAttribute	768
7.179.2.25SetAttribute	768
7.179.2.26SetAttribute	768
7.179.2.27SetAttribute	768
7.179.2.28SetAttribute	769
7.179.2.29SetAttribute	769
7.179.2.30SetAttribute	769
7.179.2.31SetAttribute	769
7.179.2.32ShortValue	769
7.179.2.33UnsignedU64Value	769
7.179.2.34UnsignedValue	769
7.179.2.35Value	769
7.179.3 Friends And Related Function Documentation	769
7.179.3.1 XMLElement	769
7.180tinyxml2::XMLComment Class Reference	769
7.180.1 Detailed Description	770
7.180.2 Constructor & Destructor Documentation	770
7.180.2.1 XMLComment	770
7.180.2.2 ~XMLComment	770
7.180.3 Member Function Documentation	770
7.180.3.1 Accept	770
7.180.3.2 ParseDeep	771
7.180.3.3 ShallowClone	771
7.180.3.4 ShallowEqual	771
7.180.3.5 ToComment	771

7.180.3.6 ToComment	771
7.180.4 Friends And Related Function Documentation	771
7.180.4.1 XMLDocument	771
7.181tinyxml2::XMLConstHandle Class Reference	771
7.181.1 Detailed Description	772
7.181.2 Constructor & Destructor Documentation	772
7.181.2.1 XMLConstHandle	772
7.181.2.2 XMLConstHandle	772
7.181.2.3 XMLConstHandle	772
7.181.3 Member Function Documentation	772
7.181.3.1 FirstChild	772
7.181.3.2 FirstChildElement	772
7.181.3.3 LastChild	772
7.181.3.4 LastChildElement	772
7.181.3.5 NextSibling	772
7.181.3.6 NextSiblingElement	772
7.181.3.7 operator=	772
7.181.3.8 PreviousSibling	772
7.181.3.9 PreviousSiblingElement	773
7.181.3.10ToDeclaration	773
7.181.3.11ToElement	773
7.181.3.12ToNode	773
7.181.3.13ToText	773
7.181.3.14ToUnknown	773
7.182tinyxml2::XMLDeclaration Class Reference	773
7.182.1 Detailed Description	774
7.182.2 Constructor & Destructor Documentation	774
7.182.2.1 XMLDeclaration	774
7.182.2.2 ~XMLDeclaration	774
7.182.3 Member Function Documentation	774
7.182.3.1 Accept	774
7.182.3.2 ParseDeep	774
7.182.3.3 ShallowClone	774
7.182.3.4 ShallowEqual	775
7.182.3.5 ToDeclaration	775
7.182.3.6 ToDeclaration	775
7.182.4 Friends And Related Function Documentation	775
7.182.4.1 XMLDocument	775
7.183tinyxml2::XMLDocument Class Reference	775
7.183.1 Detailed Description	776

7.183.2 Constructor & Destructor Documentation	777
7.183.2.1 XMLDocument	777
7.183.2.2 ~XMLDocument	777
7.183.3 Member Function Documentation	777
7.183.3.1 Accept	777
7.183.3.2 Clear	777
7.183.3.3 ClearError	777
7.183.3.4 DeepCopy	777
7.183.3.5 DeleteNode	777
7.183.3.6 Error	777
7.183.3.7 ErrorCode	778
7.183.3.8 ErrorCodeName	778
7.183.3.9 ErrorLineNum	778
7.183.3.10 ErrorName	778
7.183.3.11 ErrorStr	778
7.183.3.12 HasBOM	778
7.183.3.13 Identify	778
7.183.3.14 LoadFile	778
7.183.3.15 LoadFile	778
7.183.3.16 MarkInUse	778
7.183.3.17 NewComment	778
7.183.3.18 NewDeclaration	778
7.183.3.19 NewElement	778
7.183.3.20 NewText	779
7.183.3.21 NewUnknown	779
7.183.3.22 Parse	779
7.183.3.23 Print	779
7.183.3.24 PrintError	779
7.183.3.25 ProcessEntities	779
7.183.3.26 RootElement	779
7.183.3.27 RootElement	779
7.183.3.28 SaveFile	779
7.183.3.29 SaveFile	779
7.183.3.30 SetBOM	779
7.183.3.31 ShallowClone	780
7.183.3.32 ShallowEqual	780
7.183.3.33 ToDocument	780
7.183.3.34 ToDocument	780
7.183.3.35 WhitespaceMode	780
7.183.4 Friends And Related Function Documentation	780

7.183.4.1 XMLComment	780
7.183.4.2 XMLDeclaration	780
7.183.4.3 XMLElement	780
7.183.4.4 XMLNode	780
7.183.4.5 XMLText	780
7.183.4.6 XMLUnknown	780
7.184tinyxml2::XMLElement Class Reference	780
7.184.1 Detailed Description	784
7.184.2 Member Enumeration Documentation	784
7.184.2.1 ElementClosingType	784
7.184.3 Member Function Documentation	784
7.184.3.1 Accept	784
7.184.3.2 ArrayAttribute	784
7.184.3.3 Attribute	784
7.184.3.4 BoolAttribute	785
7.184.3.5 BoolText	785
7.184.3.6 ByteAttribute	785
7.184.3.7 ClosingType	785
7.184.3.8 DeleteAttribute	785
7.184.3.9 DoubleAttribute	785
7.184.3.10DoubleText	785
7.184.3.11FindAttribute	785
7.184.3.12FirstAttribute	785
7.184.3.13FloatAttribute	785
7.184.3.14FloatText	786
7.184.3.15GetText	786
7.184.3.16Int64Attribute	786
7.184.3.17Int64Text	786
7.184.3.18IntAttribute	786
7.184.3.19IntText	786
7.184.3.20Name	786
7.184.3.21ParseDeep	786
7.184.3.22QueryArrayAttribute	787
7.184.3.23QueryArrayText	787
7.184.3.24QueryAttribute	787
7.184.3.25QueryAttribute	787
7.184.3.26QueryAttribute	787
7.184.3.27QueryAttribute	788
7.184.3.28QueryAttribute	788
7.184.3.29QueryAttribute	788

7.184.3.30QueryAttribute	788
7.184.3.31QueryAttribute	788
7.184.3.32QueryAttribute	788
7.184.3.33QueryAttribute	788
7.184.3.34QueryBoolAttribute	788
7.184.3.35QueryBoolText	788
7.184.3.36QueryByteAttribute	788
7.184.3.37QueryByteText	789
7.184.3.38QueryDoubleAttribute	789
7.184.3.39QueryDoubleText	789
7.184.3.40QueryFloatAttribute	789
7.184.3.41QueryFloatText	789
7.184.3.42QueryInt64Attribute	789
7.184.3.43QueryInt64Text	789
7.184.3.44QueryIntAttribute	789
7.184.3.45QueryIntText	789
7.184.3.46QueryShortAttribute	790
7.184.3.47QueryShortText	790
7.184.3.48QueryStringAttribute	790
7.184.3.49QueryUnsignedAttribute	790
7.184.3.50QueryUnsignedText	790
7.184.3.51QueryUnsignedU64Attribute	790
7.184.3.52QueryUnsignedU64Text	791
7.184.3.53SetAttribute	791
7.184.3.54SetAttribute	791
7.184.3.55SetAttribute	791
7.184.3.56SetAttribute	791
7.184.3.57SetAttribute	791
7.184.3.58SetAttribute	791
7.184.3.59SetAttribute	791
7.184.3.60SetAttribute	791
7.184.3.61SetAttribute	791
7.184.3.62SetAttribute	791
7.184.3.63SetName	792
7.184.3.64SetText	792
7.184.3.65SetText	792
7.184.3.66SetText	792
7.184.3.67SetText	792
7.184.3.68SetText	792
7.184.3.69SetText	792

7.184.3.70SetText	793
7.184.3.71SetText	793
7.184.3.72SetText	793
7.184.3.73SetText	793
7.184.3.74SetText	793
7.184.3.75ShallowClone	793
7.184.3.76ShallowEqual	793
7.184.3.77ShortAttribute	793
7.184.3.78ToElement	793
7.184.3.79ToElement	793
7.184.3.80UnsignedAttribute	794
7.184.3.81UnsignedText	794
7.184.3.82UnsignedU64Attribute	794
7.184.4 Friends And Related Function Documentation	794
7.184.4.1 XMLDocument	794
7.185tinyxml2::XMLHandle Class Reference	794
7.185.1 Detailed Description	795
7.185.2 Constructor & Destructor Documentation	796
7.185.2.1 XMLHandle	796
7.185.2.2 XMLHandle	796
7.185.2.3 XMLHandle	796
7.185.3 Member Function Documentation	796
7.185.3.1 FirstChild	796
7.185.3.2 FirstChildElement	796
7.185.3.3 LastChild	796
7.185.3.4 LastChildElement	796
7.185.3.5 NextSibling	796
7.185.3.6 NextSiblingElement	796
7.185.3.7 operator=	796
7.185.3.8 PreviousSibling	796
7.185.3.9 PreviousSiblingElement	797
7.185.3.10ToDeclaration	797
7.185.3.11ToElement	797
7.185.3.12ToNode	797
7.185.3.13ToText	797
7.185.3.14ToUnknown	797
7.186tinyxml2::XMLNode Class Reference	797
7.186.1 Detailed Description	799
7.186.2 Constructor & Destructor Documentation	799
7.186.2.1 XMLNode	799

7.186.2.2 ~XMLNode	799
7.186.3 Member Function Documentation	799
7.186.3.1 Accept	799
7.186.3.2 DeepClone	800
7.186.3.3 DeleteChild	800
7.186.3.4 DeleteChildren	800
7.186.3.5 FirstChild	800
7.186.3.6 FirstChild	800
7.186.3.7 FirstChildElement	800
7.186.3.8 FirstChildElement	800
7.186.3.9 GetDocument	800
7.186.3.10 GetDocument	801
7.186.3.11 GetLineNum	801
7.186.3.12 GetUserData	801
7.186.3.13 InsertAfterChild	801
7.186.3.14 InsertEndChild	801
7.186.3.15 InsertFirstChild	801
7.186.3.16 LastChild	801
7.186.3.17 LastChild	801
7.186.3.18 LastChildElement	801
7.186.3.19 LastChildElement	801
7.186.3.20 LinkEndChild	801
7.186.3.21 NextSibling	801
7.186.3.22 NextSibling	802
7.186.3.23 NextSiblingElement	802
7.186.3.24 NextSiblingElement	802
7.186.3.25 NoChildren	802
7.186.3.26 Parent	802
7.186.3.27 Parent	802
7.186.3.28 ParseDeep	802
7.186.3.29 PreviousSibling	802
7.186.3.30 PreviousSibling	802
7.186.3.31 PreviousSiblingElement	802
7.186.3.32 PreviousSiblingElement	802
7.186.3.33 SetUserData	802
7.186.3.34 SetValue	802
7.186.3.35 ShallowClone	803
7.186.3.36 ShallowEqual	803
7.186.3.37 ToComment	803
7.186.3.38 ToComment	803

7.186.3.39ToDeclaration	803
7.186.3.40ToDeclaration	803
7.186.3.41ToDocument	803
7.186.3.42ToDocument	803
7.186.3.43ToElement	803
7.186.3.44ToElement	804
7.186.3.45ToText	804
7.186.3.46ToText	804
7.186.3.47ToUnknown	804
7.186.3.48ToUnknown	804
7.186.3.49Value	804
7.186.4 Friends And Related Function Documentation	804
7.186.4.1 XMLDocument	804
7.186.4.2 XMLElement	804
7.186.5 Member Data Documentation	804
7.186.5.1 _document	804
7.186.5.2 _firstChild	804
7.186.5.3 _lastChild	804
7.186.5.4 _next	804
7.186.5.5 _parent	804
7.186.5.6 _parseLineNum	804
7.186.5.7 _prev	805
7.186.5.8 _userData	805
7.186.5.9 _value	805
7.187tinyxml2::XMLPrinter Class Reference	805
7.187.1 Detailed Description	807
7.187.2 Constructor & Destructor Documentation	807
7.187.2.1 XMLPrinter	807
7.187.2.2 XMLPrinter	807
7.187.2.3 ~XMLPrinter	808
7.187.3 Member Function Documentation	808
7.187.3.1 ClearBuffer	808
7.187.3.2 CloseElement	808
7.187.3.3 CompactMode	808
7.187.3.4 CStr	808
7.187.3.5 CStrSize	808
7.187.3.6 OpenElement	808
7.187.3.7 Print	808
7.187.3.8 PrintSpace	808
7.187.3.9 PushAttribute	808

7.187.3.10PushAttribute	808
7.187.3.11PushAttribute	808
7.187.3.12PushAttribute	808
7.187.3.13PushAttribute	809
7.187.3.14PushAttribute	809
7.187.3.15PushAttribute	809
7.187.3.16PushAttribute	809
7.187.3.17PushAttribute	809
7.187.3.18PushAttribute	809
7.187.3.19PushComment	809
7.187.3.20PushDeclaration	809
7.187.3.21PushHeader	809
7.187.3.22PushText	809
7.187.3.23PushText	809
7.187.3.24PushText	809
7.187.3.25PushText	809
7.187.3.26PushText	809
7.187.3.27PushText	809
7.187.3.28PushText	810
7.187.3.29PushText	810
7.187.3.30PushText	810
7.187.3.31PushText	810
7.187.3.32PushText	810
7.187.3.33PushUnknown	810
7.187.3.34Putc	810
7.187.3.35SealElementIfJustOpened	810
7.187.3.36Visit	810
7.187.3.37Visit	810
7.187.3.38Visit	810
7.187.3.39Visit	810
7.187.3.40VisitEnter	811
7.187.3.41VisitEnter	811
7.187.3.42VisitExit	811
7.187.3.43VisitExit	811
7.187.3.44Write	811
7.187.3.45Write	811
7.187.4 Member Data Documentation	811
7.187.4.1 _elementJustOpened	811
7.187.4.2 _stack	811
7.188tinyxml2::XMLText Class Reference	811

7.188.1 Detailed Description	812
7.188.2 Constructor & Destructor Documentation	812
7.188.2.1 XMLText	812
7.188.2.2 ~XMLText	812
7.188.3 Member Function Documentation	812
7.188.3.1 Accept	812
7.188.3.2 CData	813
7.188.3.3 ParseDeep	813
7.188.3.4 SetCData	813
7.188.3.5 ShallowClone	813
7.188.3.6 ShallowEqual	813
7.188.3.7 ToText	813
7.188.3.8 ToText	813
7.188.4 Friends And Related Function Documentation	813
7.188.4.1 XMLDocument	813
7.189tinyxml2::XMLUnknown Class Reference	814
7.189.1 Detailed Description	814
7.189.2 Constructor & Destructor Documentation	814
7.189.2.1 XMLUnknown	814
7.189.2.2 ~XMLUnknown	814
7.189.3 Member Function Documentation	814
7.189.3.1 Accept	815
7.189.3.2 ParseDeep	815
7.189.3.3 ShallowClone	815
7.189.3.4 ShallowEqual	815
7.189.3.5 ToUnknown	815
7.189.3.6 ToUnknown	815
7.189.4 Friends And Related Function Documentation	816
7.189.4.1 XMLDocument	816
7.190tinyxml2::XMLUtil Class Reference	816
7.190.1 Member Function Documentation	817
7.190.1.1 ConvertUTF32ToUTF8	817
7.190.1.2 GetCharacterRef	817
7.190.1.3 IsNameChar	817
7.190.1.4 IsNameStartChar	817
7.190.1.5 IsUTF8Continuation	817
7.190.1.6 IsWhiteSpace	817
7.190.1.7 ReadBOM	817
7.190.1.8 SetBoolSerialization	817
7.190.1.9 SkipWhiteSpace	817

7.190.1.10SkipWhiteSpace	817
7.190.1.11StringEqual	817
7.190.1.12ToArray	817
7.190.1.13ToBool	817
7.190.1.14ToByte	817
7.190.1.15.ToDouble	817
7.190.1.16ToFloat	817
7.190.1.17ToInt	817
7.190.1.18ToInt64	817
7.190.1.19.ToShort	817
7.190.1.20ToStr	817
7.190.1.21ToStr	817
7.190.1.22ToStr	817
7.190.1.23ToStr	817
7.190.1.24ToStr	817
7.190.1.25ToStr	817
7.190.1.26ToStr	818
7.190.1.27ToStr	818
7.190.1.28ToStr	818
7.190.1.29ToStr	818
7.190.1.30ToUnsigned	818
7.190.1.31ToUnsignedU64	818
7.191tinyxml2::XMLVisitor Class Reference	818
7.191.1 Detailed Description	819
7.191.2 Constructor & Destructor Documentation	819
7.191.2.1 ~XMLVisitor	819
7.191.3 Member Function Documentation	819
7.191.3.1 Visit	819
7.191.3.2 Visit	819
7.191.3.3 Visit	819
7.191.3.4 Visit	819
7.191.3.5 VisitEnter	819
7.191.3.6 VisitEnter	820
7.191.3.7 VisitExit	820
7.191.3.8 VisitExit	820
8 File Documentation	829
8.1 drivers/include/jdirectdrive.h File Reference	829
8.2 drivers/include/jdrive.h File Reference	830
8.3 drivers/include/jdriver.h File Reference	830

8.4	drivers/include/jringdrive.h File Reference	831
8.5	drivers/include/StdCapture.h File Reference	832
8.6	include/aead_chacha_poly1305.h File Reference	832
8.7	include/chacha20.h File Reference	832
8.8	include/ciphers.h File Reference	833
8.9	include/config.h File Reference	833
8.9.1	Detailed Description	834
8.9.2	Macro Definition Documentation	835
8.9.2.1	ANONYMOUS_NAMESPACE_BEGIN	835
8.9.2.2	ANONYMOUS_NAMESPACE_END	835
8.9.2.3	CPP_TYPENAME	835
8.9.2.4	DOCUMENTED_NAMESPACE_BEGIN	835
8.9.2.5	DOCUMENTED_NAMESPACE_END	835
8.9.2.6	DOCUMENTED_TYPEDEF	835
8.9.2.7	GZIP_OS_CODE	835
8.9.2.8	IS_LITTLE_ENDIAN	835
8.9.2.9	NAMESPACE_BEGIN	835
8.9.2.10	NAMESPACE_END	835
8.9.2.11	NO_OS_DEPENDENCE	835
8.9.2.12	PREFER_BERKELEY_STYLE_SOCKETS	835
8.9.2.13	PROTOCOLPP_ALIGN_DATA	835
8.9.2.14	PROTOCOLPP_API	835
8.9.2.15	PROTOCOLPP_BOOL_AESNI_INTRINSICS_AVAILABLE	835
8.9.2.16	PROTOCOLPP_BOOL_ALIGN16	835
8.9.2.17	PROTOCOLPP_BOOL_ARM32	835
8.9.2.18	PROTOCOLPP_BOOL_ARM64	835
8.9.2.19	PROTOCOLPP_BOOL_AVX_AVAILABLE	835
8.9.2.20	PROTOCOLPP_BOOL_SLOW_WORD64	835
8.9.2.21	PROTOCOLPP_BOOL_SSE2_INTRINSICS_AVAILABLE	835
8.9.2.22	PROTOCOLPP_BOOL_SSE4_INTRINSICS_AVAILABLE	835
8.9.2.23	PROTOCOLPP_BOOL_X32	835
8.9.2.24	PROTOCOLPP_BOOL_X64	835
8.9.2.25	PROTOCOLPP_BOOL_X86	835
8.9.2.26	PROTOCOLPP_CONSTANT	835
8.9.2.27	PROTOCOLPP_CONSTEXPR	835
8.9.2.28	PROTOCOLPP_DATA_DIR	836
8.9.2.29	PROTOCOLPP_DEPRECATED	836
8.9.2.30	PROTOCOLPP_DLL	836
8.9.2.31	PROTOCOLPP_DLL_TEMPLATE_CLASS	836
8.9.2.32	PROTOCOLPP_EXTERN_DLL_TEMPLATE_CLASS	836

8.9.2.33	PROTOCOLPP_EXTERN_STATIC_TEMPLATE_CLASS	836
8.9.2.34	PROTOCOLPP_FASTCALL	836
8.9.2.35	PROTOCOLPP_INIT_PRIORITY	836
8.9.2.36	PROTOCOLPP_L1_CACHE_LINE_SIZE	836
8.9.2.37	PROTOCOLPP_NATIVE_DWORD_AVAILABLE	836
8.9.2.38	PROTOCOLPP_NO_ALIGNED_ALLOC	836
8.9.2.39	PROTOCOLPP_NO_THROW	836
8.9.2.40	PROTOCOLPP_NO_UNALIGNED_DATA_ACCESS	836
8.9.2.41	PROTOCOLPP_NO_VTABLE	836
8.9.2.42	PROTOCOLPP_NOINLINE	836
8.9.2.43	PROTOCOLPP_NOINLINE_DOTDOTDOT	836
8.9.2.44	PROTOCOLPP_RIJNDAEL_NAME	836
8.9.2.45	PROTOCOLPP_SECTION_ALIGN16	836
8.9.2.46	PROTOCOLPP_SECTION_INIT	836
8.9.2.47	PROTOCOLPP_STATIC_TEMPLATE_CLASS	836
8.9.2.48	PROTOCOLPP_THROW	836
8.9.2.49	PROTOCOLPP_UNCAUGHT_EXCEPTION_AVAILABLE	836
8.9.2.50	PROTOCOLPP_UNUSED	836
8.9.2.51	PROTOCOLPP_USER_PRIORITY	836
8.9.2.52	PROTOCOLPP_VC6_INT64	836
8.9.2.53	PROTOCOLPP_VERSION	836
8.9.2.54	TYPE_OF SOCKLEN_T	836
8.9.2.55	USE_MS_CRYPTOAPI	836
8.9.2.56	USING_NAMESPACE	837
8.9.2.57	W64LIT	837
8.9.2.58	WORKAROUND_MS_BUG_Q258000	837
8.9.3	Typedef Documentation	837
8.9.3.1	byte	837
8.9.3.2	dword	837
8.9.3.3	hword	837
8.9.3.4	lword	837
8.9.3.5	word	837
8.9.3.6	word16	837
8.9.3.7	word32	837
8.9.3.8	word64	837
8.9.4	Variable Documentation	837
8.9.4.1	LWORD_MAX	837
8.9.4.2	WORD_BITS	837
8.9.4.3	WORD_SIZE	837
8.10	include/EnumString.h File Reference	837

8.10.1 Macro Definition Documentation	838
8.10.1.1 Begin_Enum_String	838
8.10.1.2 End_Enum_String	838
8.10.1.3 Enum_String	838
8.11 include/jarray.h File Reference	838
8.12 include/jdata.h File Reference	838
8.13 include/jdsa.h File Reference	839
8.14 include/jecdsa.h File Reference	839
8.14.1 Macro Definition Documentation	840
8.14.1.1 CRYPTOPP_ENABLE_NAMESPACE_WEAK	840
8.15 include/jenum.h File Reference	840
8.15.1 Macro Definition Documentation	853
8.15.1.1 HEX	853
8.15.1.2 HEXLOG	854
8.16 include/jicmp.h File Reference	854
8.17 include/jicmpsa.h File Reference	854
8.18 include/jip.h File Reference	854
8.19 include/jipsa.h File Reference	855
8.20 include/jipsec.h File Reference	855
8.21 include/jipsecsa.h File Reference	855
8.22 include/jlte.h File Reference	855
8.23 include/jltesa.h File Reference	856
8.24 include/jmacsec.h File Reference	856
8.25 include/jmacsecsa.h File Reference	856
8.26 include/jmodes.h File Reference	857
8.26.1 Macro Definition Documentation	858
8.26.1.1 CRYPTOPP_ENABLE_NAMESPACE_WEAK	858
8.26.2 Typedef Documentation	858
8.26.2.1 Byte	858
8.26.2.2 Word	858
8.27 include/jpacket.h File Reference	858
8.28 include/jpoly1305.h File Reference	858
8.28.1 Macro Definition Documentation	859
8.28.1.1 HAS_GCC_4_4_64BIT	859
8.28.1.2 HAS_MSVC_64BIT	859
8.28.1.3 HAS_SIZEOF_INT128_64BIT	859
8.28.1.4 JPOLY1305_32BIT	859
8.29 include/jprotocol.h File Reference	859
8.30 include/jprotocolpp.h File Reference	860
8.30.1 Macro Definition Documentation	860

8.30.1.1	PROTOCOLPP_VERSION	860
8.31	include/jrand.h File Reference	860
8.32	include/jreplay.h File Reference	861
8.33	include/jrsa.h File Reference	861
8.34	include/jsecass.h File Reference	862
8.35	include/jsnow3g.h File Reference	862
8.36	include/jsrtp.h File Reference	862
8.37	include/jsrpsa.h File Reference	863
8.38	include/jstream.h File Reference	863
8.39	include/jtcp.h File Reference	863
8.40	include/jtcpsa.h File Reference	863
8.41	include/jtls.h File Reference	864
8.42	include/jtsa.h File Reference	864
8.43	include/judp.h File Reference	864
8.44	include/judpsa.h File Reference	865
8.45	include/jwifi.h File Reference	865
8.46	include/jwifisa.h File Reference	865
8.47	include/jwimax.h File Reference	866
8.48	include/jwimaxsa.h File Reference	866
8.49	include/jzuc.h File Reference	866
8.49.1	Macro Definition Documentation	867
8.49.1.1	MAKEU31	867
8.49.1.2	MAKEuint32_t	867
8.49.1.3	MulByPow2	867
8.49.1.4	ROT	867
8.50	include/mtrand.h File Reference	867
8.51	include/poly1305-16.h File Reference	867
8.51.1	Macro Definition Documentation	868
8.51.1.1	jpoly1305_block_size	868
8.51.1.2	JPOLY1305_NOINLINE	868
8.52	include/poly1305-32.h File Reference	868
8.52.1	Macro Definition Documentation	868
8.52.1.1	jpoly1305_block_size	868
8.52.1.2	JPOLY1305_NOINLINE	868
8.53	include/poly1305-64.h File Reference	868
8.53.1	Macro Definition Documentation	869
8.53.1.1	jpoly1305_block_size	869
8.54	include/poly1305-8.h File Reference	869
8.54.1	Macro Definition Documentation	869
8.54.1.1	jpoly1305_block_size	869

8.54.1.2	JPOLY1305_NOINLINE	869
8.55	include/SFMT-alti.h File Reference	869
8.55.1	Detailed Description	870
8.55.2	Macro Definition Documentation	870
8.55.2.1	SFMT_ALTI_H	870
8.55.2.2	SFMT_ALTI_SWAP	870
8.56	include/SFMT-common.h File Reference	870
8.56.1	Detailed Description	870
8.56.2	Macro Definition Documentation	871
8.56.2.1	SFMT_COMMON_H	871
8.57	include/SFMT-neon.h File Reference	871
8.57.1	Detailed Description	871
8.57.2	Macro Definition Documentation	871
8.57.2.1	rotate_bytes	871
8.58	include/SFMT-params.h File Reference	871
8.58.1	Macro Definition Documentation	872
8.58.1.1	SFMT_MEXP	872
8.58.1.2	SFMT_N	872
8.58.1.3	SFMT_N16	872
8.58.1.4	SFMT_N32	872
8.58.1.5	SFMT_N64	872
8.58.1.6	SFMT_N8	872
8.58.1.7	SFMT_PARAMS_H	872
8.59	include/SFMT-params11213.h File Reference	872
8.59.1	Macro Definition Documentation	873
8.59.1.1	SFMT_ALTI_MSK	873
8.59.1.2	SFMT_ALTI_MSK64	873
8.59.1.3	SFMT_ALTI_SL1	873
8.59.1.4	SFMT_ALTI_SL2_PERM	873
8.59.1.5	SFMT_ALTI_SL2_PERM64	873
8.59.1.6	SFMT_ALTI_SR1	873
8.59.1.7	SFMT_ALTI_SR2_PERM	873
8.59.1.8	SFMT_ALTI_SR2_PERM64	873
8.59.1.9	SFMT_IDSTR	873
8.59.1.10	SFMT_MSK1	873
8.59.1.11	SFMT_MSK2	873
8.59.1.12	SFMT_MSK3	873
8.59.1.13	SFMT_MSK4	873
8.59.1.14	SFMT_PARAMS11213_H	873
8.59.1.15	SFMT_PARITY1	874

8.59.1.16 SFMT_PARITY2	874
8.59.1.17 SFMT_PARITY3	874
8.59.1.18 SFMT_PARITY4	874
8.59.1.19 SFMT_POS1	874
8.59.1.20 SFMT_SL1	874
8.59.1.21 SFMT_SL2	874
8.59.1.22 SFMT_SR1	874
8.59.1.23 SFMT_SR2	874
8.60 include/SFMT-params1279.h File Reference	874
8.60.1 Macro Definition Documentation	875
8.60.1.1 SFMT_ALTI_MSK	875
8.60.1.2 SFMT_ALTI_MSK64	875
8.60.1.3 SFMT_ALTI_SL1	875
8.60.1.4 SFMT_ALTI_SL2_PERM	875
8.60.1.5 SFMT_ALTI_SL2_PERM64	875
8.60.1.6 SFMT_ALTI_SR1	875
8.60.1.7 SFMT_ALTI_SR2_PERM	875
8.60.1.8 SFMT_ALTI_SR2_PERM64	875
8.60.1.9 SFMT_IDSTR	875
8.60.1.10 SFMT_MSK1	875
8.60.1.11 SFMT_MSK2	875
8.60.1.12 SFMT_MSK3	875
8.60.1.13 SFMT_MSK4	875
8.60.1.14 SFMT_PARAMS1279_H	875
8.60.1.15 SFMT_PARITY1	875
8.60.1.16 SFMT_PARITY2	875
8.60.1.17 SFMT_PARITY3	875
8.60.1.18 SFMT_PARITY4	875
8.60.1.19 SFMT_POS1	875
8.60.1.20 SFMT_SL1	875
8.60.1.21 SFMT_SL2	875
8.60.1.22 SFMT_SR1	875
8.60.1.23 SFMT_SR2	875
8.61 include/SFMT-params132049.h File Reference	875
8.61.1 Macro Definition Documentation	876
8.61.1.1 SFMT_ALTI_MSK	876
8.61.1.2 SFMT_ALTI_MSK64	876
8.61.1.3 SFMT_ALTI_SL1	876
8.61.1.4 SFMT_ALTI_SL2_PERM	876
8.61.1.5 SFMT_ALTI_SL2_PERM64	876

8.61.1.6 SFMT_ALTI_SR1	876
8.61.1.7 SFMT_ALTI_SR2_PERM	876
8.61.1.8 SFMT_ALTI_SR2_PERM64	876
8.61.1.9 SFMT_IDSTR	876
8.61.1.10 SFMT_MSK1	876
8.61.1.11 SFMT_MSK2	876
8.61.1.12 SFMT_MSK3	876
8.61.1.13 SFMT_MSK4	876
8.61.1.14 SFMT_PARAMS132049_H	876
8.61.1.15 SFMT_PARITY1	877
8.61.1.16 SFMT_PARITY2	877
8.61.1.17 SFMT_PARITY3	877
8.61.1.18 SFMT_PARITY4	877
8.61.1.19 SFMT_POS1	877
8.61.1.20 SFMT_SL1	877
8.61.1.21 SFMT_SL2	877
8.61.1.22 SFMT_SR1	877
8.61.1.23 SFMT_SR2	877
8.62 include/SFMT-params19937.h File Reference	877
8.62.1 Macro Definition Documentation	878
8.62.1.1 SFMT_ALTI_MSK	878
8.62.1.2 SFMT_ALTI_MSK64	878
8.62.1.3 SFMT_ALTI_SL1	878
8.62.1.4 SFMT_ALTI_SL2_PERM	878
8.62.1.5 SFMT_ALTI_SL2_PERM64	878
8.62.1.6 SFMT_ALTI_SR1	878
8.62.1.7 SFMT_ALTI_SR2_PERM	878
8.62.1.8 SFMT_ALTI_SR2_PERM64	878
8.62.1.9 SFMT_IDSTR	878
8.62.1.10 SFMT_MSK1	878
8.62.1.11 SFMT_MSK2	878
8.62.1.12 SFMT_MSK3	878
8.62.1.13 SFMT_MSK4	878
8.62.1.14 SFMT_PARAMS19937_H	878
8.62.1.15 SFMT_PARITY1	878
8.62.1.16 SFMT_PARITY2	878
8.62.1.17 SFMT_PARITY3	878
8.62.1.18 SFMT_PARITY4	878
8.62.1.19 SFMT_POS1	878
8.62.1.20 SFMT_SL1	878

8.62.1.21 SFMT_SL2	878
8.62.1.22 SFMT_SR1	878
8.62.1.23 SFMT_SR2	878
8.63 include/SFMT-params216091.h File Reference	878
8.63.1 Macro Definition Documentation	879
8.63.1.1 SFMT_ALTI_MSK	879
8.63.1.2 SFMT_ALTI_MSK64	879
8.63.1.3 SFMT_ALTI_SL1	879
8.63.1.4 SFMT_ALTI_SL2_PERM	879
8.63.1.5 SFMT_ALTI_SL2_PERM64	879
8.63.1.6 SFMT_ALTI_SR1	879
8.63.1.7 SFMT_ALTI_SR2_PERM	879
8.63.1.8 SFMT_ALTI_SR2_PERM64	879
8.63.1.9 SFMT_IDSTR	879
8.63.1.10 SFMT_MSK1	879
8.63.1.11 SFMT_MSK2	879
8.63.1.12 SFMT_MSK3	879
8.63.1.13 SFMT_MSK4	879
8.63.1.14 SFMT_PARAMS216091_H	879
8.63.1.15 SFMT_PARITY1	880
8.63.1.16 SFMT_PARITY2	880
8.63.1.17 SFMT_PARITY3	880
8.63.1.18 SFMT_PARITY4	880
8.63.1.19 SFMT_POS1	880
8.63.1.20 SFMT_SL1	880
8.63.1.21 SFMT_SL2	880
8.63.1.22 SFMT_SR1	880
8.63.1.23 SFMT_SR2	880
8.64 include/SFMT-params2281.h File Reference	880
8.64.1 Macro Definition Documentation	881
8.64.1.1 SFMT_ALTI_MSK	881
8.64.1.2 SFMT_ALTI_MSK64	881
8.64.1.3 SFMT_ALTI_SL1	881
8.64.1.4 SFMT_ALTI_SL2_PERM	881
8.64.1.5 SFMT_ALTI_SL2_PERM64	881
8.64.1.6 SFMT_ALTI_SR1	881
8.64.1.7 SFMT_ALTI_SR2_PERM	881
8.64.1.8 SFMT_ALTI_SR2_PERM64	881
8.64.1.9 SFMT_IDSTR	881
8.64.1.10 SFMT_MSK1	881

8.64.1.11 SFMT_MSK2	881
8.64.1.12 SFMT_MSK3	881
8.64.1.13 SFMT_MSK4	881
8.64.1.14 SFMT_PARAMS2281_H	881
8.64.1.15 SFMT_PARITY1	881
8.64.1.16 SFMT_PARITY2	881
8.64.1.17 SFMT_PARITY3	881
8.64.1.18 SFMT_PARITY4	881
8.64.1.19 SFMT_POS1	881
8.64.1.20 SFMT_SL1	881
8.64.1.21 SFMT_SL2	881
8.64.1.22 SFMT_SR1	881
8.64.1.23 SFMT_SR2	881
8.65 include/SFMT-params4253.h File Reference	881
8.65.1 Macro Definition Documentation	882
8.65.1.1 SFMT_ALTI_MSK	882
8.65.1.2 SFMT_ALTI_MSK64	882
8.65.1.3 SFMT_ALTI_SL1	882
8.65.1.4 SFMT_ALTI_SL2_PERM	882
8.65.1.5 SFMT_ALTI_SL2_PERM64	882
8.65.1.6 SFMT_ALTI_SR1	882
8.65.1.7 SFMT_ALTI_SR2_PERM	882
8.65.1.8 SFMT_ALTI_SR2_PERM64	882
8.65.1.9 SFMT_IDSTR	882
8.65.1.10 SFMT_MSK1	882
8.65.1.11 SFMT_MSK2	882
8.65.1.12 SFMT_MSK3	882
8.65.1.13 SFMT_MSK4	882
8.65.1.14 SFMT_PARAMS4253_H	882
8.65.1.15 SFMT_PARITY1	883
8.65.1.16 SFMT_PARITY2	883
8.65.1.17 SFMT_PARITY3	883
8.65.1.18 SFMT_PARITY4	883
8.65.1.19 SFMT_POS1	883
8.65.1.20 SFMT_SL1	883
8.65.1.21 SFMT_SL2	883
8.65.1.22 SFMT_SR1	883
8.65.1.23 SFMT_SR2	883
8.66 include/SFMT-params44497.h File Reference	883
8.66.1 Macro Definition Documentation	884

8.66.1.1 SFMT_ALTI_MSK	884
8.66.1.2 SFMT_ALTI_MSK64	884
8.66.1.3 SFMT_ALTI_SL1	884
8.66.1.4 SFMT_ALTI_SL2_PERM	884
8.66.1.5 SFMT_ALTI_SL2_PERM64	884
8.66.1.6 SFMT_ALTI_SR1	884
8.66.1.7 SFMT_ALTI_SR2_PERM	884
8.66.1.8 SFMT_ALTI_SR2_PERM64	884
8.66.1.9 SFMT_IDSTR	884
8.66.1.10 SFMT_MSK1	884
8.66.1.11 SFMT_MSK2	884
8.66.1.12 SFMT_MSK3	884
8.66.1.13 SFMT_MSK4	884
8.66.1.14 SFMT_PARAMS44497_H	884
8.66.1.15 SFMT_PARITY1	884
8.66.1.16 SFMT_PARITY2	884
8.66.1.17 SFMT_PARITY3	884
8.66.1.18 SFMT_PARITY4	884
8.66.1.19 SFMT_POS1	884
8.66.1.20 SFMT_SL1	884
8.66.1.21 SFMT_SL2	884
8.66.1.22 SFMT_SR1	884
8.66.1.23 SFMT_SR2	884
8.67 include/SFMT-params607.h File Reference	884
8.67.1 Macro Definition Documentation	885
8.67.1.1 SFMT_ALTI_MSK	885
8.67.1.2 SFMT_ALTI_MSK64	885
8.67.1.3 SFMT_ALTI_SL1	885
8.67.1.4 SFMT_ALTI_SL2_PERM	885
8.67.1.5 SFMT_ALTI_SL2_PERM64	885
8.67.1.6 SFMT_ALTI_SR1	885
8.67.1.7 SFMT_ALTI_SR2_PERM	885
8.67.1.8 SFMT_ALTI_SR2_PERM64	885
8.67.1.9 SFMT_IDSTR	885
8.67.1.10 SFMT_MSK1	885
8.67.1.11 SFMT_MSK2	885
8.67.1.12 SFMT_MSK3	885
8.67.1.13 SFMT_MSK4	885
8.67.1.14 SFMT_PARAMS607_H	885
8.67.1.15 SFMT_PARITY1	886

8.67.1.16 SFMT_PARITY2	886
8.67.1.17 SFMT_PARITY3	886
8.67.1.18 SFMT_PARITY4	886
8.67.1.19 SFMT_POS1	886
8.67.1.20 SFMT_SL1	886
8.67.1.21 SFMT_SL2	886
8.67.1.22 SFMT_SR1	886
8.67.1.23 SFMT_SR2	886
8.68 include/SFMT-params86243.h File Reference	886
8.68.1 Macro Definition Documentation	887
8.68.1.1 SFMT_ALTI_MSK	887
8.68.1.2 SFMT_ALTI_MSK64	887
8.68.1.3 SFMT_ALTI_SL1	887
8.68.1.4 SFMT_ALTI_SL2_PERM	887
8.68.1.5 SFMT_ALTI_SL2_PERM64	887
8.68.1.6 SFMT_ALTI_SR1	887
8.68.1.7 SFMT_ALTI_SR2_PERM	887
8.68.1.8 SFMT_ALTI_SR2_PERM64	887
8.68.1.9 SFMT_IDSTR	887
8.68.1.10 SFMT_MSK1	887
8.68.1.11 SFMT_MSK2	887
8.68.1.12 SFMT_MSK3	887
8.68.1.13 SFMT_MSK4	887
8.68.1.14 SFMT_PARAMS86243_H	887
8.68.1.15 SFMT_PARITY1	887
8.68.1.16 SFMT_PARITY2	887
8.68.1.17 SFMT_PARITY3	887
8.68.1.18 SFMT_PARITY4	887
8.68.1.19 SFMT_POS1	887
8.68.1.20 SFMT_SL1	887
8.68.1.21 SFMT_SL2	887
8.68.1.22 SFMT_SR1	887
8.68.1.23 SFMT_SR2	887
8.69 include/SFMT-sse2-msc.h File Reference	887
8.69.1 Detailed Description	888
8.69.2 Macro Definition Documentation	888
8.69.2.1 SFMT_SSE2_MSC_H	888
8.70 include/SFMT-sse2.h File Reference	888
8.70.1 Detailed Description	889
8.70.2 Macro Definition Documentation	889

8.70.2.1 SFMT_SSE2_H	889
8.71 include/SFMT.h File Reference	889
8.71.1 Macro Definition Documentation	890
8.71.1.1 PRlu64	890
8.71.1.2 PRIx64	890
8.71.1.3 SFMT_MEXP	890
8.72 include/tinyxml2.h File Reference	890
8.72.1 Macro Definition Documentation	891
8.72.1.1 HAS_GCC_4_4_64BIT	891
8.72.1.2 HAS_MSVC_64BIT	891
8.72.1.3 HAS_SIZEOF_INT128_64BIT	891
8.72.1.4 TINYXML2_LIB	892
8.72.1.5 TINYXML2_MAJOR_VERSION	892
8.72.1.6 TINYXML2_MINOR_VERSION	892
8.72.1.7 TINYXML2_PATCH_VERSION	892
8.72.1.8 TIXMLASSERT	892
8.72.2 Variable Documentation	892
8.72.2.1 TINYXML2_MAX_ELEMENT_DEPTH	892
8.72.2.2 TIXML2_MAJOR_VERSION	892
8.72.2.3 TIXML2_MINOR_VERSION	892
8.72.2.4 TIXML2_PATCH_VERSION	892
8.73 include/wasp.h File Reference	892
8.74 jikev2/include/jikencrypt.h File Reference	893
8.75 jikev2/include/jikeparse.h File Reference	893
8.76 jikev2/include/jikeprf.h File Reference	894
8.77 jikev2/include/jikev2.h File Reference	894
8.77.1 Macro Definition Documentation	896
8.77.1.1 AF_ALG	896
8.77.1.2 CRYPTOPP_ENABLE_NAMESPACE_WEAK	896
8.77.1.3 SOL_ALG	896
8.78 jikev2/include/jikev2dh.h File Reference	896
8.79 jikev2/include/jikev2sa.h File Reference	896
8.80 jlogger/include/jlogger.h File Reference	897
8.80.1 Macro Definition Documentation	897
8.80.1.1 COLORIZE	897
8.80.1.2 DARK_THEME	897
8.81 jresponder/include/jexec.h File Reference	897
8.82 jresponder/include/jresponder.h File Reference	898
8.83 jresponder/include/jsecresponder.h File Reference	898
8.84 jtestbench/include/interfacepp.h File Reference	898

8.85 jtestbench/include/jmmu.h File Reference	899
8.86 jtestbench/include/jproducer.h File Reference	899
8.87 jtestbench/include/jring.h File Reference	899
8.88 jtestbench/include/jsecproducer.h File Reference	900
8.89 jtestbench/include/jsectestbench.h File Reference	900
8.90 jtestbench/include/jtestbench.h File Reference	901
8.91 jtestbench/include/optionparser.h File Reference	901
8.91.1 Detailed Description	902
8.92 platforms/SEC/include/jsec.h File Reference	905
8.93 platforms/SEC/include/jsgt.h File Reference	905
8.93.1 Function Documentation	906
8.93.1.1 get	906
8.93.1.2 jsqt	906
8.93.1.3 read	906
8.93.1.4 sgt_t	906
8.93.1.5 ~jsqt	906
8.93.2 Variable Documentation	906
8.93.2.1 addrsize	906
8.93.2.2 bpid	906
8.93.2.3 chunksize	906
8.93.2.4 entrysize	906
8.93.2.5 extend	906
8.93.2.6 length	906
8.93.2.7 m_sgt	906
8.93.2.8 offset	906
8.93.2.9 order	906
8.93.2.10 sgt_t	906
8.93.2.11 sgtfinal	907
8.94 schema/ikev2.xsd File Reference	907
8.95 schema/protocolpp.xsd File Reference	907

Chapter 1

Protocol++ (ProtocolPP) Protocol, Encryption, and Authentication Library with Testbench and Drivers



Figure 1.1: www.protocolpp.com

Protocol++ (Protocolpp) is a software tool to be used standalone or in larger software projects to protect users' data and communicate with other devices around the world. It contains several interfaces to allow a user to directly access encryption and authentication algorithms to obscure and authenticate data. The protocols can be used to create software stacks such that all levels of the network can be supported. The protocols and underlying engines can also be used in drivers to send and receive data on local and internet networks. Finally, Protocol++ provides a testbench interface to allow the protocols to be driven to a software ring to test a device-under-test (DUT) for

compliance with the protocols standards supported by Protocol++.

1.1 Protocol++ Use Cases

Protocol++([ProtocolIPP](#)) can be used for several different use cases in development, software, hardware development, stacks, and testbenches.

- **TESTBENCHES** - Protocolpp comes with a testbench to allow the interface to be connected to a Device Under Test (DUT) through software rings for test of protocols, encryption, and authentication algorithms, replay windows, randomization, Diffie-Hellman routines, and other items. In addition, Protocol++ can be used to generate XML output for all of the above items that can be read back in to drive Verilog or software drivers for development of hardware accelerators and software
- **STACKS** - The drivers in [ProtocolIPP](#) show examples of how to write software stacks that support all levels of the OSI model to allow full manipulation of all features and methodologies of the protocol stack. Want to try out a new retry routine for TCP? Change the code in level 4 of your software stack to try it out. Want to try a new algorithm for IPsec? Add it to level 3 of the stack. Developing a driver for extended packet numbers in Macsec? Run your software against the Protocol++ testbench to ensure conformance. Additional protocols that do not need the stack and require direct access to level 3 (IP/IPsec) such as Real Time Protocol (RTP) or its secure version (SRTP)? Disable TCP/UDP and TLS to drive IP/IPsec directly
- **HARDWARE DEVELOPMENT** - Protocol++ can be used for testing hardware accelerators that support encryption and authentication algorithms. Developing an AES-GCM engine for your hard drive controller? Instantiate AES-GCM using the "ciphers" interface of [ProtocolIPP](#) in your SystemC testbench to driver your Verilog or VHDL through your UVM driver. Received your silicon back from manufacturing and need to verify there are no defects? Read back in the XML files generated during pre-silicon testing that achieves 100% coverage, and execute them through [ProtocolIPP](#)'s driver (or your own driver) and compare to the expected value. Have some conformance vectors from the specification? Enter the conformance data into the XML format specified by Protocol++'s XML schema, read the data into the testbench or driver, and test the silicon and or RTL
- **SOFTWARE** - The elements of [ProtocolIPP](#) can be incorporated into larger software projects to encrypt data, authenticate, generate CRC32 values, create Signatures, verify signatures, create PRF material, generate random data over ranges as bytes, words, or double words, enable SMFT mode and generate millions of random bytes from hardware is little or no time

These are the use cases currently being used. Development continues for Internet Key Exchange (IKEv2), additional driver features (ICMP message generation and return), offline key protection, key ring use, etc.

Please see the documentation found above and www.protocolpp.com for all options

1.2 Ciphers Interface

When using Protocol++ ciphers and authentication algorithms, the interface found in the [ProtocolIPP::ciphers](#) and [ProtocolIPP::jmodes](#) classes allows access to all the engines used. There is support for the following ciphers and modes:

Authentication is provided with the following algorithms

In addition, CRC support is provided for CRC32-IEEE

```
* jarray<uint8_t> fcs(4);
* std::shared_ptr<jmodes> engine = ciphers::get_crc32();
* engine->ProcessData(output->get_ptr(), output->get_size());
* engine->result(fcs.get_ptr(), fcs.get_size());
*
```

Usage Example CHACHA20-Poly1305 Encryption

	CBC	CTR	CCM	GCM	CMAC	GMAC	XCBC--MAC	AEAD	STREAM
AES	X	X	X	X	X	X	X	-	-
DES	X	-	-	-	-	-	-	-	-
CAME-LLIA	X	X	X	X	-	-	-	-	-
SEED	X	X	-	-	-	-	-	-	-
ARIA	X	X	X	X	-	-	-	-	-
SM4	X	X	X	X	-	-	-	-	-
CHACHA20	-	-	-	-	-	-	-	X	X
SNO-W3G	-	-	-	-	-	-	-	-	X
ZUC	-	-	-	-	-	-	-	-	X

Table 1.1: ProtocolPP Ciphers and Modes

To use the CHACHA20 cipher, the ciphers interface would be called with the appropriate parameters. Because CHACHA20 is a STREAM cipher (there's no direction in processing), direction is ignored, but STREAM must be passed with the key and initial value. Protocol++ will return a shared pointer to the engine as shown

```

/*
* std::shared_ptr<ProtocolPP::jarray<uint8_t>> output = std::make_shared<ProtocolPP::jarray<uint8_t>>(
*     data_length, 0);
*
* std::shared_ptr<jmodes> engine = ciphers::get_cipher(jmodes::CHACHA20,
*                                                       jmodes::ENC,
*                                                       jmodes::AEAD,
*                                                       cipherkey->get_ptr(),
*                                                       cipherkey->get_size(),
*                                                       iv->get_ptr(),
*                                                       iv->get_size(),
*                                                       icvlen);
*
* engine->ProcessData(padpyld.get_ptr(),
*                       ciphertext.get_ptr(),
*                       padpyld.get_size(),
*                       authdat.get_ptr(),
*                       authdat.get_size());
*
* icv = jarray<uint8_t>(icvlen, 0x00);
* engine->result(icv.get_ptr(), icv.get_size());
*
* ciphertext.append(icv);
*
* Encrypted output will be contained in the output pointer
*/

```

Usage Example SHA3-256 Authentication

To use the SHA3-256 authentication algorithm, the ciphers interface would be called with the authentication algorithm and the authentication key as shown below

```

/*
* icv = jarray<uint8_t>(icvlen);
* std::shared_ptr<jmodes> hmac = ciphers::get_auth(auth_t::SHA3_256,
*                                                 authkey->get_ptr(),
*                                                 authkey->get_size());
*
* hmac->ProcessData(authdat.get_ptr(), authdat.get_size());
* hmac->result(icv.get_ptr(), m_sec->get_field<uint32_t>(field_t::ICVLEN));
*
* If the desired MAC is less than the maximum length
*
* icv->resize(newlength);
*
* or
*
* hmac->result(short_mac->get_ptr(), short_mac->get_size());
*/

```

For API Documentation:

Auth Algo- rithm	ICV Length										
	4	8	10	12	16	20	24	32	48	64	
MD5	-	X	-	X	X	-	-	-	-	-	
SHA1	X	X	X	X	X	X	-	-	-	-	
SH-A224	-	X	-	X	X	X	X	-	-	-	
SH-A256	-	X	-	X	X	X	X	X	-	-	
SH-A384	-	X	-	X	X	X	X	X	X	-	
SH-A512	-	X	-	X	X	X	X	X	X	X	
SHA3-224	-	X	-	X	X	X	X	-	-	-	
SHA3-256	-	X	-	X	X	X	X	X	-	-	
SHA3-384	-	X	-	X	X	X	X	X	X	-	
SHA3-512	-	X	-	X	X	X	X	X	X	X	
POL-Y1305	-	X	-	X	X	-	-	-	-	-	
SM3	-	X	-	X	X	-	-	-	-	-	
SNO-W3G	X	-	-	-	-	-	-	-	-	-	
ZUC	X	-	-	-	-	-	-	-	-	-	

Table 1.2: [ProtocolPP](#) Authentication Algorithms**See Also**

[ProtocolPP::jmodes](#)
[ProtocolPP::ciphers](#)

For Additional Documentation:**See Also**

[jmodes](#)

1.3 ProtocolPP Interface

Use of the protocols found in Protocol++ is accomplished by the code found in [ProtocolPP::jprotocolpp](#). See the individual protocol sections found below for brief descriptions of the protocol and see the individual classes for full documentation of the features supported

Usage Example for IPsec Decapsulation

In this example, it is assumed that the parameters necessary for the IPsec flow have been negotiated by the endpoints. Depending on the protocol, this may be accomplished either by the IKEPRF protocol or the TLSPRF protocol using Diffie-Hellman or RSA for the key generation and negotiation. It is also assumed that the user has provided a cryptographically secure random data generator for padding, IV, and TFC padding creation (in the case of ENCAP when requested by the security parameters found in the security association

```
*  
* std::shared_ptr<ProtocolPP::jipsec> decap = std::make_shared<ProtocolPP::jipsec>(randomizer,  
*                                         security);  
*
```

```
*     std::shared_ptr<ProtocolPP::jarray<uint8_t>> plaintext = std::make_shared<ProtocolPP::jarray<uint8_t>>(0
* );
*
*     decap->decap_packet(encapped_packet,
*                           plaintext);
*
*
*     The payload will be present in the plaintext shared pointer
*
```

For API Documentation:

See Also

[ProtocolPP::jprotocolpp](#)

1.4 DriverPP Interface

As mentioned previously, Protocolpp (Protocol++) can be used for software stacks to communicate on networks and across the Internet. To facilitate this, Protocol++ contains drivers of different types to send and receive data from your application. Applications setup the driver with the IP address or host name they want to communicate with and the interfaces desired. If TLS is requested for the application level, TCP is used in the Transport level to assure delivery. SRTP uses UDP for best effort delivery to the endpoint. Access is provided to the Transport level for direct access to TCP and UDP. The user is able to select data assurance at the Network Layer by choosing ESPv4 or ESPv6 otherwise IPv4 or IPv6 can be selected. Data-Link Layer interfaces can select ethernet (Macsec), Wifi, WiMax, or LTE. LTE selection inserts RLC before/after LTE encapsulation and decapsulation (see LTE stack diagram in [ProtocolPP::jlte](#))

Ring Driver

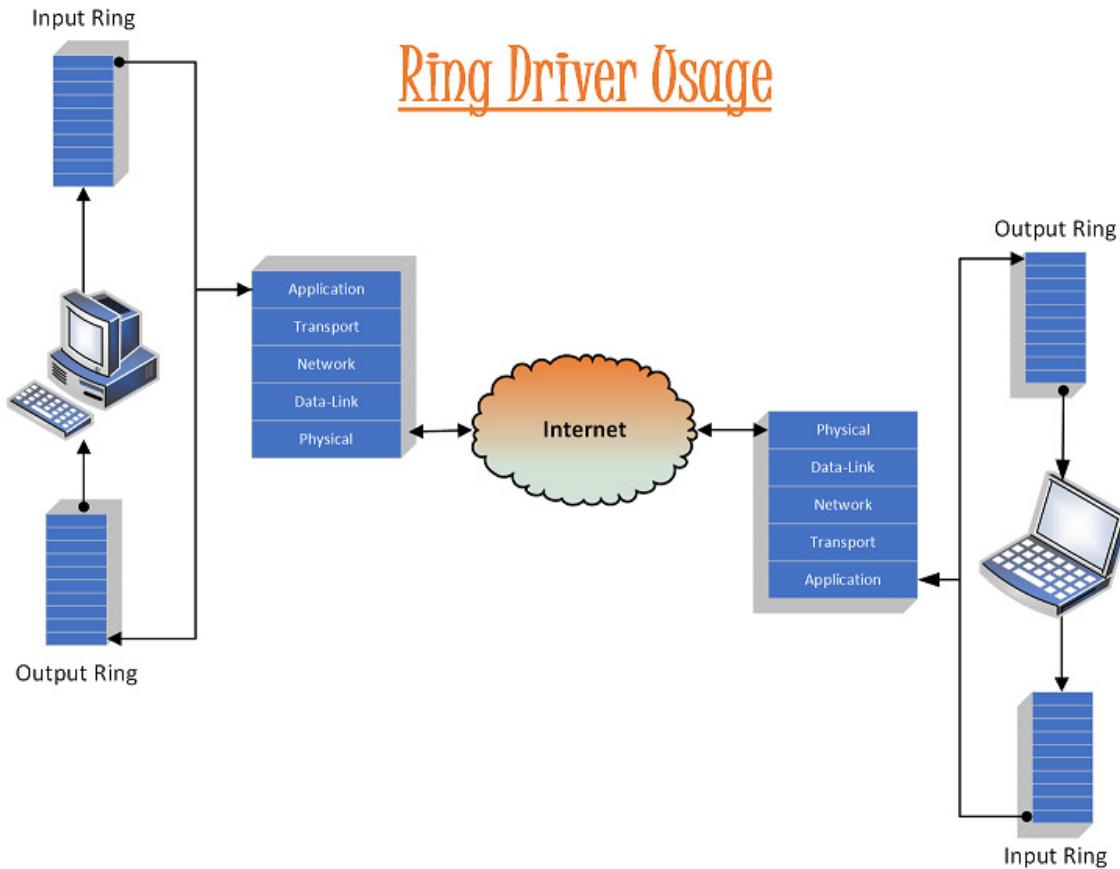


Figure 1.2: Protocol++ Ring Driver Illustration

```

/*
* // create logger object
* std::shared_ptr<InterfacePP::logger> mylogger = std::make_shared<InterfacePP::jlogger>("./driverlog.log"
* );
*
* // generate random seed from hardware device
* std::random_device rd;
* uint64_t myseed = rd();
* myseed = ((myseed << 32) | rd());
*
* // allocate memory for rings if they don't already exist
* uint8_t* iptr = ((iring != 0) ? (uint8_t*)iring : ((uint8_t*) new uint8_t[(rsize*sizeof(
*     InterfacePP::ringin))]));
* uint8_t* optr = ((oring != 0) ? (uint8_t*)oring : ((uint8_t*) new uint8_t[(rsize*sizeof(
*     InterfacePP::ringout))]));
*
* // create software rings
* char siaddr[16];
* char soaddr[16];
* sprintf(siaddr, "%p", iptr);
* sprintf(soaddr, "%p", optr);
* uint64_t iaddr = std::strtoul(siaddr, NULL, 16);
* uint64_t oaddr = std::strtoul(soaddr, NULL, 16);
*
* // track dynamic memory for deletion
* std::shared_ptr<InterfacePP::jmmu> mmu = std::make_shared<InterfacePP::jmmu>();
* mmu->set_mem("testbench", iptr);
* mmu->set_mem("testbench", optr);
*
* // attach driver to shared pointer
* std::shared_ptr<InterfacePP::jring<InterfacePP::secin>> iring =
*     std::make_shared<InterfacePP::jring<InterfacePP::secin>>(iaddr, rsize);
* std::shared_ptr<InterfacePP::jring<InterfacePP::secout>> oring =
*     std::make_shared<InterfacePP::jring<InterfacePP::secout>>(oaddr, rsize);
*
* // create the ringdriver
* std::shared_ptr<DriverPP::jringdrive<ProtocolPP::SRTP>> rdriver =
*     std::make_shared<DriverPP::jringdrive<ProtocolPP::SRTP>>(myport,
*
*     ProtocolPP::iana_t::ESPV4,

```

```

*
*           ProtocolPP::protocol_t::MACSEC,
*
*           std::string("www.protocolpp.com"),
*
*           myseed,
*
*           ProtocolPP::endian_t::LITTLE,
*
*           mymmu,
*
*           mylogger,
*
*           iring,
*
*           oring);
* // start up the driver if it correctly connected
* if (rdriver->connected()) {
*     rdriver->receive();
*     rdriver->send();
* }
* else {
*     // teardown the driver if it didn't connect
*     rdriver->teardown();
* }
*
* // send and receive run on separate threads
* tparam->type = std::string("SRTP");
* tparam->driver = driver;
*
* // threaded testbench, producer on different threads
* m_thread.push_back(std::thread(runsnd, tparam));
* m_thread.push_back(std::thread(runrcv, tparam));
*
* // check for finish
* for (int i=0; i<2; i++) {
*     m_thread[i].join();
* }
*
* return 0;
*
* void *runsnd(void *sender) {
*
*     tdata_t* sendit = static_cast<tdata_t*>(sender);
*     std::string type = sendit->type;
*
*     DriverPP::jringdrive<ProtocolPP::judp>* snd = static_cast<
*     DriverPP::jringdrive<ProtocolPP::judp>*>(sendit->driver);
*     snd->send();
*
*     return NULL;
* }
*
* void *runrcv(void *receiver) {
*
*     tdata_t* rcvit = static_cast<tdata_t*>(receiver);
*     std::string type = rcvit->type;
*
*     DriverPP::jringdrive<ProtocolPP::judp>* rcv = static_cast<
*     DriverPP::jringdrive<ProtocolPP::judp>*>(rcvit->driver);
*     rcv->receive();
*
*     return NULL;
* }
*
*

```

For API Documentation:**See Also**

[DriverPP::jdrive](#)
[DriverPP::jringdrive](#)
[DriverPP::jdirectdrive](#)

For Additional Documentation:**See Also**

[jdrive](#)
[jringdrive](#)
[jdirectdrive](#)

1.5 UDP

(See https://en.wikipedia.org/wiki/User_Datagram_Protocol)

The User Datagram Protocol (UDP) is one of the core members of the Internet protocol suite. The protocol was designed by David P. Reed in 1980 and formally defined in RFC 768.

UDP uses a simple connectionless transmission model with a minimum of protocol mechanism. It has no handshaking dialogues, and thus exposes the user's program to any unreliability of the underlying network protocol. There is no guarantee of delivery, ordering, or duplicate protection. UDP provides checksums for data integrity, and port numbers for addressing different functions at the source and destination of the datagram.

With UDP, computer applications can send messages, in this case referred to as datagrams, to other hosts on an Internet Protocol (IP) network without prior communications to set up special transmission channels or data paths. UDP is suitable for purposes where error checking and correction is either not necessary or is performed in the application, avoiding the overhead of such processing at the network interface level. Time-sensitive applications often use UDP because dropping packets is preferable to waiting for delayed packets, which may not be an option in a real-time system. If error correction facilities are needed at the network interface level, an application may use the Transmission Control Protocol (TCP) or Stream Control Transmission Protocol (SCTP) which are designed for this purpose.

Attributes

A number of UDP's attributes make it especially suited for certain applications

- It is transaction-oriented, suitable for simple query-response protocols such as the Domain Name System or the Network Time Protocol.
- It provides datagrams, suitable for modeling other protocols such as in IP tunneling or Remote Procedure Call and the Network File System.
- It is simple, suitable for bootstrapping or other purposes without a full protocol stack, such as the DHCP and Trivial File Transfer Protocol.
- It is stateless, suitable for very large numbers of clients, such as in streaming media applications for example IPTV
- The lack of retransmission delays makes it suitable for real-time applications such as Voice over IP, online games, and many protocols built on top of the Real Time Streaming Protocol.
- Works well in unidirectional communication, suitable for broadcast information such as in many kinds of service discovery and shared information such as broadcast time or Routing Information Protocol

Reliability and congestion control solutions

Lacking reliability, UDP applications must generally be willing to accept some loss, errors or duplication. Some applications, such as TFTP, may add rudimentary reliability mechanisms into the application layer as needed

Most often, UDP applications do not employ reliability mechanisms and may even be hindered by them. Streaming media, real-time multiplayer games and voice over IP (VoIP) are examples of applications that often use UDP. In these particular applications, loss of packets is not usually a fatal problem. If an application requires a high degree of reliability, a protocol such as the Transmission Control Protocol may be used instead.

In VoIP, for example, latency and jitter are the primary concerns. The use of TCP would cause jitter if any packets were lost as TCP does not provide subsequent data to the application while it is requesting re-sending of the missing data. If using UDP the end user applications must provide any necessary handshaking such as real time confirmation that the message has been received.

Applications

Numerous key Internet applications use UDP, including: the Domain Name System (DNS), where queries must be fast and only consist of a single request followed by a single reply packet, the Simple Network Management Protocol (SNMP), the Routing Information Protocol (RIP) and the Dynamic Host Configuration Protocol (DHCP).

Voice and video traffic is generally transmitted using UDP. Real-time video and audio streaming protocols are designed to handle occasional lost packets, so only slight degradation in quality occurs, rather than large delays if lost

packets were retransmitted. Because both TCP and UDP run over the same network, many businesses are finding that a recent increase in UDP traffic from these real-time applications is hindering the performance of applications using TCP, such as point of sale, accounting, and database systems. When TCP detects packet loss, it will throttle back its data rate usage. Since both real-time and business applications are important to businesses, developing quality of service solutions is seen as crucial by some.

Some VPN systems such as OpenVPN may use UDP while implementing reliable connections and error checking at the application level.

For the Protocol++ interface to UDP see

For API Documentation:

See Also

[ProtocolPP::judpsa](#)
[ProtocolPP::judp](#)
[ProtocolPP::jprotocolpp](#)

For Additional Documentation:

See Also

[judpsa](#)
[judp](#)
[jprotocolpp](#)

1.6 TCP

(See https://en.wikipedia.org/wiki/Transmission_Control_Protocol)

The Transmission Control Protocol (TCP) is a core protocol of the Internet protocol suite. It originated in the initial network implementation in which it complemented the Internet Protocol (IP). Therefore, the entire suite is commonly referred to as TCP/IP. TCP provides reliable, ordered, and error-checked delivery of a stream of octets between applications running on hosts communicating over an IP network. Major Internet applications such as the World Wide Web, email, remote administration and file transfer rely on TCP. Applications that do not require reliable data stream service may use the User Datagram Protocol (UDP), which provides a connectionless datagram service that emphasizes reduced latency over reliability.

Historical origin

In May 1974, the Institute of Electrical and Electronic Engineers (IEEE) published a paper titled "A Protocol for Packet Network Intercommunication." The paper's authors, Vint Cerf and Bob Kahn, described an internetworking protocol for sharing resources using packet-switching among the nodes. A central control component of this model was the Transmission Control Program that incorporated both connection-oriented links and datagram services between hosts. The monolithic Transmission Control Program was later divided into a modular architecture consisting of the Transmission Control Protocol at the connection-oriented layer and the Internet Protocol at the internetworking (datagram) layer. The model became known informally as TCP/IP, although formally it was henceforth called the Internet Protocol Suite.

Network function

The Transmission Control Protocol provides a communication service at an intermediate level between an application program and the Internet Protocol. It provides host-to-host connectivity at the Transport Layer of the Internet model. An application does not need to know the particular mechanisms for sending data via a link to another host, such as the required packet fragmentation on the transmission medium. At the transport layer, the protocol handles all handshaking and transmission details and presents an abstraction of the network connection to the application.

At the lower levels of the protocol stack, due to network congestion, traffic load balancing, or other unpredictable network behavior, IP packets may be lost, duplicated, or delivered out of order. TCP detects these problems, requests retransmission of lost data, rearranges out-of-order data, and even helps minimize network congestion to reduce the occurrence of the other problems. If the data still remains undelivered, its source is notified of this

failure. Once the TCP receiver has reassembled the sequence of octets originally transmitted, it passes them to the receiving application. Thus, TCP abstracts the application's communication from the underlying networking details.

TCP is utilized extensively by many popular applications carried on the Internet, including the World Wide Web (WWW), E-mail, File Transfer Protocol, Secure Shell, peer-to-peer file sharing, and many streaming media applications.

TCP is optimized for accurate delivery rather than timely delivery, and therefore, TCP sometimes incurs relatively long delays (on the order of seconds) while waiting for out-of-order messages or retransmissions of lost messages. It is not particularly suitable for real-time applications such as Voice over IP. For such applications, protocols like the Real-time Transport Protocol (RTP) running over the User Datagram Protocol (UDP) are usually recommended instead.

TCP is a reliable stream delivery service which guarantees that all bytes received will be identical with bytes sent and in the correct order. Since packet transfer over many networks is not reliable, a technique known as positive acknowledgment with retransmission is used to guarantee reliability of packet transfers. This fundamental technique requires the receiver to respond with an acknowledgment message as it receives the data. The sender keeps a record of each packet it sends. The sender also maintains a timer from when the packet was sent, and retransmits a packet if the timer expires before the message has been acknowledged. The timer is needed in case a packet gets lost or corrupted

While IP handles actual delivery of the data, TCP keeps track of the individual units of data transmission, called segments, that a message is divided into for efficient routing through the network. For example, when an HTML file is sent from a web server, the TCP software layer of that server divides the sequence of octets of the file into segments and forwards them individually to the IP software layer (Internet Layer). The Internet Layer encapsulates each TCP segment into an IP packet by adding a header that includes (among other data) the destination IP address. When the client program on the destination computer receives them, the TCP layer (Transport Layer) reassembles the individual segments, and ensures they are correctly ordered and error free as it streams them to an application.

For the Protocol++ interface to TCP see

For API Documentation:

See Also

[ProtocolPP::jtcpfa](#)
[ProtocolPP::jtcp](#)
[ProtocolPP::jprotocolpp](#)

For Additional Documentation:

See Also

[jtcpfa](#)
[jtcp](#)
[jprotocolpp](#)

1.7 ICMP

See https://en.wikipedia.org/wiki/Internet_Control_Message_Protocol

See https://en.wikipedia.org/wiki/Internet_Control_Message_Protocol_version_6

The Internet Control Message Protocol (ICMP) is one of the main protocols of the internet protocol suite. It is used by network devices, like routers, to send error messages indicating, for example, that a requested service is not available or that a host or router could not be reached. ICMP can also be used to relay query messages.[1] It is assigned protocol number 1.[2] ICMP[3] differs from transport protocols such as TCP and UDP in that it is not typically used to exchange data between systems, nor is it regularly employed by end-user network applications (with the exception of some diagnostic tools like ping and traceroute)

For the Protocol++ interface to ICMP and ICMPv6 see

For API Documentation:

See Also

[ProtocolPP::jicmpsa](#)
[ProtocolPP::jicmp](#)
[ProtocolPP::jprotocolpp](#)

For API Documentation:**See Also**

[jicmpsa](#)
[jicmp](#)
[jprotocolpp](#)

1.8 IP

(See https://en.wikipedia.org/wiki/Internet_Protocol)

The Internet Protocol (IP) is the principal communications protocol in the Internet protocol suite for relaying datagrams across network boundaries. Its routing function enables internetworking, and essentially establishes the Internet.

IP has the task of delivering packets from the source host to the destination host solely based on the IP addresses in the packet headers. For this purpose, IP defines packet structures that encapsulate the data to be delivered. It also defines addressing methods that are used to label the datagram with source and destination information.

Historically, IP was the connectionless datagram service in the original Transmission Control Program introduced by Vint Cerf and Bob Kahn in 1974; the other being the connection-oriented Transmission Control Protocol (TCP). The Internet protocol suite is therefore often referred to as TCP/IP.

The first major version of IP, Internet Protocol Version 4 (IPv4), is the dominant protocol of the Internet. Its successor is Internet Protocol Version 6 (IPv6).

Function

The Internet Protocol is responsible for addressing hosts and for routing datagrams (packets) from a source host to a destination host across one or more IP networks. For this purpose, the Internet Protocol defines the format of packets and provides an addressing system that has two functions: Identifying hosts and providing a logical location service

Datagram construction

Each datagram has two components: a header and a payload. The IP header is tagged with the source IP address, the destination IP address, and other meta-data needed to route and deliver the datagram. The payload is the data that is transported. This method of nesting the data payload in a packet with a header is called encapsulation.

IP addressing and routing

IP addressing entails the assignment of IP addresses and associated parameters to host interfaces. The address space is divided into networks and subnetworks, involving the designation of network or routing prefixes. IP routing is performed by all hosts, as well as routers, whose main function is to transport packets across network boundaries. Routers communicate with one another via specially designed routing protocols, either interior gateway protocols or exterior gateway protocols, as needed for the topology of the network.

IP routing is also common in local networks. For example, many Ethernet switches support IP multicast operations. These switches use IP addresses and Internet Group Management Protocol to control multicast routing but use MAC addresses for the actual routing.

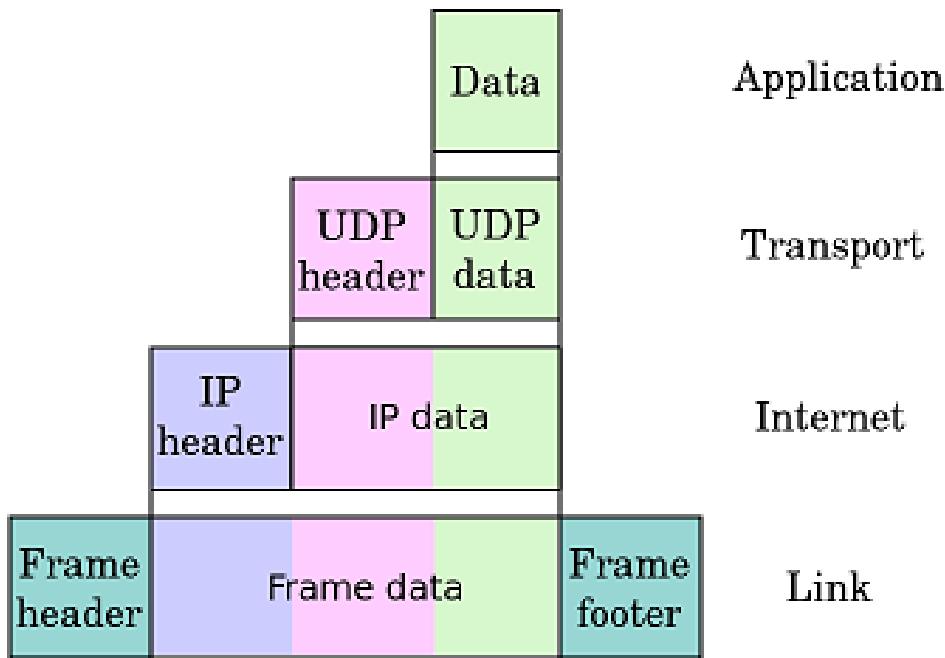


Figure 1.3: UDP Encapsulation

Reliability

The design of the Internet protocols is based on the end-to-end principle. The network infrastructure is considered inherently unreliable at any single network element or transmission medium and assumes that it is dynamic in terms of availability of links and nodes. No central monitoring or performance measurement facility exists that tracks or maintains the state of the network. For the benefit of reducing network complexity, the intelligence in the network is purposely located in the end nodes of data transmission. Routers in the transmission path forward packets to the next known, directly reachable gateway matching the routing prefix for the destination address.

As a consequence of this design, the Internet Protocol only provides best effort delivery and its service is characterized as unreliable. In network architectural language, it is a connectionless protocol, in contrast to connection-oriented modes of transmission. Various error conditions may occur, such as data corruption, packet loss, duplication and out-of-order delivery. Because routing is dynamic, meaning every packet is treated independently, and because the network maintains no state based on the path of prior packets, different packets may be routed to the same destination via different paths, resulting in out-of-order sequencing at the receiver.

Internet Protocol Version 4 (IPv4) provides safeguards to ensure that the IP packet header is error-free. A routing node calculates a checksum for a packet. If the checksum is bad, the routing node discards the packet. The routing node does not have to notify either end node, although the Internet Control Message Protocol (ICMP) allows such notification. By contrast, in order to increase performance, and since current link layer technology is assumed to provide sufficient error detection, the IPv6 header has no checksum to protect it.

All error conditions in the network must be detected and compensated by the end nodes of a transmission. The upper layer protocols of the Internet protocol suite are responsible for resolving reliability issues. For example, a host may cache network data to ensure correct ordering before the data is delivered to an application.

Link capacity and capability

The dynamic nature of the Internet and the diversity of its components provide no guarantee that any particular path is actually capable of, or suitable for, performing the data transmission requested, even if the path is available and reliable. One of the technical constraints is the size of data packets allowed on a given link. An application must assure that it uses proper transmission characteristics. Some of this responsibility lies also in the upper layer protocols. Facilities exist to examine the maximum transmission unit (MTU) size of the local link and Path MTU Discovery can be used for the entire projected path to the destination. The IPv4 internetworking layer has the capability to automatically fragment the original datagram into smaller units for transmission. In this case, IP

provides re-ordering of fragments delivered out of order.

The Transmission Control Protocol (TCP) is an example of a protocol that adjusts its segment size to be smaller than the MTU. The User Datagram Protocol (UDP) and the Internet Control Message Protocol (ICMP) disregard MTU size, thereby forcing IP to fragment oversized datagrams.

Version history

The versions currently relevant are IPv4 and IPv6.

In May 1974, the Institute of Electrical and Electronic Engineers (IEEE) published a paper entitled "A Protocol for Packet Network Intercommunication".^[6] The paper's authors, Vint Cerf and Bob Kahn, described an internetworking protocol for sharing resources using packet switching among network nodes. A central control component of this model was the "Transmission Control Program" that incorporated both connection-oriented links and datagram services between hosts. The monolithic Transmission Control Program was later divided into a modular architecture consisting of the Transmission Control Protocol at the transport layer and the Internet Protocol at the network layer. The model became known as the Department of Defense (DoD) Internet Model and Internet Protocol Suite, and informally as TCP/IP.

IP versions 0 to 3 were experimental versions, used between 1977 and 1979. The following Internet Experiment Note (IEN) documents describe versions of the Internet Protocol prior to the modern version of IPv4:

- IEN 2 (Comments on Internet Protocol and TCP), dated August 1977 describes the need to separate the TCP and Internet Protocol functionalities (which were previously combined.) It proposes the first version of the IP header, using 0 for the version field.
- IEN 26 (A Proposed New Internet Header Format), dated February 1978 describes a version of the IP header that uses a 1-bit version field.
- IEN 28 (Draft Internetwork Protocol Description Version 2), dated February 1978 describes IPv2.
- IEN 41 (Internetwork Protocol Specification Version 4), dated June 1978 describes the first protocol to be called IPv4. The IP header is different from the modern IPv4 header.
- IEN 44 (Latest Header Formats), dated June 1978 describes another version of IPv4, also with a header different from the modern IPv4 header.
- IEN 54 (Internetwork Protocol Specification Version 4), dated September 1978 is the first description of IPv4 using the header that would be standardized in RFC 760.

The dominant internetworking protocol in the Internet Layer in use today is IPv4; the number 4 is the protocol version number carried in every IP datagram. IPv4 is described in RFC 791 (1981).

Version 5 was used by the Internet Stream Protocol, an experimental streaming protocol.

The successor to IPv4 is IPv6. Its most prominent difference from version 4 is the size of the addresses. While IPv4 uses 32 bits for addressing, yielding c. 4.3 billion (4.3×10^9) addresses, IPv6 uses 128-bit addresses providing ca. 340 undecillion, or 3.4×10^{38} addresses. Although adoption of IPv6 has been slow, as of June 2008, all United States government systems have demonstrated basic infrastructure support for IPv6. IPv6 was a result of several years of experimentation and dialog during which various protocol models were proposed, such as TP/IX (RFC 1475), PIP (RFC 1621) and TUBA (TCP and UDP with Bigger Addresses, RFC 1347).

The assignment of the new protocol as IPv6 was uncertain until due diligence revealed that IPv6 had not yet been used previously. Other protocol proposals named IPv9 and IPv8 briefly surfaced, but had no affiliation with any international standards body, and have had no support

Security

During the design phase of the ARPANET and the early Internet, the security aspects and needs of a public, international network could not be adequately anticipated. Consequently, many Internet protocols exhibited vulnerabilities highlighted by network attacks and later security assessments. In 2008, a thorough security assessment and proposed mitigation of problems was published. The Internet Engineering Task Force (IETF) has been pursuing further studies

For the Protocol++ interface to IP see

For API Documentation:

See Also

[ProtocolPP::jipsa](#)
[ProtocolPP::jip](#)
[ProtocolPP::jprotocolpp](#)

For Additional Documentation:

See Also

[jipsa](#)
[jip](#)
[jprotocolpp](#)

1.9 TLS

See https://en.wikipedia.org/wiki/Transport_Layer_Security

Transport Layer Security (TLS) and its predecessor, Secure Sockets Layer (SSL), both of which are frequently referred to as 'SSL', are cryptographic protocols that provide communications security over a computer network. Several versions of the protocols are in widespread use in applications such as web browsing, email, Internet faxing, instant messaging, and voice-over-IP (VoIP). Major web sites use TLS to secure all communications between their servers and web browsers

The primary goal of the Transport Layer Security protocol is to provide privacy and data integrity between two communicating computer applications. When secured by TLS, connections between a client (e.g., a web browser) and a server (e.g., wikipedia.org) have one or more of the following properties:

- The connection is private because symmetric cryptography is used to encrypt the data transmitted. The keys for this symmetric encryption are generated uniquely for each connection and are based on a shared secret negotiated at the start of the session (see TLS handshake protocol). The server and client negotiate the details of which encryption algorithm and cryptographic keys to use before the first byte of data is transmitted (see Algorithm). The negotiation of a shared secret is both secure (the negotiated secret is unavailable to eavesdroppers and cannot be obtained, even by an attacker who places himself in the middle of the connection) and reliable (no attacker can modify the communications during the negotiation without being detected)
- The identity of the communicating parties can be authenticated using public-key cryptography. This authentication can be made optional, but is generally required for at least one of the parties (typically the server)
- The connection is reliable because each message transmitted includes a message integrity check using a message authentication code to prevent undetected loss or alteration of the data during transmission

In addition to the properties above, careful configuration of TLS can provide additional privacy-related properties such as forward secrecy, ensuring that any future disclosure of encryption keys cannot be used to decrypt any TLS communications recorded in the past

TLS supports many different methods for exchanging keys, encrypting data, and authenticating message integrity (see Algorithm). As a result, secure configuration of TLS involves many configurable parameters, and not all choices provide all of the privacy-related properties described in the list above (see authentication and key exchange table, cipher security table, and data integrity table)

Attempts have been made to subvert aspects of the communications security that TLS seeks to provide and the protocol has been revised several times to address these security threats (see Security). Web browsers have also been revised by their developers to defend against potential security weaknesses after these were discovered (see TLS/SSL support history of web browsers.)

The TLS protocol is composed of two layers: the TLS record protocol and the TLS handshake protocol

TLS is a proposed Internet Engineering Task Force (IETF) standard, first defined in 1999 and updated in RFC 5246 (August 2008) and RFC 6176 (March 2011). It is based on the earlier SSL specifications (1994, 1995, 1996) developed by Netscape Communications for adding the HTTPS protocol to their Navigator web browser.

Description

Client-server applications use the TLS protocol to communicate across a network in a way designed to prevent eavesdropping and tampering

Since protocols can operate either with or without TLS (or SSL), it is necessary for the client to indicate to the server the setup of a TLS connection. There are two main ways of achieving this. One option is to use a different port number for TLS connections (for example, port 443 for HTTPS). The other is for the client to use a protocol-specific mechanism (for example, STARTTLS for mail and news protocols) to request that the server switch the connection to TLS

Once the client and server have agreed to use TLS, they negotiate a stateful connection by using a handshaking procedure. During this handshake, the client and server agree on various parameters used to establish the connection's security:

- The handshake begins when a client connects to a TLS-enabled server requesting a secure connection and presents a list of supported cipher suites (ciphers and hash functions)
- From this list, the server picks a cipher and hash function that it also supports and notifies the client of the decision
- The server usually then sends back its identification in the form of a digital certificate. The certificate contains the server name, the trusted certificate authority (CA) and the server's public encryption key
- The client confirms the validity of the certificate before proceeding
- To generate the session keys used for the secure connection, the client either:
 - encrypts a random number with the server's public key and sends the result to the server (which only the server should be able to decrypt with its private key); both parties then use the random number to generate a unique session key for subsequent encryption and decryption of data during the session
 - uses Diffie-Hellman key exchange to securely generate a random and unique session key for encryption and decryption that has the additional property of forward secrecy: if the server's private key is disclosed in future, it cannot be used to decrypt the current session, even if the session is intercepted and recorded by a third party.

This concludes the handshake and begins the secured connection, which is encrypted and decrypted with the session key until the connection closes. If any one of the above steps fail, the TLS handshake fails, and the connection is not created

TLS and SSL are defined as 'operating over some reliable transport layer', which places them as application layer protocols in the TCP/IP reference model and as presentation layer protocols in the OSI model. The protocols use a handshake with an asymmetric cipher to establish cipher settings and a shared key for a session; the rest of the communication is encrypted using a symmetric cipher and the session key

History and development

Secure Network Programming

Early research efforts towards transport layer security included the Secure Network Programming (SNP) application programming interface (API), which in 1993 explored the approach of having a secure transport layer API closely resembling Berkeley sockets, to facilitate retrofitting preexisting network applications with security measures

SSL 1.0, 2.0 and 3.0

Netscape developed the original SSL protocols. Version 1.0 was never publicly released because of serious security flaws in the protocol; version 2.0, released in February 1995, "contained a number of security flaws which ultimately led to the design of SSL version 3.0". Released in 1996, SSL version 3.0 represented a complete redesign of the protocol produced by Paul Kocher working with Netscape engineers Phil Karlton and Alan Freier, with a reference implementation by Christopher Allen and Tim Dierks of Consensus Development. Newer versions of SSL/TLS are based on SSL 3.0. The 1996 draft of SSL 3.0 was published by IETF as a historical document in RFC 6101

Dr. Taher Elgamal, chief scientist at Netscape Communications from 1995 to 1998, is recognized as the "father of SSL"

As of 2014 the 3.0 version of SSL is considered insecure as it is vulnerable to the POODLE attack that affects all block ciphers in SSL; and RC4, the only non-block cipher supported by SSL 3.0, is also feasibly broken as used in SSL 3.0.

SSL 2.0 was deprecated (prohibited) in 2011 by RFC 6176

SSL 3.0 was deprecated in June 2015 by RFC 7568

TLS 1.0

TLS 1.0 was first defined in RFC 2246 in January 1999 as an upgrade of SSL Version 3.0, and written by Christopher Allen and Tim Dierks of Consensus Development. As stated in the RFC, "the differences between this protocol and SSL 3.0 are not dramatic, but they are significant enough to preclude interoperability between TLS 1.0 and SSL 3.0". TLS 1.0 does include a means by which a TLS implementation can downgrade the connection to SSL 3.0, thus weakening security

TLS 1.1

TLS 1.1 was defined in RFC 4346 in April 2006. It is an update from TLS version 1.0. Significant differences in this version include:

- Added protection against cipher-block chaining (CBC) attacks. The implicit initialization vector (IV) was replaced with an explicit IV
- Change in handling of padding errors
- Support for IANA registration of parameters

TLS 1.2

TLS 1.2 was defined in RFC 5246 in August 2008. It is based on the earlier TLS 1.1 specification. Major differences include:

- The MD5-SHA-1 combination in the pseudorandom function (PRF) was replaced with SHA-256, with an option to use cipher suite specified PRFs
- The MD5-SHA-1 combination in the finished message hash was replaced with SHA-256, with an option to use cipher suite specific hash algorithms. However the size of the hash in the finished message must still be at least 96 bits
- The MD5-SHA-1 combination in the digitally signed element was replaced with a single hash negotiated during handshake, which defaults to SHA-1
- Enhancement in the client's and server's ability to specify which hash and signature algorithms they accept
- Expansion of support for authenticated encryption ciphers, used mainly for Galois/Counter Mode (GCM) and CCM mode of Advanced Encryption Standard encryption
- TLS Extensions definition and Advanced Encryption Standard cipher suites were added

All TLS versions were further refined in RFC 6176 in March 2011 removing their backward compatibility with SSL such that TLS sessions never negotiate the use of Secure Sockets Layer (SSL) version 2.0

TLS 1.3 (draft)

As of January 2016, TLS 1.3 is a working draft, and details are provisional and incomplete. It is based on the earlier TLS 1.2 specification. Major differences from TLS 1.2 include:

- Removing support for weak and lesser used named elliptic curves (see Elliptic curve cryptography)
- Removing support for MD5 and SHA-224 cryptographic hash functions
- Requiring digital signatures even when a previous configuration is used
- Integrating HKDF and the semi-ephemeral DH proposal

- Replacing resumption with PSK and tickets
- Supporting 1-RTT handshakes and initial support for 0-RTT (see Round-trip delay time)
- Dropping support for many insecure or obsolete features including compression, renegotiation, non-AEAD ciphers, static RSA and static DH key exchange, custom DHE groups, point format negotiation, Change Cipher Spec protocol, Hello message UNIX time, and the length field AD input to AEAD ciphers
- Prohibiting SSL or RC4 negotiation for backwards compatibility
- Integrating use of session hash
- Deprecating use of the record layer version number and freezing the number for improved backwards compatibility
- Moving some security related algorithm details from an appendix to the specification and relegating Client-KeyShare to an appendix
- Adding of Curve25519 and Ed25519 to the TLS standard

Algorithm

Before a client and server can begin to exchange information protected by TLS, they must securely exchange or agree upon an encryption key and a cipher to use when encrypting data (see Cipher). Among the methods used for key exchange/agreement are: public and private keys generated with RSA (denoted TLS_RSA in the TLS handshake protocol), Diffie-Hellman (TLS_DH), ephemeral Diffie-Hellman (TLS_DHE), Elliptic Curve Diffie-Hellman (TLS_ECDH), ephemeral Elliptic Curve Diffie-Hellman (TLS_ECDHE), anonymous Diffie-Hellman (TLS_DH_anon), pre-shared key (TLS_PSK)[19] and Secure Remote Password (TLS_SRTP)

The TLS_DH_anon and TLS_ECDH_anon key agreement methods do not authenticate the server or the user and hence are rarely used because those are vulnerable to Man-in-the-middle attack. Only TLS_DHE and TLS_ECDHE provide forward secrecy

Public key certificates used during exchange/agreement also vary in the size of the public/private encryption keys used during the exchange and hence the robustness of the security provided. In July 2013, Google announced that it would no longer use 1024 bit public keys and would switch instead to 2048 bit keys to increase the security of the TLS encryption it provides to its users

For the Protocol++ interface to SSL/TLS see

For API Documentation:

See Also

[ProtocolPP::jtlsa](#)
[ProtocolPP::jtls](#)
[ProtocolPP::jprotocolpp](#)

For Additional Documentation:

See Also

[jtlsa](#)
[jtls](#)
[jprotocolpp](#)

1.10 IKEv2

See RFC7296 Internet Key Exchange Protocol Version 2

IKE performs mutual authentication between two parties and establishes an IKE Security Association (SA) that includes shared secret information that can be used to efficiently establish SAs for Encapsulating Security Payload (ESP) [ESP] or Authentication Header (AH) and a set of cryptographic algorithms to be used by the SAs to protect

the traffic that they carry. An initiator proposes one or more suites by listing supported algorithms that can be combined into suites in a mix-and-match fashion. IKE can also negotiate use of IP Compression (IPComp) [IP-COMP] in connection with an ESP or AH SA. The SAs for ESP or AH that get set up through that IKE SA we call "Child SAs"

All IKE communications consist of pairs of messages: a request and a response. The pair is called an "exchange", and is sometimes called a "request/response pair". The first two exchanges of messages establishing an IKE SA are called the IKE_SA_INIT exchange and the IKE_AUTH exchange; subsequent IKE exchanges are called either CREATE_CHILD_SA exchanges or INFORMATIONAL exchanges. In the common case, there is a single IKE_SA_INIT exchange and a single IKE_AUTH exchange (a total of four messages) to establish the IKE SA and the first Child SA. In exceptional cases, there may be more than one of each of these exchanges. In all cases, all IKE_SA_INIT exchanges MUST complete before any other exchange type, then all IKE_AUTH exchanges MUST complete, and following that, any number of CREATE_CHILD_SA and INFORMATIONAL exchanges may occur in any order. In some scenarios, only a single Child SA is needed between the IPsec endpoints, and therefore there would be no additional exchanges. Subsequent exchanges MAY be used to establish additional Child SAs between the same authenticated pair of endpoints and to perform housekeeping functions

For the Protocol++ interface to IKEv2 see

For API Documentation:

See Also

[ProtocolPP::jikev2sa](#)
[ProtocolPP::jikev2](#)
[ProtocolPP::jikencrypt](#)
[ProtocolPP::jikeprf](#)
[ProtocolPP::jikev2dh](#)
[ProtocolPP::jikeparse](#)
[ProtocolPP::jprotocolpp](#)

For Additional Documentation:

See Also

[jikev2sa](#)
[jikev2](#)
[jikencrypt](#)
[jikeprf](#)
[jikev2dh](#)
[jikeparse](#)
[jprotocolpp](#)

1.11 IPsec

See <https://en.wikipedia.org/wiki/IPsec>

Internet Protocol Security (IPsec) is a protocol suite for secure Internet Protocol (IP) communications that works by authenticating and encrypting each IP packet of a communication session. IPsec includes protocols for establishing mutual authentication between agents at the beginning of the session and negotiation of cryptographic keys to be used during the session. IPsec can be used in protecting data flows between a pair of hosts (host-to-host), between a pair of security gateways (network-to-network), or between a security gateway and a host (network-to-host). Internet Protocol security (IPsec) uses cryptographic security services to protect communications over Internet Protocol (IP) networks. IPsec supports network-level peer authentication, data origin authentication, data integrity, data confidentiality (encryption), and replay protection.

IPsec is an end-to-end security scheme operating in the Internet Layer of the Internet Protocol Suite, while some other Internet security systems in widespread use, such as Transport Layer Security (TLS) and Secure Shell (SSH), operate in the upper layers at the Transport Layer (TLS) and the Application layer (SSH). Hence, only IPsec protects all application traffic over an IP network. Applications can be automatically secured by IPsec at the IP layer.

History

In December 1993, the Software IP Encryption protocol swIPe (protocol) was researched at Columbia University and AT&T Bell Labs by John Ioannidis and others.

Based on the funding from the Clinton administration in hosting whitehouse.gov email (from June 1 of 1993 to January 20 of 1995) at Trusted Information Systems, Wei Xu started in July 1994 the research on IP Security, enhanced the IP protocols, developed the IPSec product on the BSDI platform, and quickly extended it on to Sun OS, HP UX, and other UNIX systems. Upon the success, Wei was facing another challenge by the slow performance of computing DES and Triple DES. The assembly software encryption was unable to support even a T1 speed under the Intel 80386 architecture. By exporting the Crypto cards from Germany, Wei further developed an automated device driver, known as plug-and-play today, in integrating with the hardware Crypto. After achieving the throughput much higher than a T1s, Wei Xu finally made the commercial product practically feasible, that was released as a part of the well-known Gauntlet firewall. In December 1994, it was deployed for the first time in production for securing some remote sites between east and west coastal states of the United States.

Another IP Encapsulating Security Payload (ESP) was researched at the Naval Research Laboratory as part of a DARPA-sponsored research project, with openly published by IETF SIPP Working Group drafted in December 1993 as a security extension for SIPP. This ESP was originally derived from the US Department of Defense SP3D protocol, rather than being derived from the ISO Network-Layer Security Protocol (NLSP). The SP3D protocol specification was published by NIST, but designed by the Secure Data Network System project of the US Department of Defense. The Security Authentication Header (AH) is derived partially from previous IETF standards work for authentication of the Simple Network Management Protocol (SNMP) version 2.

In 1995, The IPsec working group in the IETF was started to create an open freely available and vetted version of protocols that had been developed under NSA contract in the Secure Data Network System (SDNS) project. The SDNS project had defined a Security Protocol Layer 3 (SP3) that had been published by NIST and was also the basis of the ISO Network Layer Security Protocol (NLSP). Key management for SP3 was provided by the Key Management Protocol (KMP) that provided a baseline of ideas for subsequent work in the IPsec committee.

IPsec is officially standardised by the Internet Engineering Task Force (IETF) in a series of Request for Comments documents addressing various components and extensions. It specifies the spelling of the protocol name to be IPsec

Security architecture

The IPsec suite is an open standard. IPsec uses the following protocols to perform various functions:

- Authentication Headers (AH) provide connectionless data integrity and data origin authentication for IP datagrams and provides protection against replay attacks
- Encapsulating Security Payloads (ESP) provide confidentiality, data-origin authentication, connectionless integrity, an anti-replay service (a form of partial sequence integrity), and limited traffic-flow confidentiality
- Security Associations (SA) provide the bundle of algorithms and data that provide the parameters necessary for AH and/or ESP operations. The Internet Security Association and Key Management Protocol (ISAKMP) provides a framework for authentication and key exchange,[10] with actual authenticated keying material provided either by manual configuration with pre-shared keys, Internet Key Exchange (IKE and IKEv2), Kerberized Internet Negotiation of Keys (KINK), or IPSECKEY DNS records

Security association

The IP security architecture uses the concept of a security association as the basis for building security functions into IP. A security association is simply the bundle of algorithms and parameters (such as keys) that is being used to encrypt and authenticate a particular flow in one direction. Therefore, in normal bi-directional traffic, the flows are secured by a pair of security associations.

Security associations are established using the Internet Security Association and Key Management Protocol (ISAKMP). ISAKMP is implemented by manual configuration with pre-shared secrets, Internet Key Exchange (IKE and IKEv2), Kerberized Internet Negotiation of Keys (KINK), and the use of IPSECKEY DNS records. RFC 5386 defines Better-Than-Nothing Security (BTNS) as an unauthenticated mode of IPsec using an extended IKE protocol.

In order to decide what protection is to be provided for an outgoing packet, IPsec uses the Security Parameter Index (SPI), an index to the security association database (SADB), along with the destination address in a packet

header, which together uniquely identify a security association for that packet. A similar procedure is performed for an incoming packet, where IPsec gathers decryption and verification keys from the security association database.

For multicast, a security association is provided for the group, and is duplicated across all authorized receivers of the group. There may be more than one security association for a group, using different SPIs, thereby allowing multiple levels and sets of security within a group. Indeed, each sender can have multiple security associations, allowing authentication, since a receiver can only know that someone knowing the keys sent the data. Note that the relevant standard does not describe how the association is chosen and duplicated across the group; it is assumed that a responsible party will have made the choice.

Modes of operation

IPsec can be implemented in a host-to-host transport mode, as well as in a network tunneling mode.

- Transport mode

In transport mode, only the payload of the IP packet is usually encrypted or authenticated. The routing is intact, since the IP header is neither modified nor encrypted; however, when the authentication header is used, the IP addresses cannot be modified by network address translation, as this always invalidates the hash value. The transport and application layers are always secured by a hash, so they cannot be modified in any way, for example by translating the port numbers.

A means to encapsulate IPsec messages for NAT traversal has been defined by RFC documents describing the NAT-T mechanism.

- Tunnel mode

In tunnel mode, the entire IP packet is encrypted and authenticated. It is then encapsulated into a new IP packet with a new IP header. Tunnel mode is used to create virtual private networks for network-to-network communications (e.g. between routers to link sites), host-to-network communications (e.g. remote user access) and host-to-host communications (e.g. private chat)

Tunnel mode supports NAT traversal.

For the Protocol++ interface to IPsec see

For API Documentation:

See Also

[ProtocolPP::jipsecsa](#)
[ProtocolPP::jipsec](#)
[ProtocolPP::jprotocolpp](#)

For Additional Documentation:

See Also

[jipsecsa](#)
[jipsec](#)
[jprotocolpp](#)

1.12 Macsec

See https://en.wikipedia.org/wiki/IEEE_802.1AE

802.1AE is the IEEE MAC Security standard (also known as MACsec) which defines connectionless data confidentiality and integrity for media access independent protocols. It is standardized by the IEEE 802.1 working group

Details

Key management and the establishment of secure associations is outside the scope of 802.1AE, but is specified by 802.1X-2010

The 802.1AE standard specifies the implementation of a MAC Security Entities (SecY) that can be thought of as part of the stations attached to the same LAN, providing secure MAC service to the client. The standard defines

- MACsec frame format, which is similar to the Ethernet frame, but includes additional fields:
 - Security Tag, which is an extension of the EtherType
 - Message authentication code (ICV)
- Secure Connectivity Associations that represent groups of stations connected via unidirectional Secure Channels
- Security Associations within each secure channel. Each association uses its own key (SAK). More than one association is permitted within the channel for the purpose of key change without traffic interruption (standard requires devices to support at least two)
- A default cipher suite of GCM-AES-128 (Galois/Counter Mode of Advanced Encryption Standard cipher with 128-bit key)
 - GCM-AES-256 using a 256 bit key was added to the standard 5 years later.

Security tag inside each frame in addition to EtherType includes:

- association number within the channel
- packet number to provide unique initialization vector for encryption and authentication algorithms as well as protection against replay attack
- optional LAN-wide secure channel identifier (not required on point-to-point links)

The IEEE 802.1AE (MACsec) standard specifies a set of protocols to meet the security requirements for protecting data traversing Ethernet LANs

MACsec allows unauthorised LAN connections to be identified and excluded from communication within the network. In common with IPsec and SSL, MACsec defines a security infrastructure to provide data confidentiality, data integrity and data origin authentication

By assuring that a frame comes from the station that claimed to send it, MACSec can mitigate attacks on Layer 2 protocols.

For the Protocol++ interface to Macsec see

For API Documentation:

See Also

[ProtocolPP::jmacsecsa](#)
[ProtocolPP::jmacsec](#)
[ProtocolPP::jprotocolpp](#)

For API Documentation:

See Also

[jmacsecsa](#)
[jmacsec](#)
[jprotocolpp](#)

1.13 SRTP

See https://en.wikipedia.org/wiki/Secure_Real-time_Transport_Protocol

The Secure Real-time Transport Protocol (or SRTP) defines a profile of RTP (Real-time Transport Protocol), intended to provide encryption, message authentication and integrity, and replay protection to the RTP data in both unicast and multicast applications. It was developed by a small team of IP protocol and cryptographic experts from

Cisco and Ericsson including David Oran, David McGrew, Mark Baugher, Mats Naslund, Elisabetta Carrara, Karl Norman, and Rolf Blom. It was first published by the IETF in March 2004 as RFC 3711.

Since RTP is closely related to RTCP (Real Time Control Protocol) which can be used to control the RTP session, SRTP also has a sister protocol, called Secure RTCP (or SRTCP); SRTCP provides the same security-related features to RTCP, as the ones provided by SRTP to RTP.

Utilization of SRTP or SRTCP is optional to the utilization of RTP or RTCP; but even if SRTP/SRTCP are used, all provided features (such as encryption and authentication) are optional and can be separately enabled or disabled. The only exception is the message authentication feature which is indispensably required when using SRTCP.

Data flow encryption

For encryption and decryption of the data flow (and hence for providing confidentiality of the data flow), SRTP (together with SRTCP) utilizes AES as the default cipher. There are two cipher modes defined which allow the original block cipher AES to be used as a stream cipher:

Segmented Integer Counter Mode

A typical counter mode, which allows random access to any blocks, which is essential for RTP traffic running over unreliable network with possible loss of packets. In the general case, almost any function can be used in the role of "counter", assuming that this function does not repeat for a large number of iterations. But the standard for encryption of RTP data is just a usual integer incremental counter. AES running in this mode is the default encryption algorithm, with a default encryption key length of 128 bits and a default session salt key length of 112 bits.

f8-mode (not supported)

Integrity protection

To authenticate the message and protect its integrity, the HMAC-SHA1 algorithm (defined in RFC 2104) is used, which produces a 160-bit result, which is then truncated to 80 or 32 bits to become the authentication tag appended to the packet. The HMAC is calculated over the packet payload and material from the packet header, including the packet sequence number. To protect against replay attacks, the receiver maintains the indices of previously received messages, compares them with the index of each new received message and admits the new message only if it has not been played (i.e. sent) before. Such an approach heavily relies on the integrity protection being enabled (to make it impossible to spoof message indices).

Key derivation

A key derivation function is used to derive the different keys used in a crypto context (SRTP and SRTCP encryption keys and salts, SRTP and SRTCP authentication keys) from one single master key in a cryptographically secure way. Thus, the key management protocol needs to exchange only one master key, all the necessary session keys are generated by applying the key derivation function.

Periodical application of the key derivation function will result in security benefits. It prevents an attacker from collecting large amounts of ciphertext encrypted with one single session key. Certain attacks are easier to carry out when a large amount of ciphertext is available. Furthermore, multiple applications of the key derivation function provides backwards and forward security in the sense that a compromised session key does not compromise other session keys derived from the same master key. This means that even if an attacker managed to recover a certain session key, he is not able to decrypt messages secured with previous and later session keys derived from the same master key. (Note that, of course, a leaked master key reveals all the session keys derived from it.)

SRTP relies on an external key management protocol to set up the initial master key. Two protocols specifically designed to be used with SRTP are ZRTP and MIKEY.

There are also other methods to negotiate the SRTP keys. There are several vendors which offer products that use the SDES key exchange method.

For the Protocol++ interface to SRTP see

For API Documentation:

See Also

[ProtocolPP::jsrtpsa](#)
[ProtocolPP::jsrtp](#)
[ProtocolPP::jprotocolpp](#)

For Additional Documentation:

See Also

[jsrtpsa](#)
[jsrtp](#)

1.14 Wifi/WiGig

See <https://en.wikipedia.org/wiki/Wi-Fi>

See https://en.wikipedia.org/wiki/Wireless_Gigabit_Alliance



Figure 1.4: Wifi-Wigig Logos

Wi-Fi or WiFi[2] is a technology that allows electronic devices to connect to a wireless LAN (WLAN) network, mainly using the 2.4 gigahertz (12 cm) UHF and 5 gigahertz (6 cm) SHF ISM radio bands. A WLAN is usually password protected, but may be open, which allows any device within its range to access the resources of the WLAN network.

The Wi-Fi Alliance defines Wi-Fi as any "wireless local area network" (WLAN) product based on the Institute of Electrical and Electronics Engineers' (IEEE) 802.11 standards.[1] However, the term "Wi-Fi" is used in general English as a synonym for "WLAN" since most modern WLANs are based on these standards. "Wi-Fi" is a trademark of the Wi-Fi Alliance. The "Wi-Fi Certified" trademark can only be used by Wi-Fi products that successfully complete Wi-Fi Alliance interoperability certification testing.

Devices which can use Wi-Fi technology include personal computers, video-game consoles, smartphones, digital cameras, tablet computers and digital audio players. Wi-Fi compatible devices can connect to the Internet via a WLAN network and a wireless access point. Such an access point (or hotspot) has a range of about 20 meters (66 feet) indoors and a greater range outdoors. Hotspot coverage can be as small as a single room with walls that block radio waves, or as large as many square kilometres achieved by using multiple overlapping access points.

History

In 1971, ALOHAnet connected the Hawaiian Islands with a UHF wireless packet network. ALOHAnet and the ALOHA protocol were early forerunners to Ethernet, and later the IEEE 802.11 protocols, respectively.

A 1985 ruling by the U.S. Federal Communications Commission released the ISM band for unlicensed use. These frequency bands are the same ones used by equipment such as microwave ovens and are subject to interference.

In 1991, NCR Corporation with AT&T Corporation invented the precursor to 802.11, intended for use in cashier systems. The first wireless products were under the name WaveLAN.

The Australian radio-astronomer Dr John O'Sullivan with his colleagues Dr Terrence Percival AM, Mr Graham Daniels, Mr Diet Ostry, Mr John Deane developed a key patent used in Wi-Fi as a by-product of a Commonwealth Scientific and Industrial Research Organisation (CSIRO) research project, "a failed experiment to detect exploding mini black holes the size of an atomic particle". In 1992 and 1996, CSIRO obtained patents for a method later used in Wi-Fi to "unsmear" the signal

The first version of the 802.11 protocol was released in 1997, and provided up to 2 Mbit/s link speeds. This was updated in 1999 with 802.11b to permit 11 Mbit/s link speeds, and this proved to be popular.

In 1999, the Wi-Fi Alliance formed as a trade association to hold the Wi-Fi trademark under which most products are sold

Wi-Fi uses a large number of patents held by many different organizations.[9] In April 2009, 14 technology companies agreed to pay CSIRO \$250 million for infringements on CSIRO patents. This led to Australians labeling Wi-Fi as a Canberra invention, though this has been the subject of some controversy. CSIRO won a further \$220 million settlement for Wi-Fi patent-infringements in 2012 with global firms in the United States required to pay the CSIRO licensing rights estimated to be worth an additional \$1 billion in royalties

Etymology

The term Wi-Fi, commercially used at least as early as August 1999,[16] was coined by brand-consulting firm Interbrand Corporation. The Wi-Fi Alliance had hired Interbrand to determine a name that was "a little catchier than 'IEEE 802.11b Direct Sequence'". Phil Belanger, a founding member of the Wi-Fi Alliance who presided over the selection of the name "Wi-Fi", also stated that Interbrand invented Wi-Fi as a play on words with hi-fi, and also created the Wi-Fi logo.

The Wi-Fi Alliance used the "nonsense" advertising slogan "The Standard for Wireless Fidelity" for a short time after the brand name was invented, leading to the misconception that Wi-Fi was an abbreviation of "Wireless Fidelity". The yin-yang Wi-Fi logo indicates the certification of a product for interoperability

Non-Wi-Fi technologies intended for fixed points, such as Motorola Canopy, are usually described as fixed wireless. Alternative wireless technologies include mobile phone standards, such as 2G, 3G, 4G or LTE.

The name is often written as WiFi or Wifi, but these are not approved by the Wi-Fi Alliance.

IEEE 802.11 standard

The IEEE 802.11 standard is a set of media access control (MAC) and physical layer (PHY) specifications for implementing wireless local area network (WLAN) computer communication in the 2.4, 3.6, 5, and 60 GHz frequency bands. They are created and maintained by the IEEE LAN/MAN Standards Committee (IEEE 802). The base version of the standard was released in 1997, and has had subsequent amendments. The standard and amendments provide the basis for wireless network products using the Wi-Fi brand. While each amendment is officially revoked when it is incorporated in the latest version of the standard, the corporate world tends to market to the revisions because they concisely denote capabilities of their products.[28] As a result, in the market place, each revision tends to become its own standard.

Uses

To connect to a Wi-Fi LAN, a computer has to be equipped with a wireless network interface controller. The combination of computer and interface controller is called a station. For all stations that share a single radio frequency communication channel, transmissions on this channel are received by all stations within range. The transmission is not guaranteed to be delivered and is therefore a best-effort delivery mechanism. A carrier wave is used to transmit the data. The data is organised in packets on an Ethernet link, referred to as "Ethernet frames"

Network security

The main issue with wireless network security is its simplified access to the network compared to traditional wired networks such as Ethernet. With wired networking, one must either gain access to a building (physically connecting into the internal network), or break through an external firewall. To enable Wi-Fi, one merely needs to be within the range of the Wi-Fi network. Most business networks protect sensitive data and systems by attempting to disallow external access. Enabling wireless connectivity reduces security if the network uses inadequate or no encryption

An attacker who has gained access to a Wi-Fi network router can initiate a DNS spoofing attack against any other user of the network by forging a response before the queried DNS server has a chance to reply

Data security risks

The most common wireless encryption-standard, Wired Equivalent Privacy (WEP), has been shown to be easily breakable even when correctly configured. Wi-Fi Protected Access (WPA and WPA2) encryption, which became available in devices in 2003, aimed to solve this problem. Wi-Fi access points typically default to an encryption-free (open) mode. Novice users benefit from a zero-configuration device that works out-of-the-box, but this default does not enable any wireless security, providing open wireless access to a LAN. To turn security on requires the user to configure the device, usually via a software graphical user interface (GUI). On unencrypted Wi-Fi networks

connecting devices can monitor and record data (including personal information). Such networks can only be secured by using other means of protection, such as a VPN or secure Hypertext Transfer Protocol over Transport Layer Security (HTTPS).

Wi-Fi Protected Access encryption (WPA2) is considered secure, provided a strong passphrase is used. A proposed modification to WPA2 is WPA-OTP or WPA3, which stores an on-chip optically generated onetime pad on all connected devices which is periodically updated via strong encryption then hashed with the data to be sent or received. This would be unbreakable using any (even quantum) computer system as the hashed data is essentially random and no pattern can be detected if it is implemented properly. Main disadvantage is that it would need multi-GB storage chips so would be expensive for the consumers.

WiGig

WiGig is an extension to Wifi specified in 802.11ad-2012 to allow high speed transfers in the 60GHz range up to ~7Gbps. The high frequency limits the range of the radio to ~30 feet requiring access points in the same room as the user. Most things remain the same except when type=1 and subtype=6 (Control Frame with Extension). Packet ordering after the subtype changes in this case to facilitate higher speeds

For the Protocol++ interface to Wifi/WiGig see

For API Documentation:

See Also

[ProtocolPP::jwifisa](#)
[ProtocolPP::jwifi](#)
[ProtocolPP::jprotocolpp](#)

For Additional Documentation:

See Also

[jwifisa](#)
[jwifi](#)
[jprotocolpp](#)

1.15 WiMax

See <https://en.wikipedia.org/wiki/WiMAX>



Figure 1.5: WiMAX Forum Logo

WiMAX (Worldwide Interoperability for Microwave Access)[3] is a family of wireless communications standards initially designed to provide 30 to 40 megabit-per-second data rates, with the 2011 update providing up to 1 Gbit/s for fixed stations. The name "WiMAX" was created by the WiMAX Forum, which was formed in June 2001 to promote conformity and interoperability of the standard. The forum describes WiMAX as "a standards-based technology enabling the delivery of last mile wireless broadband access as an alternative to cable and DSL". IEEE 802.16m or WirelessMAN-Advanced is a candidate for the 4G, in competition with the LTE Advanced standard.

Terminology

WiMAX refers to interoperable implementations of the IEEE 802.16 family of wireless-networks standards ratified by the WiMAX Forum. (Similarly, Wi-Fi refers to interoperable implementations of the IEEE 802.11 Wireless LAN standards certified by the Wi-Fi Alliance.) WiMAX Forum certification allows vendors to sell fixed or mobile products as WiMAX certified, thus ensuring a level of interoperability with other certified products, as long as they fit the same profile.

The original IEEE 802.16 standard (now called "Fixed WiMAX") was published in 2001. WiMAX adopted some of its technology from WiBro, a service marketed in Korea

Mobile WiMAX (originally based on 802.16e-2005) is the revision that was deployed in many countries, and is the basis for future revisions such as 802.16m-2011.

WiMAX is sometimes referred to as "Wi-Fi on steroids" and can be used for a number of applications including broadband connections, cellular backhaul, hotspots, etc. It is similar to Wi-Fi, but it can enable usage at much greater distances

Uses

The bandwidth and range of WiMAX make it suitable for the following potential applications:

- Providing portable mobile broadband connectivity across cities and countries through a variety of devices.
- Providing a wireless alternative to cable and digital subscriber line (DSL) for "last mile" broadband access.
- Providing data, telecommunications (VoIP) and IPTV services (triple play).
- Providing a source of Internet connectivity as part of a business continuity plan.

- Smart grids and metering

Internet access

WiMAX can provide at-home or mobile Internet access across whole cities or countries. In many cases this has resulted in competition in markets which typically only had access through an existing incumbent DSL (or similar) operator

Additionally, given the relatively low costs associated with the deployment of a WiMAX network (in comparison with 3G, HSDPA, xDSL, HFC or FTTx), it is now economically viable to provide last-mile broadband Internet access in remote locations.

The IEEE 802.16 Standard

WiMAX is based upon IEEE Std 802.16e-2005,[13] approved in December 2005. It is a supplement to the IEEE Std 802.16-2004, and so the actual standard is 802.16-2004 as amended by 802.16e-2005. Thus, these specifications need to be considered together.

IEEE 802.16e-2005 improves upon IEEE 802.16-2004 by:

- Adding support for mobility (soft and hard handover between base stations). This is seen as one of the most important aspects of 802.16e-2005, and is the very basis of Mobile WiMAX.
- Scaling of the fast Fourier transform (FFT) to the channel bandwidth in order to keep the carrier spacing constant across different channel bandwidths (typically 1.25 MHz, 5 MHz, 10 MHz or 20 MHz). Constant carrier spacing results in a higher spectrum efficiency in wide channels, and a cost reduction in narrow channels. Also known as scalable OFDMA (SOFDMA). Other bands not multiples of 1.25 MHz are defined in the standard, but because the allowed FFT subcarrier numbers are only 128, 512, 1024 and 2048, other frequency bands will not have exactly the same carrier spacing, which might not be optimal for implementations. Carrier spacing is 10.94 kHz.
- Advanced antenna diversity schemes, and hybrid automatic repeat-request (HARQ)
- Adaptive antenna systems (AAS) and MIMO technology
- Denser sub-channelization, thereby improving indoor penetration
- Intro and low-density parity check (LDPC)
- Introducing downlink sub-channelization, allowing administrators to trade coverage for capacity or vice versa
- Adding an extra quality of service (QoS) class for VoIP applications.

SOFDMA (used in 802.16e-2005) and OFDM256 (802.16d) are not compatible thus equipment will have to be replaced if an operator is to move to the later standard (e.g., Fixed WiMAX to Mobile WiMAX).

Comparison

Comparisons and confusion between WiMAX and Wi-Fi are frequent, because both are related to wireless connectivity and Internet access

- WiMAX is a long range system, covering many kilometres, that uses licensed or unlicensed spectrum to deliver connection to a network, in most cases the Internet.
- Wi-Fi uses the 2.4 GHz, 3 GHz, 5 GHz, and 60 GHz radio frequency bands to provide access to a local network.
- Wi-Fi is more popular in end-user devices.
- Wi-Fi runs on the Media Access Control's CSMA/CA protocol, which is connectionless and contention based, whereas WiMAX runs a connection-oriented MAC.
- WiMAX and Wi-Fi have quite different quality of service (QoS) mechanisms:
 - WiMAX uses a QoS mechanism based on connections between the base station and the user device. Each connection is based on specific scheduling algorithms.

- Wi-Fi uses contention access — all subscriber stations that wish to pass data through a wireless access point (AP) are competing for the AP's attention on a random interrupt basis. This can cause subscriber stations distant from the AP to be repeatedly interrupted by closer stations, greatly reducing their throughput.

- Both IEEE 802.11, which includes Wi-Fi, and IEEE 802.16, which includes WiMAX, define Peer-to-Peer (P2P) and wireless ad hoc networks, where an end user communicates to users or servers on another Local Area Network (LAN) using its access point or base station. However, 802.11 supports also direct ad hoc or peer to peer networking between end user devices without an access point while 802.16 end user devices must be in range of the base station.

Although Wi-Fi and WiMAX are designed for different situations, they are complementary. WiMAX network operators typically provide a WiMAX Subscriber Unit that connects to the metropolitan WiMAX network and provides Wi-Fi connectivity within the home or business for local devices, e.g., computers, Wi-Fi handsets and smartphones. This enables the user to place the WiMAX Subscriber Unit in the best reception area, such as a window, and still be able to use the WiMAX network from any place within their residence.

The local area network inside one's house or business would operate as with any other wired or wireless network. If one were to connect the WiMAX Subscriber Unit directly to a WiMAX-enabled computer, that would limit access to a single device. As an alternative for a LAN, one could purchase a WiMAX modem with a built-in wireless Wi-Fi router, allowing one to connect multiple devices to create a LAN.

Using WiMAX could be an advantage, since it is typically faster than most cable modems with download speeds between 3 and 6 Mbit/s, and generally costs less than cable.

For the Protocol++ interface to WiMax see

For API Documentation:

See Also

[ProtocolPP::jwimaxsa](#)
[ProtocolPP::jwimax](#)
[ProtocolPP::jprotocolpp](#)

For Additional Documentation:

See Also

[jwimaxsa](#)
[jwimax](#)
[jprotocolpp](#)

1.16 Long Term Evolution (LTE)

See [https://en.wikipedia.org/wiki/LTE_\(telecommunication\)](https://en.wikipedia.org/wiki/LTE_(telecommunication))



Figure 1.6: LTE and LTE-Advanced Logos

LTE stands for Long Term Evolution[4] and is a registered trademark owned by ETSI (European Telecommunications Standards Institute) for the wireless data communications technology and a development of the GSM/UMTS standards. However, other nations and companies do play an active role in the LTE project. The goal of LTE was to increase the capacity and speed of wireless data networks using new DSP (digital signal processing) techniques and modulations that were developed around the turn of the millennium. A further goal was the redesign and simplification of the network architecture to an IP-based system with significantly reduced transfer latency compared to the 3G architecture. The LTE wireless interface is incompatible with 2G and 3G networks, so that it must be operated on a separate radio spectrum

LTE was first proposed by NTT DoCoMo of Japan in 2004, and studies on the new standard officially commenced in 2005. In May 2007, the LTE/SAE Trial Initiative (LSTI) alliance was founded as a global collaboration between vendors and operators with the goal of verifying and promoting the new standard in order to ensure the global introduction of the technology as quickly as possible. The LTE standard was finalized in December 2008, and the first publicly available LTE service was launched by TeliaSonera in Oslo and Stockholm on December 14, 2009 as a data connection with a USB modem. The LTE services were launched by major North American carriers as well, with the Samsung SCH-r900 being the world's first LTE Mobile phone starting on September 21, 2010 and Samsung Galaxy Indulge being the world's first LTE smartphone starting on February 10, 2011 both offered by MetroPCS and HTC ThunderBolt offered by Verizon starting on March 17 being the second LTE smartphone to be sold commercially. In Canada, Rogers Wireless was the first to launch LTE network on July 7, 2011 offering the Sierra Wireless AirCard® 313U USB mobile broadband modem, known as the "LTE Rocket™ stick" then followed closely by mobile devices from both HTC and Samsung. Initially, CDMA operators planned to upgrade to rival standards called UMB and WiMAX, but all the major CDMA operators (such as Verizon, Sprint and MetroPCS in the United States, Bell and Telus in Canada, au by KDDI in Japan, SK Telecom in South Korea and China Telecom/China Unicom in China) have announced that they intend to migrate to LTE after all. The evolution of LTE is LTE Advanced, which was standardized in March 2011. Services are expected to commence in 2013. Additional evolution known as LTE Advanced Pro have been approved in year 2015

The LTE specification provides downlink peak rates of 300Mbit/s, uplink peak rates of 75Mbit/s and QoS provisions permitting a transfer latency of less than 5ms in the radio access network. LTE has the ability to manage fast-moving mobiles and supports multi-cast and broadcast streams. LTE supports scalable carrier bandwidths, from 1.4MHz to 20MHz and supports both frequency division duplexing (FDD) and time-division duplexing (TDD). The IP-based network architecture, called the Evolved Packet Core (EPC) designed to replace the GPRS Core Network, supports seamless handovers for both voice and data to cell towers with older network technology such as GSM, UMTS and CDMA2000. The simpler architecture results in lower operating costs (for example, each E-UTRA cell will support up to four times the data and voice capacity supported by HSPA)

For the Protocol++ interface to LTE see

For API Documentation:

See Also

[ProtocolPP::jltesa](#)
[ProtocolPP::jlte](#)
[ProtocolPP::jprotocolpp](#)

For Additional Documentation:

See Also

[jltesa](#)
[jlte](#)
[jprotocolpp](#)

1.17 Digital Signature Algorithm (DSA)

For the Protocol++ interface to DSA see

For API Documentation:

See Also

[ProtocolPP::jdsa](#)
[ProtocolPP::jprotocolpp](#)

For Additional Documentation:

See Also

[jdsa](#)
[jprotocolpp](#)

1.18 Elliptic Curve Digital Signature Algorithm (ECDSA)

For the Protocol++ interface to ECDSA see

For API Documentation:

See Also

[ProtocolPP::jecdsa](#)
[ProtocolPP::jprotocolpp](#)

For Additional Documentation:

See Also

[jecdsa](#)
[jprotocolpp](#)

1.19 RSA CryptoSystem

For the Protocol++ interface to RSA see

For API Documentation:

See Also

[ProtocolPP::jrsa](#)
[ProtocolPP::jprotocolpp](#)

For Additional Documentation:

See Also

[jrsa](#)
[jprotocolpp](#)

1.20 W.A.S.P

W.A.S.P is an interface to allow protocol++ to connect to a testbench. Data is either randomly generated or can be passed to the interface using an XML format. Users need to overload the virtual functions to connect to their own testbench. The included makefile can create static and shared object libraries for W.A.S.P and Protocol++

For the W.A.S.P interface see

For API Documentation:

See Also

[ProtocolPP::wasp](#)

For Additional Documentation:

See Also

[wasp](#)

1. <http://us.norton.com/norton-security-with-backup>
2. Wifi
3. WiMax
4. LTE

For API Documentation:

See Also

[ProtocolPP::jsecass](#)
[ProtocolPP::jprotocol](#)
[ProtocolPP::judpsa](#)
[ProtocolPP::judp](#)
[ProtocolPP::jtcpssa](#)
[ProtocolPP::jtcp](#)
[ProtocolPP::jicmpsa](#)
[ProtocolPP::jicmp](#)
[ProtocolPP::jipsa](#)
[ProtocolPP::jip](#)
[ProtocolPP::jtlsa](#)
[ProtocolPP::jtls](#)
[ProtocolPP::jipsecasa](#)
[ProtocolPP::jipsec](#)
[ProtocolPP::jmacsecsa](#)
[ProtocolPP::jmacsec](#)
[ProtocolPP::jsrtpsa](#)

```
ProtocolPP::jsrtp
ProtocolPP::jwifisa
ProtocolPP::jwifi
ProtocolPP::jwimaxsa
ProtocolPP::jwimax
ProtocolPP::jltesa
ProtocolPP::jlte
ProtocolPP::jds
ProtocolPP::jcdsa
ProtocolPP::jrsa
ProtocolPP::wasp
```

For Additional Documentation:**See Also**

```
jenum
jsecass
jprotocol
judpsa
judp
jtcpsa
jtcp
jicmpsa
jicmp
jipsa
jip
jtlsa
jtls
jipsecsa
jipsec
jmacsecsa
jmacsec
jsrtpsa
jsrtp
jwifisa
jwifi
jwimaxsa
jwimax
jltesa
jlte
jds
jcdsa
jrsa
wasp
```

Protocol++ written by : John Peter Greninger © John Peter Greninger 2015-2019 All Rights Reserved

All copyrights and trademarks are the property of their respective owners

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer
- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution

- Any and all modifications must be returned to John Peter Greninger at GitHub.com <https://github.com/jpgreninger/protocolpp> for evaluation. Inclusion of modifications in the source code shall be determined solely by John Peter Greninger. Failure to provide modifications shall render this license NULL and VOID and revoke any rights to use of Protocol++
- Commercial use requires a fee-based license obtainable at www.protocolpp.com
- Academic use requires written and notarized permission from John Peter and Sheila Rocha Greninger
- US Copyrights at [https://www.copyright.gov/](http://www.copyright.gov/)
 - TXu002059872 (Version 1.0.0)
 - TXu002066632 (Version 1.2.7)
 - TXu002082674 (Version 1.4.0)
 - TXu002097880 (Version 2.0.0)
 - VAu001334497 (JPGNetworks)

The names of its contributors may not be used to endorse or promote products derived from this software without specific prior written permission

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE

Chapter 2

Namespace Index

2.1 Namespace List

Here is a list of all namespaces with brief descriptions:

DriverPP	49
InterfacePP	49
option	The namespace of The Lean Mean C++ Option Parser	50
PlatformPP	54
ProtocolPP	55
tinyxml2	172

Chapter 3

Hierarchical Index

3.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

option::Parser::Action	175
option::Parser::StoreOptionAction	753
option::Stats::CountOptionsAction	188
aead_chacha_poly1305	176
ProtocolPP::aead_chacha_poly1305	179
option::Arg	180
chacha20	182
ProtocolPP::chacha20	183
ciphers	185
ProtocolPP::ciphers	187
option::Descriptor	188
tinyxml2::DynArray< T, INITIAL_SIZE >	191
tinyxml2::DynArray< Block *, 10 >	191
tinyxml2::DynArray< char, 20 >	191
tinyxml2::DynArray< const char *, 10 >	191
tinyxml2::DynArray< tinyxml2::XMLNode *, 10 >	191
EnumString	192
ProtocolPP::EnumStringBase< DerivedType, EnumType >	194
ProtocolPP::EnumStringBase< EnumString< EnumType >, EnumType >	194
ProtocolPP::EnumString< EnumType >	193
std::exception	
InterfacePP::jring< TR >	504
ProtocolPP::jikeparse	265
ProtocolPP::wasp	763
ikev2	200
interfacepp	201
option::PrintUsageImplementation::IStringWriter	202
option::PrintUsageImplementation::FunctionWriter< Function >	199
option::PrintUsageImplementation::OStreamWriter< OStream >	724
option::PrintUsageImplementation::StreamWriter< Function, Stream >	754
option::PrintUsageImplementation::SyscallWriter< Syscall >	757
option::PrintUsageImplementation::TemporaryWriter< Temporary >	758
jarray	202
ProtocolPP::jarray< T >	203
ProtocolPP::jarray< uint8_t >	203
jdata	212
ProtocolPP::jdata	213

jdirectdrive	215
DriverPP::jdirectdrive	216
jdrive	218
DriverPP::jdrive	219
jdriver	222
DriverPP::jdriver< Q >	223
jdsa	225
ProtocolPP::jdsa	228
jecdsa	230
ProtocolPP::jecdsa	233
jenum	236
jexec	237
InterfacePP::jexec	238
jicmp	240
jicmpsa	249
ProtocolPP::jikeparse::jikecfg	257
ProtocolPP::jikencrypt	263
jikencrypt	264
jikeparse	267
ProtocolPP::jikeparse::jikepolicy	268
ProtocolPP::jikeprf	269
jikeprf	271
ProtocolPP::jikev2	272
jikev2	275
ProtocolPP::jikev2dh	297
jikev2dh	298
jikev2sa	300
jip	323
jipsa	340
jipsec	348
jipsecsa	374
InterfacePP::jlogger	385
jlte	389
jtesa	393
jmacsec	399
jmacsecsa	422
InterfacePP::jmamu	431
jmmu	433
ProtocolPP::jmodes	434
jmodes	438
jpacket	452
ProtocolPP::jpacket	453
ProtocolPP::jpoly1305	459
jpoly1305	460
ProtocolPP::jpoly1305_state_internal_t	461
jproducer	462
InterfacePP::jproducer	464
InterfacePP::jtestbench	586
jprotocol	468
ProtocolPP::jprotocol	469
ProtocolPP::jicmp	246
ProtocolPP::jip	335
ProtocolPP::jipsec	370
ProtocolPP::jlte	391
ProtocolPP::jmacsec	420
ProtocolPP::jsrtp	539
ProtocolPP::jtcp	575
ProtocolPP::jtls	605

ProtocolPP::judp	612
ProtocolPP::jwifi	621
ProtocolPP::jwimax	690
jprotocolpp	482
ProtocolPP::jprotocolpp	482
jrand	486
ProtocolPP::jrand	487
jreplay	494
ProtocolPP::jreplay< T, TE >	496
jresponder	501
InterfacePP::jresponder	501
jring	503
jringdrive	506
DriverPP::jringdrive	508
jrsa	509
ProtocolPP::jrsa	513
jsec	515
PlatformPP::jsec	516
ProtocolPP::jsecass	518
ProtocolPP::jicmpsa	252
ProtocolPP::jikev2sa	321
ProtocolPP::jipsa	345
ProtocolPP::jipsecsa	381
ProtocolPP::jltesa	396
ProtocolPP::jmacsecsa	429
ProtocolPP::jsrtpsa	558
ProtocolPP::jtcpса	582
ProtocolPP::jtlsa	608
ProtocolPP::judpsa	616
ProtocolPP::jwifisa	651
ProtocolPP::jwimaxsa	693
jsecass	519
jsecproducer	521
InterfacePP::jsecproducer	522
InterfacePP::jsectestbench	530
jsecresponder	527
InterfacePP::jsecresponder	528
jsectestbench	533
jsgt	534
PlatformPP::jsgt	535
jsnow3g	536
ProtocolPP::jsnow3g	537
jsrtp	543
jsrtpsa	560
jstream	560
ProtocolPP::jstream	560
jtcp	565
jtcpса	579
jtestbench	585
jtls	590
jtlsa	610
judp	612
judpsa	619
jwifi	624
jwifisa	654
jwimax	680
jwimaxsa	695
jzuc	701

ProtocolPP::jzuc	702
option::PrintUsageImplementation::LinePartIterator	705
option::PrintUsageImplementation::LineWrapper	707
InterfacePP::log_policy_interface	708
InterfacePP::file_log_policy	195
InterfacePP::file_quiet_log_policy	196
InterfacePP::file_stdout_log_policy	198
log_policy_interface	709
tinyxml2::MemPool	711
tinyxml2::MemPoolT< sizeof(tinyxml2::XmlAttribute) >	712
tinyxml2::MemPoolT< sizeof(tinyxml2::XMLComment) >	712
tinyxml2::MemPoolT< sizeof(tinyxml2::XMLElement) >	712
tinyxml2::MemPoolT< sizeof(tinyxml2::XMLText) >	712
tinyxml2::MemPoolT< ITEM_SIZE >	712
ProtocolPP::MTRand_int32	716
ProtocolPP::MTRand	713
ProtocolPP::MTRand53	714
ProtocolPP::MTRand_closed	715
ProtocolPP::MTRand_open	717
option::Option	718
option::Parser	724
option::PrintUsageImplementation	730
protocolpp	731
InterfacePP::ringflow	732
InterfacePP::ringin	734
InterfacePP::ringout	736
InterfacePP::secin	738
InterfacePP::secout	739
ProtocolPP::sfmt	740
SFMT	747
PlatformPP::jsec::sgt_t	748
option::Stats	749
StdCapture	752
tinyxml2::StrPair	755
ProtocolPP::W128_T	758
wasp	759
tinyxml2::XmlAttribute	764
tinyxml2::XMLConstHandle	771
tinyxml2::XMLHandle	794
tinyxml2::XMLNode	797
tinyxml2::XMLComment	769
tinyxml2::XMLDeclaration	773
tinyxml2::XMLDocument	775
tinyxml2::XMLElement	780
tinyxml2::XMLText	811
tinyxml2::XMLUnknown	814
tinyxml2::XMLUtil	816
tinyxml2::XMLVisitor	818
tinyxml2::XMLPrinter	805
ProtocolPP::jikeparse	265
ProtocolPP::wasp	763

Chapter 4

Class Index

4.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

option::Parser::Action	175
aead_chacha_poly1305	176
ProtocolPP::aead_chacha_poly1305	179
option::Arg	
Functions for checking the validity of option arguments	180
chacha20	182
ProtocolPP::chacha20	183
ciphers	185
ProtocolPP::ciphers	187
option::Stats::CountOptionsAction	188
option::Descriptor	
Describes an option, its help text (usage) and how it should be parsed	188
tinyxml2::DynArray< T, INITIAL_SIZE >	191
EnumString	192
ProtocolPP::EnumString< EnumType >	193
ProtocolPP::EnumStringBase< DerivedType, EnumType >	194
InterfacePP::file_log_policy	195
InterfacePP::file_quiet_log_policy	196
InterfacePP::file_stdout_log_policy	198
option::PrintUsageImplementation::FunctionWriter< Function >	199
ikev2	200
interfacepp	201
option::PrintUsageImplementation::IStringWriter	202
jarray	202
ProtocolPP::jarray< T >	203
jdata	212
ProtocolPP::jdata	213
jdirectdrive	215
DriverPP::jdirectdrive	216
jdrive	218
DriverPP::jdrive	219
jdriver	222
DriverPP::jdriver< Q >	223
jdsa	225
ProtocolPP::jdsa	228
jecdsa	230
ProtocolPP::jecdsa	233
jenum	236

jexec	237
InterfacePP::jexec	238
jicmp	240
ProtocolPP::jicmp	246
jicmpsa	249
ProtocolPP::jicmpsa	252
ProtocolPP::jikeparse::jikecfg	257
ProtocolPP::jikencrypt	263
jikencrypt	264
ProtocolPP::jikeparse	265
jikeparse	267
ProtocolPP::jikeparse::jikepolicy	268
ProtocolPP::jikeprf	269
jikeprf	271
ProtocolPP::jikev2	272
jikev2	275
ProtocolPP::jikev2dh	297
jikev2dh	298
jikev2sa	300
ProtocolPP::jikev2sa	321
jip	323
ProtocolPP::jip	335
jipsa	340
ProtocolPP::jipsa	345
jipsec	348
ProtocolPP::jipsec	370
jipsecsa	374
ProtocolPP::jipsecsa	381
InterfacePP::jlogger	385
jlte	389
ProtocolPP::jlte	391
jltesa	393
ProtocolPP::jltesa	396
jmacsec	399
ProtocolPP::jmacsec	420
jmacsecsa	422
ProtocolPP::jmacsecsa	429
InterfacePP::jmmu	431
jmmu	433
ProtocolPP::jmodes	434
jmodes	438
jpacket	452
ProtocolPP::jpacket	453
ProtocolPP::jpoly1305	459
jpoly1305	460
ProtocolPP::jpoly1305_state_internal_t	461
jproducer	462
InterfacePP::jproducer	464
jprotocol	468
ProtocolPP::jprotocol	469
jprotocolpp	482
ProtocolPP::jprotocolpp	482
jrand	486
ProtocolPP::jrand	487
jreplay	494
ProtocolPP::jreplay< T, TE >	496
jresponder	501
InterfacePP::jresponder	501

jring	503
InterfacePP::jring< TR >	504
jringdrive	506
DriverPP::jringdrive	508
jrsa	509
ProtocolPP::jrsa	513
jsec	515
PlatformPP::jsec	516
ProtocolPP::jsecass	518
jsecass	519
jsecproducer	521
InterfacePP::jsecproducer	522
jsecresponder	527
InterfacePP::jsecresponder	528
InterfacePP::jsectestbench	530
jsectestbench	533
jsgt	534
PlatformPP::jsgt	535
jsnow3g	536
ProtocolPP::jsnow3g	537
ProtocolPP::jsrtp	539
jsrtp	543
ProtocolPP::jsrtpsa	558
jsrtpsa	560
jstream	560
ProtocolPP::jstream	560
jtcp	565
ProtocolPP::jtcp	575
jtcpsa	579
ProtocolPP::jtcpsa	582
jtestbench	585
InterfacePP::jtestbench	586
jtls	590
ProtocolPP::jtls	605
ProtocolPP::jtlsa	608
jtlsa	610
judp	612
ProtocolPP::judp	612
ProtocolPP::judpsa	616
judpsa	619
ProtocolPP::jwifico	621
jwifico	624
ProtocolPP::jwifisa	651
jwifisa	654
jwimax	680
ProtocolPP::jwimax	690
ProtocolPP::jwimaxsa	693
jwimaxsa	695
jzuc	701
ProtocolPP::jzuc	702
option::PrintUsageImplementation::LinePartIterator	705
option::PrintUsageImplementation::LineWrapper	707
InterfacePP::log_policy_interface	708
log_policy_interface	709
tinyxml2::MemPool	711
tinyxml2::MemPoolT< ITEM_SIZE >	712
ProtocolPP::MTRand	713
ProtocolPP::MTRand53	714

ProtocolPP::MTRand_closed	715
ProtocolPP::MTRand_int32	
Mersenne Twister random number generator	716
ProtocolPP::MTRand_open	717
option::Option	
A parsed option from the command line together with its argument if it has one	718
option::PrintUsageImplementation::OStreamWriter< OStream >	724
option::Parser	
Checks argument vectors for validity and parses them into data structures that are easier to work with	724
option::PrintUsageImplementation	730
protocolpp	731
InterfacePP::ringflow	732
InterfacePP::ringin	734
InterfacePP::ringout	736
InterfacePP::secin	738
InterfacePP::secout	739
ProtocolPP::sfmt	740
SFMT	
SIMD oriented Fast Mersenne Twister(SFMT) pseudorandom number generator using C structure	747
PlatformPP::jsec::sgt_t	748
option::Stats	
Determines the minimum lengths of the buffer and options arrays used for Parser	749
StdCapture	752
option::Parser::StoreOptionAction	753
option::PrintUsageImplementation::StreamWriter< Function, Stream >	754
tinyxml2::StrPair	755
option::PrintUsageImplementation::SyscallWriter< Syscall >	757
option::PrintUsageImplementation::TemporaryWriter< Temporary >	758
ProtocolPP::W128_T	758
wasp	759
ProtocolPP::wasp	763
tinyxml2::XMLAttribute	764
tinyxml2::XMLComment	769
tinyxml2::XMLConstHandle	771
tinyxml2::XMLDeclaration	773
tinyxml2::XMLDocument	775
tinyxml2::XMLElement	780
tinyxml2::XMLHandle	794
tinyxml2::XMLNode	797
tinyxml2::XMLPrinter	805
tinyxml2::XMLText	811
tinyxml2::XMLUnknown	814
tinyxml2::XMLUtil	816
tinyxml2::XMLVisitor	818

Chapter 5

File Index

5.1 File List

Here is a list of all files with brief descriptions:

drivers/include/jdirectdrive.h	829
drivers/include/jdrive.h	830
drivers/include/jdriver.h	830
drivers/include/jringdrive.h	831
drivers/include/StdCapture.h	832
include/aead_chacha_poly1305.h	832
include/chacha20.h	832
include/ciphers.h	833
include/config.h	833
Library configuration file	833
include/EnumString.h	837
include/jarray.h	838
include/jdata.h	838
include/jdsa.h	839
include/jecdsa.h	839
include/jenum.h	840
include/jicmp.h	854
include/jicmpsa.h	854
include/jip.h	854
include/jipsa.h	855
include/jipsec.h	855
include/jipseccsa.h	855
include/jite.h	855
include/jitesa.h	856
include/jmacsec.h	856
include/jmacseccsa.h	856
include/jmodes.h	857
include/jpacket.h	858
include/jpoly1305.h	858
include/jprotocol.h	859
include/jprotocolpp.h	860
include/jrand.h	860
include/jreplay.h	861
include/jrsa.h	861
include/jsecass.h	862
include/jsnow3g.h	862
include/jsrtp.h	862
include/jsrtpsa.h	863

include/jstream.h	863
include/jtcp.h	863
include/jtcpsa.h	863
include/jtls.h	864
include/jtsa.h	864
include/judp.h	864
include/judpsa.h	865
include/jwifi.h	865
include/jwifisa.h	865
include/jwimax.h	866
include/jwimaxsa.h	866
include/jzuc.h	866
include/mtrand.h	867
include/poly1305-16.h	867
include/poly1305-32.h	868
include/poly1305-64.h	868
include/poly1305-8.h	869
include/SFMT-alti.h	869
SIMD oriented Fast Mersenne Twister(SFMT) pseudorandom number generator	869
include/SFMT-common.h	
SIMD oriented Fast Mersenne Twister(SFMT) pseudorandom number generator with jump function. This file includes common functions used in random number generation and jump	870
include/SFMT-neon.h	
SIMD oriented Fast Mersenne Twister(SFMT) for ARM with 128b NEON	871
include/SFMT-params.h	871
include/SFMT-params11213.h	872
include/SFMT-params1279.h	874
include/SFMT-params132049.h	875
include/SFMT-params19937.h	877
include/SFMT-params216091.h	878
include/SFMT-params2281.h	880
include/SFMT-params4253.h	881
include/SFMT-params44497.h	883
include/SFMT-params607.h	884
include/SFMT-params86243.h	886
include/SFMT-sse2-msc.h	
SIMD oriented Fast Mersenne Twister(SFMT) for Intel SSE2 for MSC	887
include/SFMT-sse2.h	
SIMD oriented Fast Mersenne Twister(SFMT) for Intel SSE2	888
include/SFMT.h	889
include/tinyxml2.h	890
include/wasp.h	892
jikey2/include/jikencrypt.h	893
jikey2/include/jikeparse.h	893
jikey2/include/jikeprf.h	894
jikey2/include/jikev2.h	894
jikey2/include/jikev2dh.h	896
jikey2/include/jikev2sa.h	896
jlogger/include/jlogger.h	897
jresponder/include/jexec.h	897
jresponder/include/jresponder.h	898
jresponder/include/jsecresponder.h	898
jtestbench/include/interfacepp.h	898
jtestbench/include/jmmu.h	899
jtestbench/include/jproducer.h	899
jtestbench/include/jring.h	899
jtestbench/include/jsecproducer.h	900
jtestbench/include/jsectestbench.h	900

jtestbench/include/ jtestbench.h	901
jtestbench/include/ optionparser.h	
This is the only file required to use The Lean Mean C++ Option Parser. Just #include it and you're set	901
platforms/SEC/include/ jsec.h	905
platforms/SEC/include/ sgt.h	905
schema/ lkey2.xsd	907
schema/ protocolpp.xsd	907

Chapter 6

Namespace Documentation

6.1 DriverPP Namespace Reference

Classes

- class [jdirectdrive](#)
- class [jdrive](#)
- class [jdriver](#)
- class [jringdrive](#)

Variables

- class [DriverPP::jdirectdrive_jsgt](#)

6.1.1 Variable Documentation

6.1.1.1 class DriverPP::jdirectdrive DriverPP::jsgt

6.2 InterfacePP Namespace Reference

Classes

- class [jmmu](#)
- class [jproducer](#)
- class [ringin](#)
- class [ringout](#)
- class [ringflow](#)
- class [secin](#)
- class [secout](#)
- class [jring](#)
- class [jsecproducer](#)
- class [jsectestbench](#)
- class [jtestbench](#)
- class [jexec](#)
- class [jresponder](#)
- class [jsecresponder](#)
- class [log_policy_interface](#)
- class [file_log_policy](#)

- class `file_stdout_log_policy`
- class `file_quiet_log_policy`
- class `jlogger`

6.3 option Namespace Reference

The namespace of The Lean Mean C++ Option Parser.

Classes

- struct `Descriptor`
Describes an option, its help text (usage) and how it should be parsed.
- class `Option`
A parsed option from the command line together with its argument if it has one.
- struct `Arg`
Functions for checking the validity of option arguments.
- struct `Stats`
Determines the minimum lengths of the buffer and options arrays used for Parser.
- class `Parser`
Checks argument vectors for validity and parses them into data structures that are easier to work with.
- struct `PrintUsageImplementation`

TypeDefs

- typedef `ArgStatus(* CheckArg)(const Option &option, bool msg)`
Signature of functions that check if an argument is valid for a certain type of option.

Enumerations

- enum `ArgStatus` {
 `ARG_NONE`, `ARG_OK`, `ARG_REQUIRED`, `ARG_IGNORE`,
`ARG_ILLEGAL` }
- Possible results when checking if an argument is valid for a certain option.*

Functions

- template<typename OStream >
`void printUsage` (OStream &prn, const `Descriptor` usage[], int width=80, int last_column_min_percent=50, int last_column_own_line_max_percent=75)
Outputs a nicely formatted usage string with support for multi-column formatting and line-wrapping.
- template<typename Function >
`void printUsage` (Function *prn, const `Descriptor` usage[], int width=80, int last_column_min_percent=50, int last_column_own_line_max_percent=75)
- template<typename Temporary >
`void printUsage` (const `Temporary` &prn, const `Descriptor` usage[], int width=80, int last_column_min_percent=50, int last_column_own_line_max_percent=75)
- template<typename Syscall >
`void printUsage` (Syscall *prn, int fd, const `Descriptor` usage[], int width=80, int last_column_min_percent=50, int last_column_own_line_max_percent=75)
- template<typename Function , typename Stream >
`void printUsage` (Function *prn, Stream *stream, const `Descriptor` usage[], int width=80, int last_column_min_percent=50, int last_column_own_line_max_percent=75)

6.3.1 Detailed Description

The namespace of The Lean Mean C++ [Option Parser](#).

6.3.2 Typedef Documentation

6.3.2.1 `typedef ArgStatus(* option::CheckArg)(const Option &option, bool msg)`

Signature of functions that check if an argument is valid for a certain type of option.

Every [Option](#) has such a function assigned in its [Descriptor](#).

```
* Descriptor usage[] = { {UNKNOWN, 0, "", "", Arg::None, ""}, ... };
```

A CheckArg function has the following signature:

```
ArgStatus CheckArg(const Option& option, bool msg);
```

It is used to check if a potential argument would be acceptable for the option. It will even be called if there is no argument. In that case `option.arg` will be `NULL`.

If `msg` is `true` and the function determines that an argument is not acceptable and that this is a fatal error, it should output a message to the user before returning [ARG_ILLEGAL](#). If `msg` is `false` the function should remain silent (or you will get duplicate messages).

See [ArgStatus](#) for the meaning of the return values.

While you can provide your own functions, often the following pre-defined checks (which never return [ARG_ILLEGAL](#)) will suffice:

- [Arg::None](#) For options that don't take an argument: Returns ARG_NONE.
- [Arg::Optional](#) Returns ARG_OK if the argument is attached and ARG_IGNORE otherwise.

6.3.3 Enumeration Type Documentation

6.3.3.1 `enum option::ArgStatus`

Possible results when checking if an argument is valid for a certain option.

In the case that no argument is provided for an option that takes an optional argument, return codes ARG_OK and ARG_IGNORE are equivalent.

Enumerator

- ARG_NONE** The option does not take an argument.
- ARG_OK** The argument is acceptable for the option.
- ARG_REQUIRED** The argument is acceptable for the option.
- ARG_IGNORE** The argument is not acceptable but that's non-fatal because the option's argument is optional.
- ARG_ILLEGAL** The argument is not acceptable and that's fatal.

6.3.4 Function Documentation

6.3.4.1 `template<typename OStream> void option::printUsage (OStream & prn, const Descriptor usage[], int width = 80, int last_column_min_percent = 50, int last_column_own_line_max_percent = 75)`

Outputs a nicely formatted usage string with support for multi-column formatting and line-wrapping.

`printUsage()` takes the help texts of a `Descriptor[]` array and formats them into a usage message, wrapping lines to achieve the desired output width.

Table formatting:

Aside from plain strings which are simply line-wrapped, the usage may contain tables. Tables are used to align elements in the output.

```
* // Without a table. The explanatory texts are not aligned.
* -c, --create  |Creates something.
* -k, --kill    |Destroys something.
*
* // With table formatting. The explanatory texts are aligned.
* -c, --create  |Creates something.
* -k, --kill    |Destroys something.
*
```

Table formatting removes the need to pad help texts manually with spaces to achieve alignment. To create a table, simply insert `\t` (tab) characters to separate the cells within a row.

```
* const option::Descriptor usage[] = {
* {..., "-c, --create \tCreates something." },
* {..., "-k, --kill \tDestroys something." }, ...
*
```

Note that you must include the minimum amount of space desired between cells yourself. Table formatting will insert further spaces as needed to achieve alignment.

You can insert line breaks within cells by using `\v` (vertical tab).

```
* const option::Descriptor usage[] = {
* {..., "-c,\v--create \tCreates\vsomething." },
* {..., "-k,\v--kill \tDestroys\vsomething." }, ...
*
* // results in
*
* -c,      Creates
* --create something.
* -k,      Destroys
* --kill   something.
*
```

You can mix lines that do not use `\t` or `\v` with those that do. The plain lines will not mess up the table layout. Alignment of the table columns will be maintained even across these interjections.

```
* const option::Descriptor usage[] = {
* {..., "-c, --create \tCreates something." },
* {..., "-----" },
* {..., "-k, --kill \tDestroys something." }, ...
*
* // results in
*
* -c, --create Creates something.
* -----
* -k, --kill    Destroys something.
*
```

You can have multiple tables within the same usage whose columns are aligned independently. Simply insert a dummy `Descriptor` with `help==0`.

```
* const option::Descriptor usage[] = {
* {..., "Long options:" },
* {..., "--very-long-option \tDoes something long." },
* {..., "--ultra-super-mega-long-option \tTakes forever to complete." },
* {..., 0 }, // ----- table break -----
* {..., "Short options:" },
* {..., "-s \tshort." },
* {..., "-q \tQuick." }, ...
*
* // results in
*
* Long options:
* --very-long-option          Does something long.
* --ultra-super-mega-long-option Takes forever to complete.
```

```

* Short options:
* -s  Short.
* -q  Quick.
*
* // Without the table break it would be
*
* Long options:
* --very-long-option      Does something long.
* --ultra-super-mega-long-option  Takes forever to complete.
* Short options:
* -s                      Short.
* -q                      Quick.
*

```

Output methods:

Because TheLeanMeanC++Option parser is freestanding, you have to provide the means for output in the first argument(s) to `printUsage()`. Because `printUsage()` is implemented as a set of template functions, you have great flexibility in your choice of output method. The following example demonstrates typical uses. Anything that's similar enough will work.

```

* #include <unistd.h>  // write()
* #include <iostream>   // cout
* #include <iostream>   // ostream
* #include <sstream>    // ostringstream
* #include <cstdio>     // fwrite()
* using namespace std;
*
* void my_write(const char* str, int size) {
*   fwrite(str, size, 1, stdout);
* }
*
* struct MyWriter {
*   void write(const char* buf, size_t size) const {
*     fwrite(str, size, 1, stdout);
*   }
* };
*
* struct MyWriteFunctor {
*   void operator()(const char* buf, size_t size) {
*     fwrite(str, size, 1, stdout);
*   }
* };
*
* ...
* printUsage(my_write, usage);      // custom write function
* printUsage(MyWriter(), usage);   // temporary of a custom class
* MyWriter writer;
* printUsage(writer, usage);       // custom class object
* MyWriteFunctor wfunctor;
* printUsage(&wfunctor, usage);    // custom functor
* printUsage(write, 1, usage);     // write() to file descriptor 1
* printUsage(cout, usage);        // an ostream&
* printUsage(fwrite, stdout, usage); // fwrite() to stdout
* ostringstream sstr;
* printUsage(sstr, usage);        // an ostringstream&
*
*
```

Notes:

- the `write()` method of a class that is to be passed as a temporary as `MyWriter()` is in the example, must be a `const` method, because temporary objects are passed as `const` reference. This only applies to temporary objects that are created and destroyed in the same statement. If you create an object like `writer` in the example, this restriction does not apply.
- a functor like `MyWriteFunctor` in the example must be passed as a pointer. This differs from the way functors are passed to e.g. the STL algorithms.
- All `printUsage()` templates are tiny wrappers around a shared non-template implementation. So there's no penalty for using different versions in the same program.
- `printUsage()` always interprets `Descriptor::help` as UTF-8 and always produces UTF-8-encoded output. If your system uses a different charset, you must do your own conversion. You may also need to change the font of the console to see non-ASCII characters properly. This is particularly true for Windows.
- **Security warning:** Do not insert untrusted strings (such as user-supplied arguments) into the usage. `printUsage()` has no protection against malicious UTF-8 sequences.

Parameters

<i>prn</i>	The output method to use. See the examples above.
<i>usage</i>	the Descriptor[] array whose <code>help</code> texts will be formatted.
<i>width</i>	the maximum number of characters per output line. Note that this number is in actual characters, not bytes. printUsage() supports UTF-8 in <code>help</code> and will count multi-byte UTF-8 sequences properly. Asian wide characters are counted as 2 characters.
<i>last_column_min_percent</i>	(0-100) The minimum percentage of <i>width</i> that should be available for the last column (which typically contains the textual explanation of an option). If less space is available, the last column will be printed on its own line, indented according to <i>last_column_own_line_max_percent</i> .
<i>last_column_own_line_max_percent</i>	(0-100) If the last column is printed on its own line due to less than <i>last_column_min_percent</i> of the width being available, then only <i>last_column_own_line_max_percent</i> of the extra line(s) will be used for the last column's text. This ensures an indentation. See example below.

```
* // width=20, last_column_min_percent=50 (i.e. last col. min. width=10)
* --3456789 1234567890
*           1234567890
*
* // width=20, last_column_min_percent=75 (i.e. last col. min. width=15)
* // last_column_own_line_max_percent=75
* --3456789
*           123456789012345
*           67890
*
* // width=20, last_column_min_percent=75 (i.e. last col. min. width=15)
* // last_column_own_line_max_percent=33 (i.e. max. 5)
* --3456789
*           12345
*           67890
*           12345
*           67890
*
```

6.3.4.2 template<typename Function > void option::printUsage (Function * *prn*, const Descriptor *usage*[], int *width* = 80, int *last_column_min_percent* = 50, int *last_column_own_line_max_percent* = 75)

6.3.4.3 template<typename Temporary > void option::printUsage (const Temporary & *prn*, const Descriptor *usage*[], int *width* = 80, int *last_column_min_percent* = 50, int *last_column_own_line_max_percent* = 75)

6.3.4.4 template<typename Syscall > void option::printUsage (Syscall * *prn*, int *fd*, const Descriptor *usage*[], int *width* = 80, int *last_column_min_percent* = 50, int *last_column_own_line_max_percent* = 75)

6.3.4.5 template<typename Function , typename Stream > void option::printUsage (Function * *prn*, Stream * *stream*, const Descriptor *usage*[], int *width* = 80, int *last_column_min_percent* = 50, int *last_column_own_line_max_percent* = 75)

6.4 PlatformPP Namespace Reference

Classes

- class [jsec](#)
- class [jsqt](#)

Functions

- class [PlatformPP::jsqt](#) [jsqt \(sgt_t *sgt\)](#)

6.4.1 Function Documentation

6.4.1.1 class PlatformPP::jsgt PlatformPP::jsgt(*sgt_t* * *sgt*)

6.5 ProtocolPP Namespace Reference

Classes

- class [aead_chacha_poly1305](#)
- class [chacha20](#)
- class [ciphers](#)
- class [EnumStringBase](#)
- struct [EnumString](#)
- class [jarray](#)
- class [jdata](#)
- class [jdsa](#)
- class [jecdsa](#)
- class [jicmp](#)
- class [jicmpsa](#)
- class [jip](#)
- class [jipsa](#)
- class [jipsec](#)
- class [jipsecsa](#)
- class [jlte](#)
- class [jltesa](#)
- class [jmacsec](#)
- class [jmacsecsa](#)
- class [jmodes](#)
- class [jpacket](#)
- class [jpoly1305](#)
- class [jprotocol](#)
- class [jprotocolpp](#)
- class [jrand](#)
- class [jreplay](#)
- class [jrsa](#)
- class [jsecass](#)
- class [jsnow3g](#)
- class [jsrtp](#)
- class [jsrtpsa](#)
- class [jstream](#)
- class [jtcp](#)
- class [jtcpsa](#)
- class [jtls](#)
- class [jtsa](#)
- class [judp](#)
- class [judpsa](#)
- class [jwifi](#)
- class [jwifisa](#)
- class [jwimax](#)
- class [jwimaxsa](#)
- class [jzuc](#)
- class [MTRand_int32](#)

Mersenne Twister random number generator.

- class [MTRand](#)
- class [MTRand_closed](#)
- class [MTRand_open](#)

- class [MTRand53](#)
- struct [jpoly1305_state_internal_t](#)
- union [W128_T](#)
- class [sfmt](#)
- class [wasp](#)
- class [jilkenrypt](#)
- class [jikeparse](#)
- class [jikeprf](#)
- class [jikev2](#)
- class [jikev2dh](#)
- class [jikev2sa](#)

TypeDefs

- typedef struct [ProtocolPP::jpoly1305_state_internal_t](#) `jpoly1305_state_internal_t`
- typedef union [W128_T](#) `w128_t`

Enumerations

- enum `field_t` {
 DIRECTION, VERSION, MODE, SPI,
 SEQNUM, EXTSEQNUM, ARLEN, ARWIN,
 CIPHER, CKEYLEN, CIPHERKEY, AUTH,
 AKEYLEN, AUTHKEY, IVLEN, IV,
 SALTLEN, SALT, BYTECNT, LIFETIME,
 SEQNUMOVRFLW, STATEFULFRAG, BYPASSDF, BYPASSDSCP,
 NAT, NCHK, DSECN, TTLHOP,
 FLAGS, NATSRC, NATDST, ID,
 LABEL, FRAGOFFSET, MOREFRAG, FRAGID,
 MTU, SOURCE, DESTINATION, EXTHDR,
 NH, ICVLEN, HDRLEN, TFCLEN,
 USEXT, RANDIV, NODEJUMBO, AUDIT,
 AUDITLOG, CHECKSUM, LENGTH, TYPE,
 CODE, POINTER, IDENTIFIER, GATEWAY,
 ORIGTIMESTAMP, RXTIMESTAMP, TXTIMESTAMP, MESSAGE,
 ICMPCODE, DATACTRL, PDUTYPE, POLLBIT,
 EXTENSION, RSN, SNLEN, HDREXT,
 LENGTHIND, HFNI, SUFI, FMS,
 BITMAPLEN, BITMAP, PGKINDEX, PTKINDENT,
 SDUTYPE, KDID, NMP, HRW,
 BEARER, FRESH, ETHERTYPE, SL,
 TCIAN, PN, XPN, SCI,
 SSC, SAKEY, ENRECEIVE, ENTRANSMIT,
 PROTECTFRAMES, INUSE, CREATETIME, STARTTIME,
 STOPTIME, PADDING, CC, MARKER,
 PT, ROC, TIMESTAMP, SSRC,
 CSRC, BLKSIZE, MKI, MKILEN,
 MKIDATA, ACKNUM, OFFSET, RCVWINDOW,
 URGENT, STATE, OPTIONS, SEGLEN,
 PRECEDENCE, SNDUNA, SNDNXT, SNDWND,
 SNDUP, SNDW1, SNDW2, ISS,
 RCVNXT, RCVWND, RCVUP, IRS,
 TCPTIMEOUT, EPOCH, CIPHERSUITE, ENCTHENMAC,
 IVE, FRAMECTL, CTLEXT, PROTVER,
 SUBTYPE, TDS, FDS, MFRAG,
 RETRY, PWRMGMT, MDATA, WEP,
 ORDER, KDFALG, ADDR1, ADDR2,
 ADDR3, SEQCTL, ADDR4, QOSCTL,
 HTCTL, EXTIV, KEYID, SPPCAP,
 FCS, HT, EC, ESF,
 CI, EKS, CID, FID,
 EH, HCS, IKECNXT, EXCHG,
 NXTPYLD, MAJOR_VERSION, MINOR_VERSION, MSGIDINIT,
 MSGIDRESP, SPIi, SPIr, Ni,
 Nr, SKd, SKei, SKer,
 SKsi, SKsr, INTEG, SKai,
 SKar, PRF, PRFLEN, SKpi,
 SKpr, DH, IKEAUTH, BITSIZE,
 PRVKEY, PUBKEY, PRIME, SUBPRIME,
 GENERATOR, GX, GY, CURVE
 }
- enum `Endian_t` { BIG, LITTLE }
- enum `platform_t` { WASPPLAT, SECPLAT }
- enum `pad_t` {
 RANDOM, INCREMENT, ZERO, SIZE,
 UNKNWN
 }

- enum `protocol_t`{
NO_ERROR, PROTOCOL, MACSEC, WIFI,
WIGIG, WIMAX, LTE, RLC,
TLS, SRTP, UDPP, TCPP,
ICMPP, IP, IPSEC, XMLPARSER,
WASP, RINGDRIVER, DIRECTDRIVER, DRIVER,
LOOPBACK, ETHERNET, IKEV2, NOPROTO }
- enum `direction_t`{
ENCAP, DECAP, ENC, DEC,
DOWNLINK, UPLINK, NODIR }
- enum `cipher_t`{
NULL_CIPHER, DES_CBC, DES40_CBC, TDES_CBC,
AES_CBC, AES_CTR, AES_CCM, AES_GCM,
CHACHA20, CHACHA20_POLY1305, CAMELLIA_CBC, CAMELLIA_CTR,
CAMELLIA_CCM, CAMELLIA_GCM, ARIA_CBC, ARIA_CTR,
ARIA_GCM, SEED_CBC, SEED_CTR, SEED_GCM,
SM4_CBC, SM4_CTR, SM4_GCM, SM4_CCM,
ARC4, SNOWE, ZUCE, NONE_ENC }

encryption ciphers

- enum `auth_t`{
NULL_AUTH, MD5, SHA1, SHA224,
SHA256, SHA384, SHA512, SHA3_224,
SHA3_256, SHA3_384, SHA3_512, AES_XCBC_MAC,
POLY1305, AES_CMAC, AES_GMAC, CRC32,
SM3, SNOWA, ZUCA, NONE_AUTH }

authentication types

- enum `iana_t`{

```

HOPOPT, ICMP, IGMP, GGP,
IPV4, ST, TCP, CBT,
EGP, IGP, BBN_RCC_MON, NVP_II,
PUP, ARGUS, EMCON, XNET,
CHAOS, UDP, MUX, DCN_MEAS,
HMP, PRM, XNS_IDP, TRUNK_1,
TRUNK_2, LEAF_1, LEAF_2, RDP,
IRTP, ISO_TP4, NETBLT, MFE_NSP,
MERIT_INP, DCCP, THREE_PC, IDPR,
XTP, DDP, IDPR_CMTP, TP_PLUS,
IL, IPV6, SDRP, IPV6_ROUTE,
IPV6_FRAG, IDRP, RSVP, GRE,
DSR, BNA, ESP, AH,
I_NLSP, SWIPE, NARP, MOBILE,
TLSP, SKIP, ICMPV6, IPV6_NONXT,
IPV6_OPTS, HOST_INT_PROT, CFTP, LOCAL_NET,
SAT_EXPAK, KRYPTOLAN, RVD, IPPC,
DFS, SAT_MON, VISA, IPCV,
CPNX, CPHB, WSN, PVP,
BR_SAT_MON, SUN_ND, WB_MON, WB_EXPAK,
ISO_IP, VMTP, SECURE_VMTP, VINES,
TTP, IPTM, NSFNET_IGP, DGP,
TCF, EIGRP, OSPFIGP, SPRITE_RPC,
LARP, MTP, AX_25, MICP,
SCC_SP, ETHERIP, IENCAP, PRIV_ENCRYPT,
GMTP, IFMP, PNNI, PIM,
ARIS, SCPS, QNX, A_N,
IPCOMP, SNP, COMPAQ_PEER, IPX_IN_IP,
VRRP, PGM, ZERO_HOP_PROT, L2TP,
DDX, IATP, STP, SRP,
UTI, SMP, SM, PTP,
ISIS_OVER_IPV4, FIRE, CRTP, CRUDP,
SSCOPMCE, IPLT, SPS, PIPE,
SCTP, FC, RSVP_E2E_IGNORE, MOBILITY_HEADER,
UDPLITE, MPLS_IN_IP, MANET, HIP,
SHIM6, WESP, ROHC, JUMBOGRAM =194 }

```

- enum `err_t` {
 `ERR_NONE, ERR_LATE, ERR_REPLAY, ERR_ROLLOVER,`
`ERR_ROLLUNDER, ERR_PROGRAM, ERR_ICV, ERR_CRC,`
`ERR_READ_ENDOFILE, ERR_READ_FAILFILE, ERR_READ_BADFILE, ERR_WRITE_FAILFILE,`
`ERR_WRITE_BADFILE, ERR_CHECKSUM, ERR_LISTEN, ERR_ACKNUM,`
`ERR_CLOSED, ERR_BITS, ERR_TTL, ERR_JUMBOGRAM_FORMAT,`
`WARN_DUMMY, WARN_IPV6_ROUTE, WARN_ZERO_DATA, ERR_ICMP_HDR_EXT_LEN,`
`ERR_ICMP_SEGMENTS_LEFT, ERR_MULTICAST_EXT_HDR, ERR_UNKNOWN_ROUTE_TYPE, ERR_CIPHER_KEY_SIZE,`
`ERR_AUTH_KEY_SIZE, ERR_IV_SIZE, ERR_SALT_SIZE, ERR_ICV_SIZE,`
`ERR_UNKNOWN_NXTHDR, ERR_FORMAT_ERROR, WARN_INPUT_QUEUE_FULL, WARN_OUTPUT_QUEUE_EMPTY,`
`ERR_SA_NOT_FOUND, ERR_KEY_NOKEY, ERR_KEY_EXPIRED, ERR_KEY_REVOKED,`
`ERR_KEY_ACCESS, ERR_KEY_NOT_SUPP, ERR_KEY_QUOTA, ERR_KEY_INVALID,`
`ERR_KEY_REJECTED, ERR_KEY_NOMEM, ERR_KEY_INTR, ERR_UNKNOWN }`
- enum `replay_t` {
 `NORMAL, SHIFT, WINDOW, REPLAY,`
`LATE, ROLLOVER, ROLLUNDER }`
- enum `ipmode_t` { `TRANSPORT, TUNNEL` }
- enum `icmppmsg_t` {

```

ECHORPLYMSG, RSVD1, RSVD2, DESTUNRCH,
SRCQNCH, RDIRMSG, ALTHOST, RSVD,
ECHORQST, RTRADVERT, RTRSOLCIT, TIMEXCEED,
BADIPHDR, TIMESTMP, TIMERPLY, INFORQST,
INFORPLY, ADMSKRQST, ADMSKRLY, TRACERTE,
NODEST, PKTBIG, TIMEOUT, PARAM,
PRVTE1, PRVTE2, RSVDE, ECHORQSTV6,
ECHORPLYV6, PRVTI1, PRVTI2, RSVDI }

• enum icmpcode_t {
    ECHORPLY, NONETWRK, NOHOST, NOPROT,
    NOPORT, FRAGRQD, RTEFAIL, DSTUNKNWN,
    DSTHSTUNKNWN, SRCHSTISOLT, NETWKPROHIB, HOSTPROHIB,
    NETWKNOTOS, HOSTNOTOS, COMMPROHIB, HOSTVILATE,
    CUTOFF, REDIRNETWK, REDIRHOST, REDIRTOSN,
    REDIRTOSH, TTL_EXPIRE, FRAG_EXPIRE, PTRINDERR,
    OPTERR, BADLENGTH, NOROUTE, COMMPROHIBV6,
    BYNDSRCADDR, ADDRUNREACH, PORTUNREACH, SRCADDRPLCY,
    REJECTDST, HOPLMTEXCD, FRAGTIME, ERRHDRFIELD,
    NXTHDRERR, IPV6OPTERR, SEQNUMRST }

• enum tlsver_t {
    SSL30 = 0x0300, TLS10 = 0x0301, TLS11 = 0x0302, TLS12 = 0x0303,
    DTLS = 0xFFFF }

• enum tlstype_t {
    CHG_CIPHER_SPEC = 20, ALERT = 21, HANDSHAKE = 22, APPLICATION = 23,
    HEARTBEAT = 24 }

• enum tlslvl_t { TLSLVL0, WARNING, FATAL }

• enum tls_handshake_t {
    HELLO_REQUEST =0, CLIENT_HELLO =1, SERVER_HELLO =2, NEW_SESSION_TICKET =4,
    CERTIFICATE =11, SERVER_KEY_EXCHANGE =12, CERTIFICATE_REQUEST =13, SERVER_HELLO_DONE =14,
    CERTIFICATE_VERIFY =15, CLIENT_KEY_EXCHANGE =16, FINISHED =20 }

• enum tls_error_t {
    CLOSE = 0, MESSAGE_FATAL =10, BAD_RECORD_MAC =20, DECRYPT_FATAL =21,
    RECORD_OVERFLOW =22, DECOMPRESS_FAIL =30, HANDSHAKE_FAIL =40, NO_CERTIFICATE =41,
    BAD_CERTIFICATE =42, UNSUPPORTED_CERTIFICATE =43, CERTIFICATE_REVOKED =44, CERTIFICATE_EXPIRED =45,
    CERTIFICATE_UNKNOWN =46, ILLEGAL_PARAMETER =47, UNKNOWN_CA =48, ACCESS_DENIED =49,
    DECODE_ERROR =50, DECRYPT_ERROR =51, EXPORT_RESTRICTION =60, PROTOCOL_VERSION =70,
    INSUFFICIENT_SECURITY =71, INTERNAL_ERROR =80, USER_CANCELED =90, NO_RENEGOTIATION =100,
    UNSUPPORTED_EXTENSION =110, CERTIFICATE_UNOBTAINABLE =111, UNRECOGNIZED_NAME =112, BAD_CERTIFICATE_STATUS_RESPONSE =113,
    BAD_CERTIFICATE_HASH_VALUE =114, UNKNOWN_PSK_IDENTITY =115, NO_APPLICATION_PROTOCOL =120 }

• enum srtpcipher_t {
    NULL_SRTP, AES_CTR_SHA1, AEAD_AES_128_GCM, AEAD_AES_256_GCM,
    AEAD_AES_128_GCM_8, AEAD_AES_256_GCM_8, AEAD_AES_128_GCM_12, AEAD_AES_256_GCM_12,
    AEAD_AES_128_CCM, AEAD_AES_256_CCM, AEAD_AES_128_CCM_8, AEAD_AES_256_CCM_8,
    AEAD_AES_128_CCM_12, AEAD_AES_256_CCM_12, ARIA_128_CTR_HMAC_SHA1_80, ARIA_128_CTR_HMAC_SHA1_32,
    ARIA_192_CTR_HMAC_SHA1_80, ARIA_192_CTR_HMAC_SHA1_32, ARIA_256_CTR_HMAC_SHA1_80,
    ARIA_256_CTR_HMAC_SHA1_32,
    AEAD_ARIA_128_GCM, AEAD_ARIA_256_GCM }

• enum tls_ciphersuite_t {
    TLS_NULL_WITH_NULL_NULL =0x0000, TLS_RSA_WITH_NULL_MD5 =0x0001, TLS_RSA_WITH_NUL-

```

```

L_SHA =0x0002, TLS_RSA_EXPORT_WITH_RC4_40_MD5 =0x0003,
TLS_RSA_WITH_RC4_128_MD5 =0x0004, TLS_RSA_WITH_RC4_128_SHA =0x0005, TLS_RSA_EXPORT-
RT_WITH_DES40_CBC_SHA =0x0008, TLS_RSA_WITH_DES_CBC_SHA =0x0009,
TLS_RSA_WITH_3DES_EDE_CBC_SHA =0x000A, TLS_DH_DSS_EXPORT_WITH_DES40_CBC_SHA
=0x000B, TLS_DH_DSS_WITH_DES_CBC_SHA =0x000C, TLS_DH_DSS_WITH_3DES_EDE_CBC_SHA
=0x000D,
TLS_DH_RSA_EXPORT_WITH_DES40_CBC_SHA =0x000E, TLS_DH_RSA_WITH_DES_CBC_SHA
=0x000F, TLS_DH_RSA_WITH_3DES_EDE_CBC_SHA =0x0010, TLS_DHE_DSS_EXPORT_WITH_D-
ES40_CBC_SHA =0x0011,
TLS_DHE_DSS_WITH_DES_CBC_SHA =0x0012, TLS_DHE_DSS_WITH_3DES_EDE_CBC_SHA
=0x0013, TLS_DHE_RSA_EXPORT_WITH_DES40_CBC_SHA =0x0014, TLS_DHE_RSA_WITH_DES-
_CBC_SHA =0x0015,
TLS_DHE_RSA_WITH_3DES_EDE_CBC_SHA =0x0016, TLS_DH_ANON_EXPORT_WITH_RC4_40_M-
D5 =0x0017, TLS_DH_ANON_WITH_RC4_128_MD5 =0x0018, TLS_DH_ANON_EXPORT_WITH_DES40-
_CBC_SHA =0x0019,
TLS_DH_ANON_WITH_DES_CBC_SHA =0x001A, TLS_DH_ANON_WITH_3DES_EDE_CBC_SHA
=0x001B, TLS_KRB5_WITH_DES_CBC_SHA =0x001E, TLS_KRB5_WITH_3DES_EDE_CBC_SHA
=0x001F,
TLS_KRB5_WITH_RC4_128_SHA =0x0020, TLS_KRB5_WITH_DES_CBC_MD5 =0x0022, TLS_KRB5_W-
ITH_3DES_EDE_CBC_MD5 =0x0023, TLS_KRB5_WITH_RC4_128_MD5 =0x0024,
TLS_KRB5_EXPORT_WITH_DES_CBC_40_SHA =0x0026, TLS_KRB5_EXPORT_WITH_RC4_40_SHA
=0x0028, TLS_KRB5_EXPORT_WITH_DES_CBC_40_MD5 =0x0029, TLS_KRB5_EXPORT_WITH_RC4_-
40_MD5 =0x002B,
TLS_PSK_WITH_NULL_SHA =0x002C, TLS_DHE_PSK_WITH_NULL_SHA =0x002D, TLS_RSA_PSK_W-
ITH_NULL_SHA =0x002E, TLS_RSA_WITH_AES_128_CBC_SHA =0x002F,
TLS_DH_DSS_WITH_AES_128_CBC_SHA =0x0030, TLS_DH_RSA_WITH_AES_128_CBC_SHA
=0x0031, TLS_DHE_DSS_WITH_AES_128_CBC_SHA =0x0032, TLS_DHE_RSA_WITH_AES_128_C-
BC_SHA =0x0033,
TLS_DH_ANON_WITH_AES_128_CBC_SHA =0x0034, TLS_RSA_WITH_AES_256_CBC_SHA =0x0035,
TLS_DH_DSS_WITH_AES_256_CBC_SHA =0x0036, TLS_DH_RSA_WITH_AES_256_CBC_SHA
=0x0037,
TLS_DHE_DSS_WITH_AES_256_CBC_SHA =0x0038, TLS_DHE_RSA_WITH_AES_256_CBC_SH-
A =0x0039, TLS_DH_ANON_WITH_AES_256_CBC_SHA =0x003A, TLS_RSA_WITH_NULL_SHA256
=0x003B,
TLS_RSA_WITH_AES_128_CBC_SHA256 =0x003C, TLS_RSA_WITH_AES_256_CBC_SHA256 =0x003D,
TLS_DH_DSS_WITH_AES_128_CBC_SHA256 =0x003E, TLS_DH_RSA_WITH_AES_128_CBC_SHA256
=0x003F,
TLS_DHE_DSS_WITH_AES_128_CBC_SHA256 =0x0040, TLS_RSA_WITH_CAMELLIA_128_CBC_SHA
=0x0041, TLS_DH_DSS_WITH_CAMELLIA_128_CBC_SHA =0x0042, TLS_DH_RSA_WITH_CAMELLIA_-
128_CBC_SHA =0x0043,
TLS_DHE_DSS_WITH_CAMELLIA_128_CBC_SHA =0x0044, TLS_DHE_RSA_WITH_CAMELLIA_128_C-
BC_SHA =0x0045, TLS_DH_ANON_WITH_CAMELLIA_128_CBC_SHA =0x0046, TLS_DHE_RSA_WITH-
_AES_128_CBC_SHA256 =0x0067,
TLS_DH_DSS_WITH_AES_256_CBC_SHA256 =0x0068, TLS_DH_RSA_WITH_AES_256_CBC_SHA256
=0x0069, TLS_DHE_DSS_WITH_AES_256_CBC_SHA256 =0x006A, TLS_DHE_RSA_WITH_AES_256_C-
BC_SHA256 =0x006B,
TLS_DH_ANON_WITH_AES_128_CBC_SHA256 =0x006C, TLS_DH_ANON_WITH_AES_256_CBC_SH-
A256 =0x006D, TLS_RSA_WITH_CAMELLIA_256_CBC_SHA =0x0084, TLS_DH_DSS_WITH_CAMELLIA_-
256_CBC_SHA =0x0085,
TLS_DH_RSA_WITH_CAMELLIA_256_CBC_SHA =0x0086, TLS_DHE_DSS_WITH_CAMELLIA_256_CB-
C_SHA =0x0087, TLS_DHE_RSA_WITH_CAMELLIA_256_CBC_SHA =0x0088, TLS_DH_ANON_WITH_-
CAMELLIA_256_CBC_SHA =0x0089,
TLS_PSK_WITH_RC4_128_SHA =0x008A, TLS_PSK_WITH_3DES_EDE_CBC_SHA =0x008B, TLS_PSK-
_WITH_AES_128_CBC_SHA =0x008C, TLS_PSK_WITH_AES_256_CBC_SHA =0x008D,
TLS_DHE_PSK_WITH_RC4_128_SHA =0x008E, TLS_DHE_PSK_WITH_3DES_EDE_CBC_SHA =0x008-
F, TLS_DHE_PSK_WITH_AES_128_CBC_SHA =0x0090, TLS_DHE_PSK_WITH_AES_256_CBC_SHA
=0x0091,
TLS_RSA_PSK_WITH_RC4_128_SHA =0x0092, TLS_RSA_PSK_WITH_3DES_EDE_CBC_SHA =0x0093,
```

TLS_RSA_PSK_WITH_AES_128_CBC_SHA =0x0094, TLS_RSA_PSK_WITH_AES_256_CBC_SHA =0x0095,
 TLS_RSA_WITH_SEED_CBC_SHA =0x0096, TLS_DH_DSS_WITH_SEED_CBC_SHA =0x0097, TLS_DH_RSA_WITH_SEED_CBC_SHA =0x0098, TLS_DHE_DSS_WITH_SEED_CBC_SHA =0x0099,
 TLS_DHE_RSA_WITH_SEED_CBC_SHA =0x009A, TLS_DH_ANON_WITH_SEED_CBC_SHA =0x009B, TLS_RSA_WITH_AES_128_GCM_SHA256 =0x009C, TLS_RSA_WITH_AES_256_GCM_SHA384 =0x009D,
 TLS_DHE_RSA_WITH_AES_128_GCM_SHA256 =0x009E, TLS_DHE_RSA_WITH_AES_256_GCM_SHA384 =0x009F, TLS_DH_RSA_WITH_AES_128_GCM_SHA256 =0x00A0, TLS_DH_RSA_WITH_AES_256_GCM_SHA384 =0x00A1,
 TLS_DHE_DSS_WITH_AES_128_GCM_SHA256 =0x00A2, TLS_DHE_DSS_WITH_AES_256_GCM_SHA384 =0x00A3, TLS_DH_DSS_WITH_AES_128_GCM_SHA256 =0x00A4, TLS_DH_DSS_WITH_AES_256_GCM_SHA384 =0x00A5,
 TLS_DH_ANON_WITH_AES_128_GCM_SHA256 =0x00A6, TLS_PSK_WITH_AES_128_GCM_SHA256 =0x00A8, TLS_PSK_WITH_AES_256_GCM_SHA384 =0x00A9,
 TLS_DHE_PSK_WITH_AES_128_GCM_SHA256 =0x00AA, TLS_DHE_PSK_WITH_AES_256_GCM_SHA384 =0x00AB, TLS_RSA_PSK_WITH_AES_128_GCM_SHA256 =0x00AC, TLS_RSA_PSK_WITH_AES_256_GCM_SHA384 =0x00AD,
 TLS_PSK_WITH_AES_128_CBC_SHA256 =0x00AE, TLS_PSK_WITH_AES_256_CBC_SHA384 =0x00AF, TLS_PSK_WITH_NULL_SHA256 =0x00B0, TLS_PSK_WITH_NULL_SHA384 =0x00B1,
 TLS_DHE_PSK_WITH_AES_128_CBC_SHA256 =0x00B2, TLS_DHE_PSK_WITH_AES_256_CBC_SHA384 =0x00B3, TLS_DHE_PSK_WITH_NULL_SHA256 =0x00B4, TLS_DHE_PSK_WITH_NULL_SHA384 =0x00B5,
 TLS_RSA_PSK_WITH_AES_128_CBC_SHA256 =0x00B6, TLS_RSA_PSK_WITH_AES_256_CBC_SHA384 =0x00B7, TLS_RSA_PSK_WITH_NULL_SHA256 =0x00B8, TLS_RSA_PSK_WITH_NULL_SHA384 =0x00B9,
 TLS_RSA_WITH_CAMELLIA_128_CBC_SHA256 =0x00BA, TLS_DH_DSS_WITH_CAMELLIA_128_CBC_SHA256 =0x00BB, TLS_DH_RSA_WITH_CAMELLIA_128_CBC_SHA256 =0x00BC, TLS_DHE_DSS_WITH_CAMELLIA_128_CBC_SHA256 =0x00BD,
 TLS_DHE_RSA_WITH_CAMELLIA_128_CBC_SHA256 =0x00BE, TLS_DH_ANON_WITH_CAMELLIA_128_CBC_SHA256 =0x00BF, TLS_RSA_WITH_CAMELLIA_256_CBC_SHA256 =0x00C0, TLS_DH_DSS_WITH_CAMELLIA_256_CBC_SHA256 =0x00C1,
 TLS_DH_RSA_WITH_CAMELLIA_256_CBC_SHA256 =0x00C2, TLS_DHE_DSS_WITH_CAMELLIA_256_CBC_SHA256 =0x00C3, TLS_DHE_RSA_WITH_CAMELLIA_256_CBC_SHA256 =0x00C4, TLS_DH_ANON_WITH_CAMELLIA_256_CBC_SHA256 =0x00C5,
 TLS_ECDH_ECDSA_WITH_NULL_SHA =0xC001, TLS_ECDH_ECDSA_WITH_RC4_128_SHA =0xC002, TLS_ECDH_ECDSA_WITH_3DES_EDE_CBC_SHA =0xC003, TLS_ECDH_ECDSA_WITH_AES_128_CBC_SHA =0xC004,
 TLS_ECDH_ECDSA_WITH_AES_256_CBC_SHA =0xC005, TLS_ECDHE_ECDSA_WITH_NULL_SHA =0xC006, TLS_ECDHE_ECDSA_WITH_RC4_128_SHA =0xC007, TLS_ECDHE_ECDSA_WITH_3DES_EDE_CBC_SHA =0xC008,
 TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA =0xC009, TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA =0xC00A, TLS_ECDH_RSA_WITH_NULL_SHA =0xC00B, TLS_ECDH_RSA_WITH_RC4_128_SHA =0xC00C,
 TLS_ECDH_RSA_WITH_3DES_EDE_CBC_SHA =0xC00D, TLS_ECDH_RSA_WITH_AES_128_CBC_SHA =0xC00E, TLS_ECDH_RSA_WITH_AES_256_CBC_SHA =0xC00F, TLS_ECDHE_RSA_WITH_NULL_SHA =0xC010,
 TLS_ECDHE_RSA_WITH_RC4_128_SHA =0xC011, TLS_ECDHE_RSA_WITH_3DES_EDE_CBC_SHA =0xC012, TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA =0xC013, TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA =0xC014,
 TLS_ECDH_ANON_WITH_NULL_SHA =0xC015, TLS_ECDH_ANON_WITH_RC4_128_SHA =0xC016, TLS_ECDH_ANON_WITH_3DES_EDE_CBC_SHA =0xC017, TLS_ECDH_ANON_WITH_AES_128_CBC_SHA =0xC018,
 TLS_ECDH_ANON_WITH_AES_256_CBC_SHA =0xC019, TLS_SRPSHA_WITH_3DES_EDE_CBC_SHA =0xC01A, TLS_SRPSHA_RSA_WITH_3DES_EDE_CBC_SHA =0xC01B, TLS_SRPSHA_DSS_WITH_3DES_EDE_CBC_SHA =0xC01C,
 TLS_SRPSHA_WITH_AES_128_CBC_SHA =0xC01D, TLS_SRPSHA_RSA_WITH_AES_128_CBC_SHA =

`A` =0xC01E, `TLS_SR_P_SHA_DSS_WITH_AES_128_CBC_SHA` =0xC01F, `TLS_SR_P_SHA_WITH_AES_256_CBC_SHA` =0xC020,
`TLS_SR_P_SHA_RSA_WITH_AES_256_CBC_SHA` =0xC021, `TLS_SR_P_SHA_DSS_WITH_AES_256_CBC_SHA` =0xC022, `TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256` =0xC023, `TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384` =0xC024,
`TLS_ECDH_ECDSA_WITH_AES_128_CBC_SHA256` =0xC025, `TLS_ECDH_ECDSA_WITH_AES_256_CBC_SHA384` =0xC026, `TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA384` =0xC027, `TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384` =0xC028,
`TLS_ECDH_RSA_WITH_AES_128_CBC_SHA256` =0xC029, `TLS_ECDH_RSA_WITH_AES_256_CBC_SHA384` =0xC02A, `TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256` =0xC02B, `TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384` =0xC02C,
`TLS_ECDH_ECDSA_WITH_AES_128_GCM_SHA256` =0xC02D, `TLS_ECDH_ECDSA_WITH_AES_256_GCM_SHA384` =0xC02E, `TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256` =0xC02F, `TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384` =0xC030,
`TLS_ECDH_RSA_WITH_AES_128_GCM_SHA384` =0xC031, `TLS_ECDH_RSA_WITH_AES_256_GCM_SHA384` =0xC032, `TLS_ECDHE_PSK_WITH_RC4_128_SHA` =0xC033, `TLS_ECDHE_PSK_WITH_3DES_EDE_CBC_SHA` =0xC034,
`TLS_ECDHE_PSK_WITH_AES_128_CBC_SHA` =0xC035, `TLS_ECDHE_PSK_WITH_AES_256_CBC_SHA` =0xC036, `TLS_ECDHE_PSK_WITH_AES_128_CBC_SHA256` =0xC037, `TLS_ECDHE_PSK_WITH_AES_256_CBC_SHA384` =0xC038,
`TLS_ECDHE_PSK_WITH_NULL_SHA` =0xC039, `TLS_ECDHE_PSK_WITH_NULL_SHA256` =0xC03A, `TLS_ECDHE_PSK_WITH_NULL_SHA384` =0xC03B, `TLS_RSA_WITH_ARIA_128_CBC_SHA256` =0xC03C,
`TLS_RSA_WITH_ARIA_256_CBC_SHA384` =0xC03D, `TLS_DH_DSS_WITH_ARIA_128_CBC_SHA256` =0xC03E, `TLS_DH_DSS_WITH_ARIA_256_CBC_SHA384` =0xC03F, `TLS_DH_RSA_WITH_ARIA_128_CBC_SHA256` =0xC040,
`TLS_DH_RSA_WITH_ARIA_256_CBC_SHA384` =0xC041, `TLS_DHE_DSS_WITH_ARIA_128_CBC_SHA256` =0xC042, `TLS_DHE_DSS_WITH_ARIA_256_CBC_SHA384` =0xC043, `TLS_DHE_RSA_WITH_ARIA_128_CBC_SHA256` =0xC044,
`TLS_DHE_RSA_WITH_ARIA_256_CBC_SHA384` =0xC045, `TLS_DHE_ANON_WITH_ARIA_128_CBC_SHA256` =0xC046, `TLS_DHE_ANON_WITH_ARIA_256_CBC_SHA384` =0xC047, `TLS_ECDHE_ECDSA_WITH_ARIA_128_CBC_SHA256` =0xC048,
`TLS_ECDHE_ECDSA_WITH_ARIA_256_CBC_SHA384` =0xC049, `TLS_ECDH_ECDSA_WITH_ARIA_128_CBC_SHA256` =0xC04A, `TLS_ECDH_ECDSA_WITH_ARIA_256_CBC_SHA384` =0xC04B, `TLS_ECDHE_RSA_WITH_ARIA_128_CBC_SHA256` =0xC04C,
`TLS_ECDHE_RSA_WITH_ARIA_256_CBC_SHA384` =0xC04D, `TLS_ECDH_RSA_WITH_ARIA_128_CBC_SHA256` =0xC04E, `TLS_ECDH_RSA_WITH_ARIA_256_CBC_SHA384` =0xC04F, `TLS_RSA_WITH_ARIA_128_GCM_SHA256` =0xC050,
`TLS_RSA_WITH_ARIA_256_GCM_SHA384` =0xC051, `TLS_DHE_RSA_WITH_ARIA_128_GCM_SHA256` =0xC052, `TLS_DHE_RSA_WITH_ARIA_256_GCM_SHA384` =0xC053, `TLS_DH_RSA_WITH_ARIA_128_GCM_SHA256` =0xC054,
`TLS_DH_RSA_WITH_ARIA_256_GCM_SHA384` =0xC055, `TLS_DHE_DSS_WITH_ARIA_128_GCM_SHA256` =0xC056, `TLS_DHE_DSS_WITH_ARIA_256_GCM_SHA384` =0xC057, `TLS_DH_DSS_WITH_ARIA_128_GCM_SHA256` =0xC058,
`TLS_DH_DSS_WITH_ARIA_256_GCM_SHA384` =0xC059, `TLS_DHE_ANON_WITH_ARIA_128_GCM_SHA256` =0xC05A, `TLS_DHE_ANON_WITH_ARIA_256_GCM_SHA384` =0xC05B, `TLS_ECDHE_ECDSA_WITH_ARIA_128_GCM_SHA256` =0xC05C,
`TLS_ECDHE_ECDSA_WITH_ARIA_256_GCM_SHA384` =0xC05D, `TLS_ECDH_ECDSA_WITH_ARIA_128_GCM_SHA256` =0xC05E, `TLS_ECDH_ECDSA_WITH_ARIA_256_GCM_SHA384` =0xC05F, `TLS_ECDHE_RSA_WITH_ARIA_128_GCM_SHA256` =0xC060,
`TLS_ECDHE_RSA_WITH_ARIA_256_GCM_SHA384` =0xC061, `TLS_ECDH_RSA_WITH_ARIA_128_GCM_SHA256` =0xC062, `TLS_ECDH_RSA_WITH_ARIA_256_GCM_SHA384` =0xC063, `TLS_PSK_WITH_ARIA_128_CBC_SHA256` =0xC064,
`TLS_PSK_WITH_ARIA_256_CBC_SHA384` =0xC065, `TLS_DHE_PSK_WITH_ARIA_128_CBC_SHA256` =0xC066, `TLS_DHE_PSK_WITH_ARIA_256_CBC_SHA384` =0xC067, `TLS_RSA_PSK_WITH_ARIA_128_CBC_SHA256` =0xC068,
`TLS_RSA_PSK_WITH_ARIA_256_CBC_SHA384` =0xC069, `TLS_PSK_WITH_ARIA_128_GCM_SHA256` =0xC06A, `TLS_PSK_WITH_ARIA_256_GCM_SHA384` =0xC06B, `TLS_DHE_PSK_WITH_ARIA_128_GC-`

```

M_SHA256 =0xC06C,
TLS_DHE_PSK_WITH_ARIA_256_GCM_SHA384 =0xC06D, TLS_RSA_PSK_WITH_ARIA_128_GCM_SH-
A256 =0xC06E, TLS_RSA_PSK_WITH_ARIA_256_GCM_SHA384 =0xC06F, TLS_ECDHE_PSK_WITH_A-
RIA_128_CBC_SHA256 =0xC070,
TLS_ECDHE_PSK_WITH_ARIA_256_CBC_SHA384 =0xC071, TLS_ECDHE_ECDSA_WITH_CAMELLI-
A_128_CBC_SHA256 =0xC072, TLS_ECDHE_ECDSA_WITH_CAMELLIA_256_CBC_SHA384 =0xC073,
TLS_ECDH_ECDSA_WITH_CAMELLIA_128_CBC_SHA256 =0xC074,
TLS_ECDH_ECDSA_WITH_CAMELLIA_256_CBC_SHA384 =0xC075, TLS_ECDHE_RSA_WITH_CAME-
LLIA_128_CBC_SHA256 =0xC076, TLS_ECDHE_RSA_WITH_CAMELLIA_256_CBC_SHA384 =0xC077,
TLS_ECDH_RSA_WITH_CAMELLIA_128_CBC_SHA256 =0xC078,
TLS_ECDH_RSA_WITH_CAMELLIA_256_CBC_SHA384 =0xC079, TLS_RSA_WITH_CAMELLIA_128_G-
CM_SHA256 =0xC07A, TLS_RSA_WITH_CAMELLIA_256_GCM_SHA384 =0xC07B, TLS_DHE_RSA_WI-
TH_CAMELLIA_128_GCM_SHA256 =0xC07C,
TLS_DHE_RSA_WITH_CAMELLIA_256_GCM_SHA384 =0xC07D, TLS_DH_RSA_WITH_CAMELLIA_128-
_GCM_SHA256 =0xC07E, TLS_DH_RSA_WITH_CAMELLIA_256_GCM_SHA384 =0xC07F, TLS_DHE_D-
SS_WITH_CAMELLIA_128_GCM_SHA256 =0xC080,
TLS_DHE_DSS_WITH_CAMELLIA_256_GCM_SHA384 =0xC081, TLS_DH_DSS_WITH_CAMELLIA_128-
_GCM_SHA256 =0xC082, TLS_DH_DSS_WITH_CAMELLIA_256_GCM_SHA384 =0xC083, TLS_DH_AN-
ON_WITH_CAMELLIA_128_GCM_SHA256 =0xC084,
TLS_DH_ANON_WITH_CAMELLIA_256_GCM_SHA384 =0xC085, TLS_ECDHE_ECDSA_WITH_CAME-
LLIA_128_GCM_SHA256 =0xC086, TLS_ECDHE_ECDSA_WITH_CAMELLIA_256_GCM_SHA384 =0x-
C087, TLS_ECDH_ECDSA_WITH_CAMELLIA_128_GCM_SHA256 =0xC088,
TLS_ECDH_ECDSA_WITH_CAMELLIA_256_GCM_SHA384 =0xC089, TLS_ECDHE_RSA_WITH_CAME-
LLIA_128_GCM_SHA256 =0xC08A, TLS_ECDHE_RSA_WITH_CAMELLIA_256_GCM_SHA384 =0xC08B,
TLS_ECDH_RSA_WITH_CAMELLIA_128_GCM_SHA256 =0xC08C,
TLS_ECDH_RSA_WITH_CAMELLIA_256_GCM_SHA384 =0xC08D, TLS_PSK_WITH_CAMELLIA_128_G-
CM_SHA256 =0xC08E, TLS_PSK_WITH_CAMELLIA_256_GCM_SHA384 =0xC08F, TLS_DHE_PSK_WIT-
H_CAMELLIA_128_GCM_SHA256 =0xC090,
TLS_DHE_PSK_WITH_CAMELLIA_256_GCM_SHA384 =0xC091, TLS_RSA_PSK_WITH_CAMELLIA_-
128_GCM_SHA256 =0xC092, TLS_RSA_PSK_WITH_CAMELLIA_256_GCM_SHA384 =0xC093, TLS_PS-
K_WITH_CAMELLIA_128_CBC_SHA256 =0xC094,
TLS_PSK_WITH_CAMELLIA_256_CBC_SHA384 =0xC095, TLS_DHE_PSK_WITH_CAMELLIA_128_CB-
C_SHA256 =0xC096, TLS_DHE_PSK_WITH_CAMELLIA_256_CBC_SHA384 =0xC097, TLS_RSA_PSK_-
WITH_CAMELLIA_128_CBC_SHA256 =0xC098,
TLS_RSA_PSK_WITH_CAMELLIA_256_CBC_SHA384 =0xC099, TLS_ECDHE_PSK_WITH_CAMELL-
IA_128_CBC_SHA256 =0xC09A, TLS_ECDHE_PSK_WITH_CAMELLIA_256_CBC_SHA384 =0xC09B,
TLS_RSA_WITH_AES_128_CCM =0xC09C,
TLS_RSA_WITH_AES_256_CCM =0xC09D, TLS_DHE_RSA_WITH_AES_128_CCM =0xC09E, TLS_DHE-
_RSA_WITH_AES_256_CCM =0xC09F, TLS_RSA_WITH_AES_128_CCM_8 =0xC0A0,
TLS_RSA_WITH_AES_256_CCM_8 =0xC0A1, TLS_DHE_RSA_WITH_AES_128_CCM_8 =0xC0A2, TLS-
_DHE_RSA_WITH_AES_256_CCM_8 =0xC0A3, TLS_PSK_WITH_AES_128_CCM =0xC0A4,
TLS_PSK_WITH_AES_256_CCM =0xC0A5, TLS_DHE_PSK_WITH_AES_128_CCM =0xC0A6, TLS_DHE-
_PSK_WITH_AES_256_CCM =0xC0A7, TLS_PSK_WITH_AES_128_CCM_8 =0xC0A8,
TLS_PSK_WITH_AES_256_CCM_8 =0xC0A9, TLS_PSK_DHE_WITH_AES_128_CCM_8 =0xC0AA, TLS-
_PSK_DHE_WITH_AES_256_CCM_8 =0xC0AB, TLS_ECDHE_ECDSA_WITH_AES_128_CCM =0xC0AC,
TLS_ECDHE_ECDSA_WITH_AES_256_CCM =0xC0AD, TLS_ECDHE_ECDSA_WITH_AES_128_CCM_8 =
0xC0AE, TLS_ECDHE_ECDSA_WITH_AES_256_CCM_8 =0xC0AF, TLS_ECDHE_RSA_WITH_CHACH-
A20_POLY1305_SHA256 =0xCCA8,
TLS_ECDHE_ECDSA_WITH_CHACHA20_POLY1305_SHA256 =0xCCA9, TLS_DHE_RSA_WITH_CHA-
CHA20_POLY1305_SHA256 =0xCCAA, TLS_PSK_WITH_CHACHA20_POLY1305_SHA256 =0xCCAB,
TLS_ECDHE_PSK_WITH_CHACHA20_POLY1305_SHA256 =0xCCAC,
TLS_DHE_PSK_WITH_CHACHA20_POLY1305_SHA256 =0xCCAD, TLS_RSA_PSK_WITH_CHACHA20-
_POLY1305_SHA256 =0xCCAE }

```

- enum `wifitype_t` { `WIFIMNG`, `WIFIDAT`, `WIFICTL` }
- enum `wifisub_t` {
 `WIFI0`, `WIFI8`, `WIFI4`, `WIFI12`,
 `WIFI2`, `WIFI10`, `WIFI6`, `WIFI14`,
 `WIFI1`, `WIFI9`, `WIFI5`, `WIFI13`
}

```

WIFI3, WIFI11, WIFI7, WIFI15 }
• enum wifictext_t {
    WCTRL0, SSW, GRANT, WCTRL3,
    POLL, SSWACK, DMGDTs, WCTRL7,
    WCTRL8, SSWfdbck, DMGCTS, WCTRL11,
    SPR, WCTRL13, GRANTACK }
• enum wimaxmode_t { OFDM, OFDMA, AGMH_OFDM, AGMH_OFDMA }
• enum macsecmode_t {
    AES_GCM_128, AES_GCM_256, AES_GCM_XPN_128, AES_GCM_XPN_256,
    NULL_MACSEC }
• enum dir_id_t { DIR_IN, DIR_FWD, DIR_OUT }
• enum role_id_t { ROLE_INITIATOR =0x08, ROLE_RESPONDER =0x20, ROLE_ANY =0x28 }
• enum ipsec_mode_t { TRANSPORT_MODE, TUNNEL_MODE }
• enum exchg_t {
    EXCHG_IKE_SA_INIT =34, EXCHG_IKE_AUTH, EXCHG_CREATE_CHILD_SA, EXCHG_INFORMATIONAL,
    EXCHG_IKE_SESSION_RESUME, EXCHG_GSA_AUTH, EXCHG_GSA_REGISTRATION, EXCHG_GSA_REKEY,
    EXCHG_IKE_PARSE =253, EXCHG_IKE resend, EXCHG_IKE_LISTEN }
• enum ike_pyld_t {
    PYLD_NONE =0, PYLD_SA =33, PYLD_KE, PYLD_IDI,
    PYLD_IDR, PYLD_CERT, PYLD_CERTREQ, PYLD_AUTH,
    PYLD_NINR, PYLD_N, PYLD_D, PYLD_V,
    PYLD_TSI, PYLD_TSR, PYLD_SK, PYLD_CP,
    PYLD_EAP, PYLD_GSPM, PYLD_IDG, PYLD_GSA,
    PYLD_KD, PYLD_SKF, PYLD_PS }
• enum transform_type_t {
    IKE_ENCR =1, IKE_PRF, IKE_INTEG, IKE_DH,
    IKE_ESN }
• enum transform_attr_t { ATTR_KEY_LENGTH =14 }
• enum encr_id_t {
    ENCR_DES_IV64 =1, ENCR_DES, ENCR_3DES, ENCR_RC5,
    ENCR_IDEA, ENCR_CAST, ENCR_BLOWFISH, ENCR_3IDEA,
    ENCR_DES_IV32, ENCR_RSVD1, ENCR_NULL, ENCR_AES_CBC,
    ENCR_AES_CTR, ENCR_AES_CCM_8, ENCR_AES_CCM_12, ENCR_AES_CCM_16,
    ENCR_UNASSIGNED, ENCR_AES_GCM_8, ENCR_AES_GCM_12, ENCR_AES_GCM_16,
    ENCR_NULL_AUTH_AES_GMAC, ENCR_RSVD_AES_XTS, ENCR_CAMELLIA_CBC, ENCR_CAMELLIA_CTR,
    ENCR_CAMELLIA_CCM_8, ENCR_CAMELLIA_CCM_12, ENCR_CAMELLIA_CCM_16, ENCR_CHACHA20_POLY1305,
    ENCR_AES_CCM_8_IIV, ENCR_AES_GCM_16_IIV, ENCR_CHACHA20_POLY1305_IIV }
• enum prf_id_t {
    PRF_HMAC_MD5 =1, PRF_HMAC_SHA1, PRF_HMAC_TIGER, PRF_AES128_XCBC,
    PRF_HMAC_SHA2_256, PRF_HMAC_SHA2_384, PRF_HMAC_SHA2_512, PRF_AES128_CMAC }
• enum integ_id_t {
    AUTH_NONE, AUTH_HMAC_MD5_96, AUTH_HMAC_SHA1_96, AUTH_DES_MAC,
    AUTH_KPDK_MD5, AUTH_AES_XCBC_96, AUTH_HMAC_MD5_128, AUTH_HMAC_SHA1_160,
    AUTH_AES_CMAC_96, AUTH_AES_128_GMAC, AUTH_AES_192_GMAC, AUTH_AES_256_GMAC,
    AUTH_HMAC_SHA2_256_128, AUTH_HMAC_SHA2_384_192, AUTH_HMAC_SHA2_512_256 }
• enum dh_id_t {
    DH_NONE, DH_MODP_768, DH_MODP_1024, DH_RSVD3,
    DH_RSVD4, DH_MODP_1536, DH_UNASSIGNED6, DH_UNASSIGNED7,
    DH_UNASSIGNED8, DH_UNASSIGNED9, DH_UNASSIGNED10, DH_UNASSIGNED11,
    DH_UNASSIGNED12, DH_UNASSIGNED13, DH_MODP_2048, DH_MODP_3072,
    DH_MODP_4096, DH_MODP_6144, DH_MODP_8192, DH_ECP_256,
    DH_ECP_384, DH_ECP_521, DH_MODP_1024_160, DH_MODP_2048_224,
    DH_MODP_2048_256, DH_ECP_192_RANDOM, DH_ECP_224_RANDOM, DH_BRAINPOOL_P224R1,
    DH_BRAINPOOL_P256R1, DH_BRAINPOOL_P384R1, DH_BRAINPOOL_P512R1, DH_CURVE_25519,

```

```

DH_CURVE_448 }

• enum esn_id_t { ESN_NO, ESN_YES }

• enum id_type_t {
    ID_IPV4_ADDR =1, ID_FQDN, ID_RFC822_ADDR, ID_UNASSIGNED4,
    ID_IPV6_ADDR, ID_UNASSIGNED6, ID_UNASSIGNED7, ID_UNASSIGNED8,
    ID_DER ASN1_DN, ID_DER ASN1_GN, ID_KEY_ID, ID_FC_NAME,
    ID_NULL }

• enum encoding_t {
    CERT_X509_PKCS_7 =1, CERT_PGP, CERT_DNS, CERT_X509_SIGNATURE,
    CERT_RESERVED5, CERT_KERBEROS, CERT_CRL, CERT_ARL,
    CERT_SPKI, CERT_RAW_RSA, CERT_X509_ATTRIBUTE, CERT_HASH_URL,
    CERT_HASH_URL_BUNDLE, CERT_OSCP_CONTENT, CERT_RAW_PUBLIC_KEY }

• enum auth_method_t {
    AUTH_RSA =1, AUTH_PSK, AUTH_DSS, AUTH_UNASSIGNED4,
    AUTH_UNASSIGNED5, AUTH_UNASSIGNED6, AUTH_UNASSIGNED7, AUTH_UNASSIGNED8,
    AUTH_ECDSA_P256, AUTH_ECDSA_P384, AUTH_ECDSA_P512, AUTH_GENERIC_SECURE_PASSWORD,
    AUTH_NULL, AUTH_DIGITAL_SIGNATURE }

• enum notify_err_t {
    NERR_UNSUPPORTED_CRIT_PAYLD =1, NERR_RSVD2, NERR_RSVD3, NERR_INVALID_IKE_SPI,
    NERR_INVALID_MAJOR_VERSION, NERR_RSVD6, NERR_INVALID_SYNTAX, NERR_RSVD8,
    NERR_INVALID_MESSAGE_ID, NERR_RSVD10, NERR_INVALID_SPI, NERR_RSVD12,
    NERR_RSVD13, NERR_NO_PROPOSAL_CHOSEN, NERR_RSVD15, NERR_RSVD16,
    NERR_INVALID_KE_PAYLOAD, NERR_RSVD18, NERR_RSVD19, NERR_RSVD20,
    NERR_RSVD21, NERR_RSVD22, NERR_RSVD23, NERR_AUTH_FAILED,
    NERR_RSVD25, NERR_RSVD26, NERR_RSVD27, NERR_RSVD28,
    NERR_RSVD29, NERR_RSVD30, NERR_RSVD31, NERR_RSVD32,
    NERR_RSVD33, NERR_SINGLE_PAIR_REQUIRED, NERR_NO_ADDITIONAL_SAS, NERR_INTERNAL_ADDRESS_FAILURE,
    NERR_FAILED_CP_REQUIRED, NERR_TS_UNACCEPTABLE, NERR_INVALID_SELECTORS, NERR_UNACCEPTABLE_ADDRESSES,
    NERR_UNEXPECTED_NAT_DETECTED, NERR_USE_ASSIGNED_HOA, NERR_TEMPORARY_FAILURE, NERR_CHILD_SA_NOT_FOUND,
    NERR_INVALID_GROUP_ID, NERR_AUTHORIZATION_FAILED }

• enum notify_status_t {
    NSTAT_INITIAL_CONTACT =16384, NSTAT_SET_WINDOW_SIZE, NSTAT_ADDITIONAL_TS_POSSIBLE, NSTAT_IPCOMP_SUPPORTED,
    NSTAT_NAT_DETECTION_SOURCE_IP, NSTAT_NAT_DETECTION_DESTINATION_IP, NSTAT_COOKIE, NSTAT_USE_TRANSPORT_MODE,
    NSTAT_HTTP_CERT_LOOKUP_SUPPORTED, NSTAT_REKEY_SA, NSTAT_ESP_TFC_PADDING_NOT_SUPPORTED, NSTAT_NON_FIRST_FRAGMENTS_ALSO,
    NSTAT_MOBIKE_SUPPORTED, NSTAT_ADDITIONAL_IP4_ADDRESS, NSTAT_ADDITIONAL_IP6_ADDRESS, NSTAT_NO_ADDITIONAL_ADDRESSES,
    NSTAT_UPDATE_SA_ADDRESSES, NSTAT_COOKIE2, NSTAT_NO_NATS_ALLOWED, NSTAT_AUTH_LIFETIME,
    NSTAT_MULTIPLE_AUTH_SUPPORTED, NSTAT_ANOTHER_AUTH_FOLLOWS, NSTAT_REDIRECT_SUPPORTED, NSTAT_REDIRECT,
    NSTAT_REDIRECTED_FROM, NSTAT_TICKET_LT_OPAQUE, NSTAT_TICKET_REQUEST, NSTAT_TICKET_ACK,
    NSTAT_TICKET_NACK, NSTAT_TICKET_OPAQUE, NSTAT_LINK_ID, NSTAT_USE_WESP_MODE,
    NSTAT_ROHC_SUPPORTED, NSTAT_EAP_ONLY_AUTHENTICATION, NSTAT_CHILDLESS_IKEV2_SUPPORTED, NSTAT_QUICK_CRASH_DETECTION,
    NSTAT_IKEV2_MESSAGE_ID_SYNC_SUPPORTED, NSTAT_IPSEC_REPLY_COUNTER_SYNC_SUPPORTED, NSTAT_IKEV2_MESSAGE_ID_SYNC, NSTAT_IPSEC_REPLY_COUNTER_SYNC,
    NSTAT_SECURE_PASSWORD_METHODS, NSTAT_PSK_PERSIST, NSTAT_PSK_CONFIRM, NSTAT_ERX_SUPPORTED,
    NSTAT_IFOM_CAPABILITY, NSTAT_SENDER_REQUEST_ID, NSTAT_IKEV2_FRAGMENTATION_SUP-
}

```

```

PORTED, NSTAT_SIGNATURE_HASH_ALGORITHMS,
NSTAT_CLONE_IKE_SA_SUPPORTED, NSTAT_CLONE_IKE_SA, NSTAT_PUZZLE, NSTAT_USE_PPK,
NSTAT_PPK_IDENTITY, NSTAT_NO_PPK_AUTH }

• enum ike_ipcomp_t { IPCOMP_OUI =1, IPCOMP_DEFLATE, IPCOMP_LZS, IPCOMP_LZJH }

• enum ip_proto_t {
IP_PROTO_ANY, IP_PROTO_ICMP, IP_PROTO_TCP, IP_PROTO_UDP,
IP_PROTO_ICMPV6, IP_PROTO_MH }

• enum protocol_id_t {
PROTO_IKE =1, PROTO_AH, PROTO_ESP, PROTO_FC_ESP_HDR,
PROTO_FC_CT_AUTH }

• enum ike_ts_t { TS_IPV4_ADDR_RANGE =7, TS_IPV6_ADDR_RANGE, TS_FC_ADDR_RANGE }

• enum cfg_resp_t { CFG_REQUEST =1, CFG_REPLY, CFG_SET, CFG_ACK }

• enum cfg_attr_t {
CFG_INTERNAL_IP4_ADDRESS =1, CFG_INTERNAL_IP4_NETMASK, CFG_INTERNAL_IP4_DNS, CFG_
INTERNAL_IP4_NBNS,
CFG_RSVD5, CFG_INTERNAL_IP4_DHCP, CFG_APPLICATION_VERSION, CFG_INTERNAL_IP6_ADD-
RESS,
CFG_RSVD9, CFG_INTERNAL_IP6_DNS, CFG_RSVD11, CFG_INTERNAL_IP6_DHCP,
CFG_INTERNAL_IP4_SUBNET, CFG_SUPPORTED_ATTRIBUTES, CFG_INTERNAL_IP6_SUBNET, CF-
G_MIP6_HOME_PREFIX,
CFG_INTERNAL_IP6_LINK, CFG_INTERNAL_IP6_PREFIX, CFG_HOME_AGENT_ADDR, CFG_P_CSC-
G_IP4_ADDR,
CFG_P_CSCF_IP6_ADDR, CFG_FTT_KAT, CFG_EXT_SRC_IP4_NAT, CFG_TIMEOUT_LIVENESS,
CFG_INTERNAL_DNS_DOMAIN, CFG_INTERNAL_DNS_SEC_TA }

• enum ike_gateway_t { GATEWAY_IPV4_VPN_ADDR =1, GATEWAY_IPV6_VPN_ADDR, GATEWAY_FQD-
N_VPN_ADDR }

• enum rohc_attr_t {
ROHC_MAX_CID =1, ROHC_PROFILE, ROHC_INTEG, ROHC_ICV_LEN,
ROHC_MRRU }

• enum ike_secpass_t { SECURE_PASSWORD_PACE =1, SECURE_PASSW0RD_AUGPAKE, SECURE_P-
ASSWORD_PSK_AUTH }

• enum ike_hash_t {
HASH_SHA1, HASH_SHA2_256, HASH_SHA2_384, HASH_SHA2_512,
HASH_IDENTITY }

• enum rsapadtype_t { PKCS15, PSS }

```

Functions

- Begin_Enum_String (field_t)
- Begin_Enum_String (endian_t)
- Begin_Enum_String (platform_t)
- Begin_Enum_String (pad_t)
- Begin_Enum_String (protocol_t)
- *param ETH_P PPP_MP Dummy type
for PPP MP frames(0x0008)*@param ETH_P_LOCALTALK-Localtalk pseudo type(0x0009)*@param ETH-
_P_CAN-CAN Begin_Enum_String (ethid_t)
- Begin_Enum_String (direction_t)
- Begin_Enum_String (cipher_t)
- Begin_Enum_String (auth_t)
- Begin_Enum_String (iana_t)
- Begin_Enum_String (err_t)
- Begin_Enum_String (replay_t)
- Begin_Enum_String (ipmode_t)
- Begin_Enum_String (icmppmsg_t)
- Begin_Enum_String (icmpcode_t)
- Begin_Enum_String (tlsver_t)

- `Begin_Enum_String (tlstype_t)`
- `Begin_Enum_String (tlslvl_t)`
- `Begin_Enum_String (tls_handshake_t)`
- `Begin_Enum_String (tls_error_t)`
- `Begin_Enum_String (srtpcipher_t)`
- `Begin_Enum_String (tls_ciphersuite_t)`
- `Begin_Enum_String (wifitype_t)`
- `Begin_Enum_String (wifisub_t)`
- `Begin_Enum_String (wifictext_t)`
- `Begin_Enum_String (wimaxmode_t)`
- `Begin_Enum_String (macsecmode_t)`
- `Begin_Enum_String (dir_id_t)`
- `Begin_Enum_String (role_id_t)`
- `Begin_Enum_String (ipsec_mode_t)`
- `Begin_Enum_String (exchg_t)`
- `Begin_Enum_String (ike_pyld_t)`
- `Begin_Enum_String (transform_type_t)`
- `Begin_Enum_String (transform_attr_t)`
- `Begin_Enum_String (encr_id_t)`
- `Begin_Enum_String (prf_id_t)`
- `Begin_Enum_String (integ_id_t)`
- `Begin_Enum_String (dh_id_t)`
- `Begin_Enum_String (esn_id_t)`
- `Begin_Enum_String (id_type_t)`
- `Begin_Enum_String (encoding_t)`
- `Begin_Enum_String (auth_method_t)`
- `Begin_Enum_String (notify_err_t)`
- `Begin_Enum_String (notify_status_t)`
- `Begin_Enum_String (ike_ipcomp_t)`
- `Begin_Enum_String (ip_proto_t)`
- `Begin_Enum_String (protocol_id_t)`
- `Begin_Enum_String (ike_ts_t)`
- `Begin_Enum_String (cfg_resp_t)`
- `Begin_Enum_String (cfg_attr_t)`
- `Begin_Enum_String (ike_gateway_t)`
- `Begin_Enum_String (rohc_attr_t)`
- `Begin_Enum_String (ike_secpass_t)`
- `Begin_Enum_String (ike_hash_t)`
- `Begin_Enum_String (rsapadtype_t)`
- static unsigned short `U8TO16` (const unsigned char *p)
- static void `U16TO8` (unsigned char *p, unsigned short v)
- static unsigned long `U8TO32` (const unsigned char *p)
- static void `U32TO8` (unsigned char *p, unsigned long v)
- static void `jpoly1305_blocks` (`jpoly1305_state_internal_t` *st, const unsigned char *m, `size_t` bytes)
- static unsigned long long `U8TO64` (const unsigned char *p)
- static void `U64TO8` (unsigned char *p, unsigned long long v)
- static void `jpoly1305_blocks` (`jpoly1305_state_internal_t` *st, const unsigned char *m, `size_t` bytes)
- static vector unsigned int `vec_recursion` (vector unsigned int a, vector unsigned int b, vector unsigned int c, vector unsigned int d)
- void `sfmt_gen_rand_all` ()
- static void `gen_rand_array` (`w128_t` *array, int size)
- static void `swap` (`w128_t` *array, int size)
- static void `do_recursion` (`w128_t` *r, `w128_t` *a, `w128_t` *b, `w128_t` *c, `w128_t` *d)
- static void `rshift128` (`w128_t` *out, `w128_t` const *in, int shift)
- static void `lshift128` (`w128_t` *out, `w128_t` const *in, int shift)

- static void `neon_recursion` (uint32x4_t *r, uint32x4_t a, uint32x4_t b, uint32x4_t c, uint32x4_t d)
- static void `gen_rand_array` (`w128_t` *array, int size)
- static void `mm_recursion` (`_m128i` *r, `_m128i` a, `_m128i` b, `_m128i` c, `_m128i` *d)
- static void `gen_rand_array` (`w128_t` *array, int size)
- static void `mm_recursion` (`_m128i` *r, `_m128i` a, `_m128i` b, `_m128i` c, `_m128i` d)
- static void `gen_rand_array` (`w128_t` *array, int size)

Variables

- `End_Enum_String`

6.5.1 Detailed Description

`mtrand.h` C++ include file for MT19937, with initialization improved 2002/1/26. Coded by Takuji Nishimura and Makoto Matsumoto. Ported to C++ by Jasper Bedaux 2003/1/1 (see <http://www.bedaux.net/mtrand/>). The generators returning floating point numbers are based on a version by Isaku Wada, 2002/01/09

Copyright (C) 1997 - 2002, Makoto Matsumoto and Takuji Nishimura, All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
3. The names of its contributors may not be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Any feedback is very welcome. <http://www.math.keio.ac.jp/matumoto/emt.html> email: matumoto@math.keio.ac.jp

Feedback about the C++ port should be sent to Jasper Bedaux, see <http://www.bedaux.net/mtrand/> for e-mail address and info.

[jpoly1305](#) implementation using 16 bit * 16 bit = 32 bit multiplication and 32 bit addition

Protocol++ modified by : John Peter Greninger © John Peter Greninger 2015-2019 All Rights Reserved

All copyrights and trademarks are the property of their respective owners

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer

- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution
- Any and all modifications must be returned to John Peter Greninger at GitHub.com <https://github.com/jpgreninger/protocolpp> for evaluation. Inclusion of modifications in the source code shall be determined solely by John Peter Greninger. Failure to provide modifications shall render this license NULL and VOID and revoke any rights to use of Protocol++
- Commercial use requires a fee-based license obtainable at www.protocolpp.com
- Academic use requires written and notarized permission from John Peter and Sheila Rocha Greninger
- US Copyrights at <https://www.copyright.gov/>
 - TXU002059872 (Version 1.0.0)
 - TXU002066632 (Version 1.2.7)
 - TXU002082674 (Version 1.4.0)
 - TXU002097880 (Version 2.0.0)

The names of its contributors may not be used to endorse or promote products derived from this software without specific prior written permission

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE

[jpoly1305](#) implementation using 32 bit * 32 bit = 64 bit multiplication and 64 bit addition

Protocol++ modified by : John Peter Greninger © John Peter Greninger 2015-2019 All Rights Reserved

All copyrights and trademarks are the property of their respective owners

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer
- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution
- Any and all modifications must be returned to John Peter Greninger at GitHub.com <https://github.com/jpgreninger/protocolpp> for evaluation. Inclusion of modifications in the source code shall be determined solely by John Peter Greninger. Failure to provide modifications shall render this license NULL and VOID and revoke any rights to use of Protocol++
- Commercial use requires a fee-based license obtainable at www.protocolpp.com
- Academic use requires written and notarized permission from John Peter and Sheila Rocha Greninger
- US Copyrights at <https://www.copyright.gov/>
 - TXU002059872 (Version 1.0.0)
 - TXU002066632 (Version 1.2.7)
 - TXU002082674 (Version 1.4.0)

- TXu002097880 (Version 2.0.0)

The names of its contributors may not be used to endorse or promote products derived from this software without specific prior written permission

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE

[jpoly1305](#) implementation using 64 bit * 64 bit = 128 bit multiplication and 128 bit addition

Protocol++ modified by : John Peter Greninger © John Peter Greninger 2015-2019 All Rights Reserved

All copyrights and trademarks are the property of their respective owners

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer
- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution
- Any and all modifications must be returned to John Peter Greninger at GitHub.com <https://github.com/jpgreninger/protocolpp> for evaluation. Inclusion of modifications in the source code shall be determined solely by John Peter Greninger. Failure to provide modifications shall render this license NULL and VOID and revoke any rights to use of Protocol++
- Commercial use requires a fee-based license obtainable at www.protocolpp.com
- Academic use requires written and notarized permission from John Peter and Sheila Rocha Greninger
- US Copyrights at [https://www.copyright.gov/](http://www.copyright.gov/)
 - TXu002059872 (Version 1.0.0)
 - TXu002066632 (Version 1.2.7)
 - TXu002082674 (Version 1.4.0)
 - TXu002097880 (Version 2.0.0)

The names of its contributors may not be used to endorse or promote products derived from this software without specific prior written permission

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE

[jpoly1305](#) implementation using 8 bit * 8 bit = 16 bit multiplication and 32 bit addition based on the public domain reference version in supercop by djb

Protocol++ modified by : John Peter Greninger © John Peter Greninger 2015-2019 All Rights Reserved

All copyrights and trademarks are the property of their respective owners

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer
- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution
- Any and all modifications must be returned to John Peter Greninger at GitHub.com <https://github.com/jpgreninger/protocolpp> for evaluation. Inclusion of modifications in the source code shall be determined solely by John Peter Greninger. Failure to provide modifications shall render this license NULL and VOID and revoke any rights to use of Protocol++
- Commercial use requires a fee-based license obtainable at www.protocolpp.com
- Academic use requires written and notarized permission from John Peter and Sheila Rocha Greninger
- US Copyrights at <https://www.copyright.gov/>
 - TXu002059872 (Version 1.0.0)
 - TXu002066632 (Version 1.2.7)
 - TXu002082674 (Version 1.4.0)
 - TXu002097880 (Version 2.0.0)

The names of its contributors may not be used to endorse or promote products derived from this software without specific prior written permission

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE

6.5.2 Typedef Documentation

6.5.2.1 `typedef struct ProtocolPP::jpoly1305_state_internal_t ProtocolPP::jpoly1305_state_internal_t`

6.5.2.2 `typedef union W128_T ProtocolPP::w128_t`

128-bit data type

6.5.3 Enumeration Type Documentation

6.5.3.1 `enum ProtocolPP::auth_method_t`

IKEv2 AUTH METHOD fields

Parameters

AUTH_RSA	- RSA Authentication mode
AUTH_PSK	- PSK Authentication mode
AUTH_DSS	- DSS Authentication mode
AUTH_UNASSIGNED4	- Unassigned
AUTH_UNASSIGNED5	- Unassigned
AUTH_UNASSIGNED6	- Unassigned
AUTH_UNASSIGNED7	- Unassigned
AUTH_UNASSIGNED8	- Unassigned
AUTH_ECDSA_P256	- ECDSA with SHA-256 on the P-256 Curve
AUTH_ECDSA_P384	- ECDSA with SHA-384 on the P-384 Curve
AUTH_ECDSA_P512	- ECDSA with SHA-512 on the P-512 Curve
AUTH_GENERIC_SECURE_PASSWORD	- Generic Secure Password Authentication Method
AUTH_NULL	- No PRF Authentication mode
AUTH_DIGITAL_SIGNATURE	- Digital Signature

Enumerator

AUTH_RSA RSA Authentication.

AUTH_PSK PSK Authentication mode.

AUTH_DSS DSS Authentication mode.

AUTH_UNASSIGNED4 Unassigned.

AUTH_UNASSIGNED5 Unassigned.

AUTH_UNASSIGNED6 Unassigned.

AUTH_UNASSIGNED7 Unassigned.

AUTH_UNASSIGNED8 Unassigned.

AUTH_ECDSA_P256 ECDSA with SHA-256 on the P-256 Curve.

AUTH_ECDSA_P384 ECDSA with SHA-384 on the P-384 Curve.

AUTH_ECDSA_P512 ECDSA with SHA-512 on the P-512 Curve.

AUTH_GENERIC_SECURE_PASSWORD Generic Secure Password Authentication Method.

AUTH_NULL No PRF Authentication method.

AUTH_DIGITAL_SIGNATURE Digital Signature.

6.5.3.2 enum ProtocolPP::auth_t

authentication types

Enumerator

NULL_AUTH NULL authentication - no authentication present.

MD5 MD5 authentication mode with 16 byte MAC.

SHA1 SHA1 authentication mode with 20 byte MAC.
SHA224 SHA224 authentication mode with 28 byte MAC.
SHA256 SHA256 authentication mode with 32 byte MAC.
SHA384 SHA384 authentication mode with 48 byte MAC.
SHA512 SHA512 authentication mode with 64 byte MAC.
SHA3_224 SHA3_224 authentication mode with 28 byte MAC.
SHA3_256 SHA3_256 authentication mode with 32 byte MAC.
SHA3_384 SHA3_384 authentication mode with 48 byte MAC.
SHA3_512 SHA3_512 authentication mode with 64 byte MAC.
AES_XCBC_MAC AES-XCBC-MAC authentication mode with 12 byte MAC.
POLY1305 POLY1305 authentication mode with 16 byte MAC.
AES_CMAC AES-CMAC authenticate only with 16 byte MAC.
AES_GMAC AES-GMAC authenticate only with 16 byte MAC.
CRC32 CRC32 IEEE polynomial with 4-byte output.
SM3 SM3 mode authentication.
SNOWA SNOW3G F9 mode authentication.
ZUCA ZUC F9 mode authentication.
NONE_AUTH None authentication.

6.5.3.3 enum ProtocolPP::cfg_attr_t

IKE Configuration Payload Attribute Types

Parameters

CFG_INTERNAL_IP4_ADDRESS	- Configure an internal IPv4 Address
CFG_INTERNAL_IP4_NETMASK	- Configure an internal IPv4 NETMASK
CFG_INTERNAL_IP4_DNS	- Configure a IPv4 DNS setting
CFG_INTERNAL_IP4_NBNS	- Configure a IPv4 NBNS setting
CFG_RSVD5	- Reserved Configuration
CFG_INTERNAL_IP4_DHCP	- Configure a IPv4 DHCP setting
CFG_APPLICATION_VERSION	- Configure Application Version
CFG_INTERNAL_IP6_ADDRESS	- Configure an internal IPv6 Address
CFG_RSVD9	- Reserved Configuration
CFG_INTERNAL_IP6_DNS	- Configure a IPv6 DNS setting

<i>CFG_RSVD11</i>	- Reserved Configuration
<i>CFG_INTERNAL_IP6_DHCP</i>	12 - Configure a IPv6 DHCP setting
<i>CFG_INTERNAL_IP4_SUBNET</i>	- Configure an internal IPv4 SUBNET
<i>CFG_SUPPORTED_ATTRIBUTES</i>	- Configure supported attributes
<i>CFG_INTERNAL_IP6_SUBNET</i>	- Configure an internal IPv6 SUBNET
<i>CFG_MIP6_HOME_PREFIX</i>	- MIP6 Home Prefix
<i>CFG_INTERNAL_IP6_LINK</i>	- Internal IPv6 Link
<i>CFG_INTERNAL_IP6_PREFIX</i>	- Internal IPv6 Prefix
<i>CFG_HOME_AGENT_ADDR</i>	- Home Agent Address
<i>CFG_P_CSCF_IP4_ADDR</i>	- P CSCF IPv4 Address
<i>CFG_P_CSCF_IP6_ADDR</i>	- P CSCF IPv6 Address
<i>CFG_FTT_KAT</i>	- FTT KAT
<i>CFG_EXT_SR_C_IP4_NAT</i>	- External Source IPv4 NAT Information
<i>CFG_TIMEOUT_LIVENESS</i>	- Timeout Period for Liveness Check Number
<i>CFG_INTERNAL_DNS_DOMAIN</i>	- Internal DNS Domain
<i>CFG_INTERNAL_DNS_SEC_TA</i>	- Internal DNS SEC TA

Enumerator

CFG_INTERNAL_IP4_ADDRESS Configure an internal IPv4 Address.

CFG_INTERNAL_IP4_NETMASK Configure an internal IPv4 NETMASK.

CFG_INTERNAL_IP4_DNS Configure a IPv4 DNS setting.

CFG_INTERNAL_IP4_NBNS Configure a IPv4 NBNS setting.

CFG_RSVD5 Reserved Configuration.

CFG_INTERNAL_IP4_DHCP Configure a IPv4 DHCP setting.

CFG_APPLICATION_VERSION Configure Application Version.

CFG_INTERNAL_IP6_ADDRESS Configure an internal IPv6 Address.

CFG_RSVD9 Reserved Configuration.

CFG_INTERNAL_IP6_DNS Configure a IPv6 DNS setting.

CFG_RSVD11 Reserved Configuration.

CFG_INTERNAL_IP6_DHCP Configure a IPv6 DHCP setting.

CFG_INTERNAL_IP4_SUBNET Configure an internal IPv4 SUBNET.

CFG_SUPPORTED_ATTRIBUTES Configure supported attributes.

CFG_INTERNAL_IP6_SUBNET Configure an internal IPv6 SUBNET.

CFG_MIP6_HOME_PREFIX MIP6 Home Prefix.

CFG_INTERNAL_IP6_LINK Internal IPv6 Link.

CFG_INTERNAL_IP6_PREFIX Internal IPv6 Prefix.
CFG_HOME_AGENT_ADDR Home Agent Address.
CFG_P_CSCG_IP4_ADDR P CSCF IPv4 Address.
CFG_P_CSCF_IP6_ADDR P CSCF IPv6 Address.
CFG_FTT_KAT FTT KAT.
CFG_EXT_SRC_IP4_NAT External Source IPv4 NAT Information.
CFG_TIMEOUT_LIVENESS Timeout Period for Liveness Check Number.
CFG_INTERNAL_DNS_DOMAIN Internal DNS Domain.
CFG_INTERNAL_DNS_SEC_TA Internal DNS SEC TA.

6.5.3.4 enum ProtocolPP::cfg_resp_t

IKE Configuration Payload CFG Types

Parameters

<i>CFG_REQUEST</i>	- Request the configuration
<i>CFG_REPLY</i>	- Reply to a configuration request
<i>CFG_SET</i>	- Set the configuration
<i>CFG_ACK</i>	- Acknowledge the configuration request

Enumerator

CFG_REQUEST Request the configuration.
CFG_REPLY Reply to a configuration request.
CFG_SET Set the configuration.
CFG_ACK Acknowledge the configuration request.

6.5.3.5 enum ProtocolPP::cipher_t

encryption ciphers

Enumerator

NULL_CIPHER NULL cipher - no encryption present.
DES_CBC DES-CBC encryption mode.
DES40_CBC DES-CBC encryption with 40-bit key.
TDES_CBC Triple-DES CBC encryption mode with 192-bit key.
AES_CBC AES-CBC encryption mode.
AES_CTR AES-CTR encryption mode.
AES_CCM AES-CCM dual encryption and authentication mode.
AES_GCM AES-GCM dual encryption and authentication mode.
CHACHA20 CHACHA20 encryption mode.
CHACHA20_POLY1305 CHACHA20 and POLY1305 AEAD dual mode.
CAMELLIA_CBC Camellia CBC encryption mode.
CAMELLIA_CTR Camellia CTR mode.
CAMELLIA_CCM Camellia CCM AEAD mode.
CAMELLIA_GCM Camellia GCM AEAD mode.
ARIA_CBC Korean cipher ARIA in CBC mode.

ARIA_CTR Korean Cipher ARIA in CTR mode.
ARIA_GCM Korean Cipher ARIA in GCM mode.
SEED_CBC Korean Block Cipher SEED in CBC mode.
SEED_CTR Korean Block Cipher SEED in CTR mode.
SEED_GCM Korean Block Cipher SEED in GCM mode.
SM4_CBC Chinese Block Cipher SM4 in CBC mode.
SM4_CTR Chinese Block Cipher SM4 in CTR mode.
SM4_GCM Chinese Block Cipher SM4 in GCM mode.
SM4_CCM Chinese Block Cipher SM4 in CCM mode.
ARC4 ARC4 Stream Cipher (weak not recommended)
SNOWE SNOW3G F8 encryption algorithm.
ZUCE ZUC F8 encryption algorithm.
NONE_ENC No encryption algorithm chosen.

6.5.3.6 enum ProtocolPP::dh_id_t

IKEv2 DH_ID fields

Parameters

<i>DH_NONE</i>	- No Diffie-Hellman
<i>DH_MODP_768</i>	- Diffie-Hellman MODP Group 768-bit
<i>DH_MODP_-1024</i>	- Diffie-Hellman MODP Group 1024-bit
<i>DH_RSVD3</i>	- Reserved
<i>DH_RSVD4</i>	- Reserved
<i>DH_MODP_-1536</i>	- Diffie-Hellman MODP Group 1536-bit
<i>DH_UNASSIGN-ED6</i>	- Unassigned
<i>DH_UNASSIGN-ED7</i>	- Unassigned
<i>DH_UNASSIGN-ED8</i>	- Unassigned
<i>DH_UNASSIGN-ED9</i>	- Unassigned
<i>DH_UNASSIGN-ED10</i>	- Unassigned
<i>DH_UNASSIGN-ED11</i>	- Unassigned
<i>DH_UNASSIGN-ED12</i>	- Unassigned
<i>DH_UNASSIGN-ED13</i>	- Unassigned
<i>DH_MODP_-2048</i>	- Diffie-Hellman MODP Group 2048-bit
<i>DH_MODP_-3072</i>	- Diffie-Hellman MODP Group 3072-bit

<code>DH_MODP_4096</code>	- Diffie-Hellman MODP Group 4096-bit
<code>DH_MODP_6144</code>	- Diffie-Hellman MODP Group 6144-bit
<code>DH_MODP_8192</code>	- Diffie-Hellman MODP Group 8192-bit
<code>DH_ECP_256</code>	- Diffie-Hellman Elliptic Curve with 256-bit
<code>DH_ECP_384</code>	- Diffie-Hellman Elliptic Curve with 384-bit
<code>DH_ECP_521</code>	- Diffie-Hellman Elliptic Curve with 521-bit
<code>DH_MODP_1024_160</code>	- Diffie-Hellman MODP Group 1024-bit with 160-bit Subgroup
<code>DH_MODP_2048_224</code>	- Diffie-Hellman MODP Group 2048-bit with 224-bit Subgroup
<code>DH_MODP_2048_256</code>	- Diffie-Hellman MODP Group 2048-bit with 256-bit Subgroup
<code>DH_ECP_192_RANDOM</code>	- Diffie-Hellman Elliptic Curve with 192-bit random curve
<code>DH_ECP_224_RANDOM</code>	- Diffie-Hellman Elliptic Curve with 224-bit random curve
<code>DH_BRAINPOOL_P224R1</code>	- Diffie-Hellman BrainPool curve with 224-bit prime
<code>DH_BRAINPOOL_P256R1</code>	- Diffie-Hellman BrainPool curve with 256-bit prime
<code>DH_BRAINPOOL_P384R1</code>	- Diffie-Hellman BrainPool curve with 384-bit prime
<code>DH_BRAINPOOL_P512R1</code>	- Diffie-Hellman BrainPool curve with 512-bit prime
<code>DH_CURVE_25519</code>	- Diffie-Hellman Elliptic Curve 25519
<code>DH_CURVE_448</code>	- Diffie-Hellman Elliptic Curve 448

Enumerator

`DH_NONE` No Diffie-Hellman.

`DH_MODP_768` Diffie-Hellman MODP Group 768-bit.

`DH_MODP_1024` Diffie-Hellman MODP Group 1024-bit.

`DH_RSVD3` Reserved.

`DH_RSVD4` Reserved.

`DH_MODP_1536` Diffie-Hellman MODP Group 1536-bit.

`DH_UNASSIGNED6` Unassigned.

`DH_UNASSIGNED7` Unassigned.

`DH_UNASSIGNED8` Unassigned.

`DH_UNASSIGNED9` Unassigned.

`DH_UNASSIGNED10` Unassigned.

`DH_UNASSIGNED11` Unassigned.

`DH_UNASSIGNED12` Unassigned.

`DH_UNASSIGNED13` Unassigned.

`DH_MODP_2048` MODP Group 2048-bit.

`DH_MODP_3072` MODP Group 3072-bit.

`DH_MODP_4096` MODP Group 4096-bit.

`DH_MODP_6144` MODP Group 6144-bit.

DH_MODP_8192 MODP Group 8192-bit.
DH_ECP_256 ECP Group 256-bit.
DH_ECP_384 ECP Group 384-bit.
DH_ECP_521 ECP Group 521-bit.
DH_MODP_1024_160 MODP Group 1024-bit with 160-bit Prime Order Subgroup.
DH_MODP_2048_224 MODP Group 2048-bit with 224-bit Prime Order Subgroup.
DH_MODP_2048_256 MODP Group 2048-bit with 256-bit Prime Order Subgroup.
DH_ECP_192_RANDOM ECP Group 192-bit Random.
DH_ECP_224_RANDOM ECP Group 224-bit Random.
DH_BRAINPOOL_P224R1 BrainPool P224R1.
DH_BRAINPOOL_P256R1 Brain Pool P256R1.
DH_BRAINPOOL_P384R1 BrainPool P384R1.
DH_BRAINPOOL_P512R1 BrainPool P512R1.
DH_CURVE_25519 Curve 25519.
DH_CURVE_448 Curve 448.

6.5.3.7 enum ProtocolPP::dir_id_t

IKEv2 DIR_ID fields

Parameters

DIR_IN	- Direction of processing is into host
DIR_FWD	- Forward packets on this stream
DIR_OUT	- Direction of processing is out of the host

Enumerator

DIR_IN Direction of processing is into host.
DIR_FWD Forward packets on this stream.
DIR_OUT Direction of processing is out of the host.

6.5.3.8 enum ProtocolPP::direction_t

processing direction

Parameters

ENCAP	- Encapsulation of data
DECAP	- Decapsulation of data
ENC	- Encryption of data
DEC	- Decryption of data
DOWNLINK	- For SNOW3G and ZUC tower to user
UPLINK	- For SNOW3G and ZUC user to tower

Enumerator

ENCAP Encapsulate the payload with protocol.
DECAP Decapsulate the packet with the protocol.
ENC Encryption.
DEC Decryption.
DOWNLINK For SNOW3G and ZUC tower to user.
UPLINK For SNOW3G and ZUC user to tower.
NODIR No direction selected.

6.5.3.9 enum ProtocolPP::encoding_t

IKEv2 CERT ENCODING fields

Parameters

<i>CERT_X509_PKCS_7</i>	- Certificate encoded as X509_PKCS_7
<i>CERT_PGP</i>	- Certificate encoded as PGP
<i>CERT_DNS</i>	- Certificate encoded as DNS
<i>CERT_X509_SIGNATURE</i>	- Certificate encoded as X509_SIGNATURE
<i>CERT_RESERVED5</i>	- Certificate encoded as RESERVED5
<i>CERT_KERBEROS</i>	- Certificate encoded as KERBEROS
<i>CERT_CRL</i>	- Certificate encoded as CRL
<i>CERT_ARL</i>	- Certificate encoded as ARL
<i>CERT_SPKI</i>	- Certificate encoded as SPKI
<i>CERT_RAW_RSA</i>	- Certificate encoded as RAW_RSA
<i>CERT_X509_ATTRIBUTE</i>	- Certificate encoded as X509_ATTRIBUTE
<i>CERT_HASH_URL</i>	- Certificate encoded as HASH_URL of X509
<i>CERT_HASH_URL_BUNDLE</i>	- Certificate encoded as HASH_URL_BUNDLE of X509
<i>CERT_OSCP_CONTENT</i>	- Certificate encoded as OSCP Content
<i>CERT_RAW_PUBLIC_KEY</i>	- Certificate encoded as raw public key

Enumerator

CERT_X509_PKCS_7 Certificate encoded as X509 PKCS #7.

CERT_PGP Certificate encoded as PGP.

CERT_DNS Certificate encoded as DNS.

CERT_X509_SIGNATURE Certificate encoded as X509 Signature.

CERT_RESERVED5 Certificate encoded as RESERVED #5.

CERT_KERBEROS Certificate encoded as KERBEROS.

CERT_CRL Certificate encoded as CRL.

CERT_ARL Certificate encoded as ARL.

CERT_SPKI Certificate encoded as SPKI.

CERT_RAW_RSA Certificate encoded as RAW RSA.

CERT_X509_ATTRIBUTE Certificate encoded as X509_ATTRIBUTE.

CERT_HASH_URL Certificate encoded as HASH_URL.

CERT_HASH_URL_BUNDLE Certificate encoded as HASH_URL_BUNDLE.

CERT_OSCP_CONTENT Certificate encoded as OCSP content.

CERT_RAW_PUBLIC_KEY Certificate encoded as raw public key.

6.5.3.10 enum ProtocolPP::encr_id_t

IKEv2 ENCR_ID fields

Parameters

<i>ENCR_DES_I-V64</i>	- Data Encryption Standard (DES) with 64-bit IV
<i>ENCR_DES</i>	- Data Encryption Standard (DES)
<i>ENCR_3DES</i>	- Triple DES with three separate keys
<i>ENCR_RC5</i>	- RC5 Encryption
<i>ENCR_IDEA</i>	- IDEA Encryption
<i>ENCR_CAST</i>	- Cast Encryption
<i>ENCR_BLOWFI-SH</i>	- Blowfish encryption
<i>ENCR_3IDEA</i>	- Three key IDEA
<i>ENCR_DES_I-V32</i>	- Data Encryption Standard (DES) with 32-bit IV
<i>ENCR_RSVD1</i>	- Reserved
<i>ENCR_NULL</i>	- NULL Encryption (NOT ALLOWED)
<i>ENCR_AES_C-BC</i>	- AES-CBC Encryption
<i>ENCR_AES_C-TR</i>	- AES-CTR Encryption
<i>ENCR_AES_C-CM_8</i>	- AES-CCM Encryption with 8-byte ICV
<i>ENCR_AES_C-CM_12</i>	- AES-CCM Encryption with 12-byte ICV
<i>ENCR_AES_C-CM_16</i>	- AES-CCM Encryption with 16-byte ICV
<i>ENCR_UNASSIGNED</i>	- Unassigned
<i>ENCR_AES_G-CM_8</i>	- AES-GCM Encryption with 8-byte ICV
<i>ENCR_AES_G-CM_12</i>	- AES-GCM Encryption with 12-byte ICV
<i>ENCR_AES_G-CM_16</i>	- AES-GCM Encryption with 16-byte ICV
<i>ENCR_NULL_AUTH_AES_GMAC_AC</i>	- Null Encryption with AES-GMAC Authentication (NOT ALLOWED)
<i>ENCR_RSVD-AES_XTS</i>	- Reserved for IEEE P1619 XTS-AES
<i>ENCR_CAMELLIA_CBC</i>	- Camellia-CBC Encryption
<i>ENCR_CAMELLIA_CTR</i>	- Camellia-CTR Encryption
<i>ENCR_CAMELLIA_CCM_8</i>	- Camellia-CCM Encryption with 8-byte ICV
<i>ENCR_CAMELLIA_CCM_12</i>	- Camellia-CCM Encryption with 12-byte ICV
<i>ENCR_CAMELLIA_CCM_16</i>	- Camellia-CCM Encryption with 16-byte ICV

<code>ENCR_CHACH-A20_POLY1305</code>	- CHACHA20 Encryption with POLY1305 Authentication and 16-byte ICV
<code>ENCR_AES_C-CM_8_IIV</code>	- AES-CCM Encryption with 8-byte ICV and Implicit IV
<code>ENCR_AES_G-CM_16_IIV</code>	- AES-GCM Encryption with 16-byte ICV and Implicit IV
<code>ENCR_CHACH-A20_POLY1305-IIV</code>	- CHACHA20 Encryption with POLY1305 Authentication and 16-byte ICV with Implicit IV

Enumerator

`ENCR_DES_IV64` Data Encryption Standard (DES) with 64-bit IV.

`ENCR_DES` Data Encryption Standard (DES)

`ENCR_3DES` Triple DES with separate keys.

`ENCR_RC5` RC5 Encryption.

`ENCR_IDEA` IDEA Encryption.

`ENCR_CAST` Cast Encryption.

`ENCR_BLOWFISH` Blowfish Encryption.

`ENCR_3IDEA` Three key IDEA Encryption.

`ENCR_DES_IV32` Data Encryption Standard with 32-bit IV.

`ENCR_RSVD1` Reserved.

`ENCR_NULL` Null Encryption (NOT ALLOWED)

`ENCR_AES_CBC` AES-CBC Encryption.

`ENCR_AESCTR` AES-CTR Encryption.

`ENCR_AES_CCM_8` AES-CCM Encryption with 8-byte ICV.

`ENCR_AES_CCM_12` AES-CCM Encryption with 12-byte ICV.

`ENCR_AES_CCM_16` AES-CCM Encryption with 16-byte ICV.

`ENCR_UNASSIGNED` Unassigned.

`ENCR_AES_GCM_8` AES-GCM Encryption with 8-byte ICV.

`ENCR_AES_GCM_12` AES-GCM Encryption with 12-byte ICV.

`ENCR_AES_GCM_16` AES-GCM Encryption with 16-byte ICV.

`ENCR_NULL_AUTH_AES_GMAC` Null Encryption with AES-GMAC Authentication (NOT ALLOWED)

`ENCR_RSVD_AES_XTS` Reserved for IEEE P1619 XTS-AES.

`ENCR_CAMELLIA_CBC` Camellia-CBC Encryption.

`ENCR_CAMELLIACTR` Camellia-CTR Encryption.

`ENCR_CAMELLIA_CCM_8` Camellia-CCM Encryption with 8-byte ICV.

`ENCR_CAMELLIA_CCM_12` Camellia-CCM Encryption with 12-byte ICV.

`ENCR_CAMELLIA_CCM_16` Camellia-CCM Encryption with 16-byte ICV.

`ENCR_CHACHA20_POLY1305` CHACHA20 Encryption with POLY1305 Authentication and 16-byte ICV.

`ENCR_AES_CCM_8_IIV` AES-CCM Encryption with 8-byte ICV and Implicit IV.

`ENCR_AES_GCM_16_IIV` AES-GCM Encryption with 16-byte ICV and Implicit IV.

`ENCR_CHACHA20_POLY1305_IIV` CHACHA20 Encryption with POLY1305 Authentication and 16-byte ICV with Implicit IV.

6.5.3.11 enum ProtocolPP::endian_t

Endian formatting for data. Endianness is within words, double words, etc.

Parameters

<i>BIG</i>	- Big endian with formatting as b[3]b[2]b[1]b[0]
<i>LITTLE</i>	- Little endian with formatting as b[0]b[1]b[2]b[3]

Enumerator

BIG Big endian with formatting as b[3]b[2]b[1]b[0].

LITTLE Little endian with formatting as b[0]b[1]b[2]b[3].

6.5.3.12 enum ProtocolPP::err_t

Status values for packet processing with format

Protocol (8-bits)	Error (8-bits)	Line Number (16-bits)
-------------------	----------------	-----------------------

Table 6.1: Format for Protocol++ Status Word

Parameters

<i>ERR_NONE</i>	- No error
<i>ERR_LATE</i>	- Anti-replay late packet error found
<i>ERR_REPLAY</i>	- Anti-replay error found
<i>ERR_ROLLOVER</i>	- Packet number rollover error
<i>ERR_ROLLUNDER</i>	- Packet number rollunder error
<i>ERR_PROGRAM</i>	- Programming error
<i>ERR_ICV</i>	- ICV check failure
<i>ERR_CRC</i>	- CRC check failure
<i>ERR_READ-ENDFILE</i>	- Read end of file
<i>ERR_READ-FAILFILE</i>	- Read failed to input file handle
<i>ERR_READ-BADFILE</i>	- Read of missing file handle
<i>ERR_WRITE-FAILFILE</i>	- Write failed to file output
<i>ERR_WRITE-BADFILE</i>	- Write error to file output
<i>ERR_CHECKSUM</i>	- Checksum error
<i>ERR_LISTEN</i>	- TCP server received packet while listening without SYN=1
<i>ERR_ACKNUM</i>	- TCP received ACKNUM is incorrect
<i>ERR_CLOSED</i>	- TCP session CLOSED during TX/RX
<i>ERR_BITS</i>	- TCP header reserved bits error
<i>ERR_TTL</i>	- TTL field of packet is zero
<i>ERR_JUMBOGRAMFORMAT</i>	- IPv6 JUMBOGRAM formatting error

<i>WARN_DUMMY</i>	- Dummy packet warning
<i>WARN_IPV6_R- ROUTE</i>	- IPv6 Route header but not for this destination
<i>WARN_ZERO_- DATA</i>	- Payload had zero length data
<i>ERR_ICMP_HD- R_EXT_LEN</i>	- Header extension length is ODD for ROUTE header
<i>ERR_ICMP_SE- GMENTS_LEFT</i>	- In ROUTE header, Segments left is larger than N
<i>ERR_MULTICA- ST_EXT_HDR</i>	- In ROUTE header, Multicast address present
<i>ERR_UNKNOW- N_ROUTE_TY- PE</i>	- In ROUTE header, Unknown route type (only type 0 is supported)
<i>ERR_CIPHER_- KEY_SIZE</i>	- For the cipher requested, the key size is incorrect
<i>ERR_AUTH_K- EY_SIZE</i>	- For the authentication requested, the key size is incorrect
<i>ERR_IV_SIZE</i>	- For the initialization vector requested, the size is incorrect
<i>ERR_SALT_SI- ZE</i>	- For the cipher salt requested, the size is incorrect
<i>ERR_ICV_SIZE</i>	- For the authentication requested, the ICV size is incorrect
<i>ERR_UNKNOW- N_NXTHDR</i>	- For the IP packet, unknown NH value
<i>ERR_FORMAT- _ERROR</i>	- Malformed packet
<i>WARN_INPUT_- QUEUE_FULL</i>	- Send queue is full
<i>WARN_OUTPU- T_QUEUE_EM- PTY</i>	- Receive queue is empty
<i>ERR_SA_NOT_- FOUND</i>	- Security Association not found
<i>ERR_NO_KEY</i>	- The key specified is invalid or not found
<i>ERR_KEY_EXP- IRED</i>	- The key specified has expired
<i>ERR_KEY_RE- VOKED</i>	- The key specified has been revoked
<i>ERR_KEY_AC- CESS</i>	- The key exists but is not readable by the calling process or cant be modified by the user
<i>ERR_KEY_NO- T_SUPP</i>	- The key type does not support reading of the payload data
<i>ERR_KEY_QU- OTA</i>	- The key quota for this user would be exceeded by creating this key
<i>ERR_KEY_INV- ALID</i>	- The payload data was invalid
<i>ERR_KEY_REJ- ECTED</i>	- The attempt to generate a new key was rejected
<i>ERR_KEY_NO- MEM</i>	- Insufficient memory to create a key
<i>ERR_KEY_INTR</i>	- The request was interrupted by a signal
<i>ERR_UNKNOW- N</i>	- Unknown error

Enumerator

ERR_NONE No error.

ERR_LATE Anti-replay late packet error found.

ERR_REPLAY Anti-replay error found.

ERR_ROLLOVER Packet number rollover error.

ERR_ROLLUNDER Packet number rollunder error.

ERR_PROGRAM Programming error.

ERR_ICV ICV check failure.

ERR_CRC CRC check failure.

ERR_READ_ENDOFILE Read end of file.

ERR_READ_FAILFILE Read failed to input file handle.

ERR_READ_BADFILE Read of missing file handle.

ERR_WRITE_FAILFILE Write failed to file output.

ERR_WRITE_BADFILE Write error to file output.

ERR_CHECKSUM Checksum error.

ERR_LISTEN TCP server received packet while listening without SYN=1.

ERR_ACKNUM TCP received ACKNUM is incorrect.

ERR_CLOSED TCP session CLOSED during TX/RX.

ERR_BITS TCP header reserved bits error.

ERR_TTL TTL field of packet is zero.

ERR_JUMBOGRAM_FORMAT IPv6 JUMBOGRAM formatting error.

WARN_DUMMY Dummy packet warning.

WARN_IPV6_ROUTE IPv6 Route header but not for this destination.

WARN_ZERO_DATA Payload had zero length data.

ERR_ICMP_HDR_EXT_LEN Header extension length is ODD for ROUTE header.

ERR_ICMP_SEGMENTS_LEFT In ROUTE header, Segments left is larger than N.

ERR_MULTICAST_EXT_HDR In ROUTE header, Multicast address present.

ERR_UNKNOWN_ROUTE_TYPE In ROUTE header, Unknown route type (only type 0 is supported)

ERR_CIPHER_KEY_SIZE Key size incorrect for cipher.

ERR_AUTH_KEY_SIZE Key size incorrect for authentication.

ERR_IV_SIZE Initialization vector size is incorrect.

ERR_SALT_SIZE Salt size for cipher is incorrect.

ERR_ICV_SIZE ICV size is incorrect for authentication scheme.

ERR_UNKNOWN_NXTHDR For the IP packet, unknown NH value.

ERR_FORMAT_ERROR Malformed packet.

WARN_INPUT_QUEUE_FULL Send queue is full.

WARN_OUTPUT_QUEUE_EMPTY Receive queue is empty.

ERR_SA_NOT_FOUND Security Association not found.

ERR_KEY_NOKEY The key specified is invalid or not found.

ERR_KEY_EXPIRED The key specified has expired.

ERR_KEY_REVOKED The key specified has been revoked.

ERR_KEY_ACCESS The key exists but is not readable by the calling process or can't be modified by the user.

ERR_KEY_NOT_SUPP The key type does not support reading of the payload data.

ERR_KEY_QUOTA The key quota for this user would be exceeded by creating this key.

ERR_KEY_INVALID The payload data was invalid.

ERR_KEY_REJECTED The attempt to generate a new key was rejected.

ERR_KEY_NOMEM Insufficient memory to create a key.

ERR_KEY_INTR The request was interrupted by a signal.

ERR_UNKNOWN Unknown error.

6.5.3.13 enum ProtocolPP::esn_id_t

IKEv2 ESN_ID fields

Parameters

<i>ESN_NO</i>	- Disable Extended Sequence Number Use
<i>ESN_YES</i>	- Enable Extended Sequence Number Use

Enumerator

ESN_NO Disable Extended Sequence Number Use.

ESN_YES Enable Extended Sequence Number Use.

6.5.3.14 enum ProtocolPP::exchg_t

IKEv2 Exchange Type fields

Parameters

<i>EXCHG_IKE_SA_INIT</i>	- IKE Security Association Initial Exchange
<i>EXCHG_IKE_AUTH</i>	- IKE Authentication Exchange
<i>EXCHG_CREATE_CHILD_SA</i>	- Create Child Security Association Exchange
<i>EXCHG_INFORMATIONAL</i>	- INFORMATIONAL Exchange
<i>EXCHG_IKE_SESSION_RESUME</i>	- IKE Session Resume Exchange
<i>EXCHG_GSA_AUTH</i>	- GSA Authentication Exchange
<i>EXCHG_GSA_REGISTRATION</i>	- GSA Registration Exchange
<i>EXCHG_GSA_REKEY</i>	- GSA Rekey Exchange
<i>EXCHG_IKE_PARSE</i>	- Parse the received packet
<i>EXCHG_IKE_RESEND</i>	- Resend the packet
<i>EXCHG_IKE_LISTEN</i>	- Set IKE in LISTEN mode on the socket

Enumerator

EXCHG_IKE_SA_INIT IKE Security Association Initial Exchange.

EXCHG_IKE_AUTH IKE Authentication Exchange.

EXCHG_CREATE_CHILD_SA Create Child Security Association Exchange.

EXCHG_INFORMATIONAL INFORMATIONAL Exchange.

EXCHG_IKE_SESSION_RESUME IKE Session Resume Exchange.

EXCHG_GSA_AUTH GSA Authentication Exchange.

EXCHG_GSA_REGISTRATION GSA Registration Exchange.

EXCHG_GSA_REKEY GSA Rekey Exchange.

EXCHG_IKE_PARSE Parse the received packet.

EXCHG_IKE_RESEND Resend the packet.

EXCHG_IKE_LISTEN Set IKE in LISTEN mode on the socket.

6.5.3.15 enum ProtocolPP::field_t

Parameters

<i>DIRECTION</i>	- Direction of processing (ENCAP or DECAP)
------------------	--

6.5.4 IKEv2 fields

Parameters

<i>IKECNXT</i>	- IKE Connection Name
<i>EXCHG</i>	- IKE Exchange type
<i>NXTPYLD</i>	- Next IKE payload type
<i>MAJOR_REVISION</i>	- IKE Major revision value
<i>MINOR_REVISION</i>	- IKE Minor revision value
<i>FLAGS</i>	- IKE Minor revision value
<i>MSGIDINIT</i>	- IKE Message Identifier when Initiator(starts at zero)
<i>MSGIDRESP</i>	- IKE Message Identifier when Responder(starts at zero)
<i>SPIi</i>	- IKE Initiator Security Parameter Index (SPI)
<i>SPIr</i>	- IKE Responder Security Parameter Index (SPI)
<i>Ni</i>	- IKE Initiator Nonce
<i>Nr</i>	- IKE Responder Nonce
<i>SKd</i>	- IKE PRF Key for Child SA Key Generation
<i>CIPHER</i>	- IKE Encryption Cipher
<i>CKEYLEN</i>	- IKE Encryption Key Length
<i>IVLEN</i>	- IKE Initialization Vector (ICV) Length
<i>SALTLEN</i>	- IKE Salt Length
<i>SKei</i>	- IKE Initiator Encryption Key
<i>SKer</i>	- IKE Responder Encryption Key
<i>IV</i>	- IKE Initialization Vector
<i>SKsi</i>	- IKE Initiator Salt Value
<i>SKsr</i>	- IKE Responder Salt Value
<i>INTEG</i>	- IKE Integrity Algorithm
<i>AKEYLEN</i>	- IKE Integrity Key Length
<i>ICVLEN</i>	- IKE Integrity Check Value (ICV)
<i>SKai</i>	- IKE Initiator Integrity Key
<i>SKar</i>	- IKE Responder Integrity Key
<i>PRF</i>	- IKE Psuedo Random Function (PRF)
<i>prflen</i>	- IKE Psuedo Random Function (PRF) output length
<i>SKpi</i>	- IKE Initiator Psuedo Random Key
<i>SKpr</i>	- IKE Responder Psuedo Random Key
<i>DH</i>	- IKE Diffie-Hellman Group ID
<i>IKEAUTH</i>	- IKE Signing Algorithm
<i>ARLEN</i>	- Length of the Anti-Replay Window

6.5.5 IPsec fields

Parameters

<i>VERSION</i>	- First nibble in the IP header defines IPv4 or IPv6
<i>MODE</i>	- Mode of operation either TUNNEL or TRANSPORT

<i>SPI</i>	- Security parameter index (SPI)
<i>SEQNUM</i>	- Sequence number for the packet from ESP header
<i>EXTSEQNUM</i>	- Extended sequence number for this packet
<i>ARLEN</i>	- Length of the anti-replay window
<i>ARWIN</i>	- Anti-Replay window represented as a byte array
<i>CIPHER</i>	- Cipher for encryption
<i>CKEYLEN</i>	- Length of the cipher key
<i>CIPHERKEY</i>	- Cipher key for encryption
<i>AUTH</i>	- Authentication mode
<i>AKEYLEN</i>	- Length of the authentication key
<i>AUTHKEY</i>	- Authentication key
<i>IVLEN</i>	- Length of the initialization vector in bytes (IV)
<i>IV</i>	- Initialization vector
<i>SALTLEN</i>	- Length of the salt for encryption use
<i>SALT</i>	- Salt for encryption use
<i>BYTECNT</i>	- Number of bytes encrypted or decrypted by this flow
<i>LIFETIME</i>	- Amount of time for this flow to exist
<i>SEQNUMOVRF-LW</i>	- Enable SEQNUM overflow during ENCAP
<i>STATEFULFRAG</i>	- Stateful FRAG bit
<i>BYPASSDF</i>	- Enable bypassing the DONT FRAGMENT bit
<i>BYPASSDSCP</i>	- Enable bypassing the DiffServices copy from inner to outer header
<i>NAT</i>	- Enable Network Address Translation (NAT) insertion
<i>NCHK</i>	- Enable NAT checksum generation and checking
<i>DSECN</i>	- Diff Services/Congestion bits
<i>TTLHOP</i>	- Time-to-Live field for the packet
<i>FLAGS</i>	- Flags field in IPv4
<i>NATSRC</i>	- Source port for NAT
<i>NATDST</i>	- Destination port for NAT
<i>ID</i>	- Identification field for IPv4
<i>LABEL</i>	- Label field for IPv6
<i>FRAGOOFFSET</i>	- Fragment offset from first packet
<i>MOREFRAG</i>	- Enable fragmentation of packet
<i>FRAGID</i>	- Fragmentation ID for this segment of the overall packet
<i>MTU</i>	- Maximum Transmission unit (MTU)
<i>SOURCE</i>	- Source address
<i>DESTINATION</i>	- Destination address
<i>EXTHDR</i>	- IPv6 Extension header
<i>NH</i>	- Next header in either the extension header or the payload
<i>ICVLEN</i>	- Length of the ICV
<i>HDRLEN</i>	- Length of the IP header
<i>TFCLEN</i>	- Length of the Traffic Flow Confidentiality (TFC) padding
<i>USEEXT</i>	- Use extended sequence number
<i>RANDIV</i>	- Use random IV rather than user IV
<i>NODEJUMBO</i>	- Nodal JUMBOGRAM support
<i>AUDIT</i>	- Enable auditing of IP flow
<i>AUDITLOG</i>	- Path to the audit log
<i>CHECKSUM</i>	- IPv4 header checksum
<i>LENGTH</i>	-Length of the packet(IPv4) or length of the packet minus the standard 40-byte header (IPv6)

6.5.6 ICMP fields

Parameters

<i>VERSION</i>	- Version (ICMP or ICMPv6)
<i>TYPE</i>	- ICMP Type
<i>CODE</i>	- ICMP SubType
<i>CHECKSUM</i>	- Checksum field
<i>POINTER</i>	- Pointer to error in packet
<i>IDENTIFIER</i>	- Identifier from the invoking message
<i>SEQNUM</i>	- Sequence number of the packet
<i>GATEWAY</i>	- Gateway Internet Address
<i>MTU</i>	- Maximum transmission unit for the next-hop link
<i>ORIGTIMESTAMP</i>	- Timestamp from the origination node
<i>MP</i>	
<i>RXTIMESTAMP</i>	- Receive timestamp
<i>TXTIMESTAMP</i>	- Transmit timestamp
<i>SOURCE</i>	- Source Address
<i>DESTINATION</i>	- Destination Address
<i>MESSAGE</i>	- Message to send
<i>ICMPCODE</i>	- Code to send

6.5.7 IP fields

Parameters

<i>VERSION</i>	- First nibble in the IP header defines IPv4 or IPv6
<i>DSECN</i>	- Diff Services/Congestion bits
<i>ID</i>	- Identification field for IPv4
<i>LABEL</i>	- Label field for IPv6
<i>FRAGOFFSET</i>	- Fragment offset from first packet
<i>NH</i>	- Next header in either the extension header or the payload
<i>TTLHOP</i>	- Time-to-Live field for the packet
<i>CHECKSUM</i>	- IPv4 header checksum
<i>FLAGS</i>	- Flags field in IPv4
<i>LENGTH</i>	-Length of the packet(IPv4) or length of the packet minus the standard 40-byte header (IPv6)
<i>EXTHDR</i>	- IPv6 Extension header
<i>NODEJUMBO</i>	- Nodal JUMBOGRAM support

6.5.8 LTE/3GPP fields

Parameters

<i>DATACTRL</i>	- Data or Control flag (Control - 0, Data - 1)
<i>PDUTYPE</i>	- Control PDU type field
<i>POLLBIT</i>	- Polling bit
<i>EXTENSION</i>	- Extension bit
<i>RSN</i>	- Reset sequence number (RSN)
<i>SNLEN</i>	- Sequence number length in bits
<i>HDREXT</i>	- Header extension type field
<i>LENGTHIND</i>	- Length indicator field
<i>SEQNUM</i>	- Sequence number
<i>HFNI</i>	- Hyper frame number indicator (HFNI)
<i>SUFI</i>	- Super Field (SUFI)
<i>FMS</i>	- PDCP SN of the first missing PDCP SDU (FMS)
<i>PGKINDEX</i>	- Five LSBs of PGK identity
<i>PTKIDENT</i>	- PTK identity
<i>SDUTYPE</i>	- PDCP SDU type (IP, ARP, PC5SIG, NONIP, RSVD)
<i>KDID</i>	- Kd identity
<i>NMP</i>	- Number of missing PDCP SDUs with associated COUNT value
<i>HRW</i>	- PDCP SN of the PDCP SDU received on WLAN with highest associated PDCP COUNT value
<i>BEARER</i>	- Bearer value
<i>FRESH</i>	- Fresh value
<i>BITMAPLEN</i>	- Length of the BitMap
<i>BITMAP</i>	- Bitmap for LTE

6.5.9 MacSEC fields

Parameters

<i>SOURCE</i>	- Source address
<i>DESTINATION</i>	- Destination address
<i>ETHERTYPE</i>	- Ethertype type
<i>SL</i>	- Short Length field
<i>TCIAN</i>	- TAG Control Information and Association Number
<i>PN</i>	- Packet number
<i>XPN</i>	- Extended packet number
<i>SCI</i>	- Secure Channel Identifier
<i>SSCI</i>	- Short Secure Channel Identifier
<i>SAKEY</i>	- Security Association Key
<i>ENRECEIVE</i>	- Enable reception of packets
<i>ENTRANSIT</i>	- Enable transmission of packets
<i>PROTECTFRA-MES</i>	- Protect outgoing packets
<i>INUSE</i>	- This MacSEC flow is in use and receiving packets
<i>CREATETIME</i>	- Time this MacSEC was created
<i>STARTTIME</i>	- Time this MacSEC last enabled reception of packets
<i>STOPTIME</i>	- Time this MacSEC last disabled reception of packets

6.5.10 SRTP fields

Parameters

<i>VERSION</i>	- Version of SRTP used (currently set to 2)
<i>PADDING</i>	- Padding flag to indicate padding in the payload
<i>EXTENSION</i>	- Flag to indicate presence of extension headers
<i>CC</i>	- Number of CSRC fields present
<i>MARKER</i>	- Marker flag
<i>PT</i>	- Type of payload present (much like NH in IPsec)
<i>SEQNUM</i>	- Sequence number for the packet
<i>ROC</i>	- Rollover counter
<i>TIMESTAMP</i>	- Timestamp for the packet
<i>SSRC</i>	- Synchronization Source Identifier
<i>CSRC</i>	- Contributing Source Identifier
<i>BLKSIZE</i>	- Block size of the encryption cipher

6.5.11 TCP fields

Parameters

<i>SOURCE</i>	- Source address
<i>DESTINATION</i>	- Destination address
<i>SEQNUM</i>	- Sequence number
<i>ACKNUM</i>	- Acknowledge number
<i>OFFSET</i>	- Data offset
<i>FLAGS</i>	- Flags field
<i>RCVWINDOW</i>	- Receive size window
<i>CHECKSUM</i>	- Checksum field
<i>URGENT</i>	- Urgent pointer
<i>STATE</i>	- TCP state of operation
<i>OPTIONS</i>	- Options field
<i>SEGLEN</i>	- Segment length
<i>PRECEDENCE</i>	- Precendence of this flow
<i>SNDUNA</i>	- Send UNA
<i>SNDNXT</i>	- Send next seqnum
<i>SNDWND</i>	- Send window size
<i>SNDUP</i>	- Send UP
<i>SNDW1</i>	- Send window 1
<i>SNDW2</i>	- Send window 2
<i>ISS</i>	- ISS
<i>RCVNXT</i>	- Receive next seqnum expected
<i>RCWWND</i>	- Receive window size
<i>RCVUP</i>	- RCVUP
<i>IRS</i>	- IRS

<i>TCPTIMEOUT</i>	- Timeout for TCP shutdown
-------------------	----------------------------

6.5.12 TLS fields

Parameters

<i>TYPE</i>	- Type of packet being sent/received
<i>VERSION</i>	- Version of TLS used
<i>LENGTH</i>	- Length of the packet
<i>SEQNUM</i>	- Current or recently received packet
<i>EPOCH</i>	- Current EPOCH in DTLS
<i>CIPHERSUITE</i>	- Cipher and authentication scheme for this flow
<i>ENCTHENMAC</i>	- Enable ENCRYPT-THEN-MAC for TLS
<i>IVEX</i>	- Implicit(false) or Explicit(true) IV

6.5.13 UDP fields

Parameters

<i>SOURCE</i>	- Source address
<i>DESTINATION</i>	- Destination address
<i>LENGTH</i>	- Length of the packet
<i>CHECKSUM</i>	- Checksum for UDP packet

6.5.14 Wifi/Wigig fields

Parameters

<i>FRAMECTL</i>	- Frame control sequence in entirety
<i>CTLEXT</i>	- Control frame extension
<i>PROTVER</i>	- Protocol version
<i>TYPE</i>	- Type
<i>SUBTYPE</i>	- SubType
<i>TDS</i>	- To Data Stream (TDS)
<i>FDS</i>	- From Data Stream (FDS)
<i>MFRAG</i>	- More fragmentation
<i>RETRY</i>	- retry bit
<i>PWRMGMT</i>	- Power management
<i>MDATA</i>	- More data field
<i>WEP</i>	- Wireless Encryption Protection (WEP)
<i>ORDER</i>	- Order
<i>KDFALG</i>	- Key derivation function algorithm (KDF)
<i>ID</i>	- Duration ID
<i>ADD1</i>	- First address
<i>ADD2</i>	- Second address
<i>ADD3</i>	- Third address

<i>SEQCTL</i>	- Sequence Control
<i>ADDR4</i>	- Fourth address
<i>QOSCTL</i>	- Quality of Service field
<i>HTCTL</i>	- HT control field
<i>PN</i>	- Packet Number
<i>EXTIV</i>	- Extended IV flag
<i>KEYID</i>	- Key identification
<i>HDRLEN</i>	- Length of the header according to option bits
<i>SPPCAP</i>	- SPPCAP
<i>FCS</i>	- Enable CRC checking for Wifi packet

6.5.15 WiMax fields

Parameters

<i>HT</i>	- Header type where zero for generic MAC header
<i>EC</i>	- Encryption Control 0=no encryption 1=encryption
<i>TYPE</i>	- Indicates the subheaders and payload types
<i>ESF</i>	- Extended subheader field ESF=1 then ESF present and follows MAC header
<i>CI</i>	- CRC indicator 0=no CRC 1=CRC present
<i>EKS</i>	- Encryption key sequence. Index of TEK and IV used to encrypt the payload
<i>LENGTH</i>	- length in bytes of MAC PDU including header and CRC if present
<i>CID</i>	- Connection Identifier
<i>FID</i>	- Flow Identifier (AGHM header only)
<i>EH</i>	- Extended header group indicator (AGHM header only)
<i>HCS</i>	- Header check sequence 8-bit header checksum
<i>PN</i>	- Packet Number

6.5.16 RSA CryptoSystem

Parameters

<i>BITSIZE</i>	- Size of the private key
<i>PRVKEY</i>	- Private key
<i>PUBKEY</i>	- Public key

6.5.17 Digital Signature Algorithm (DSA)

Parameters

<i>BITSIZE</i>	- Size of the private key
<i>PRIME</i>	- Prime value for DSA (usually called "p")
<i>SUBPRIME</i>	- Prime value for DSA (usually called "q")
<i>GENERATOR</i>	- Prime value for DSA (usually called "g")
<i>PRVKEY</i>	- Private key
<i>PUBKEY</i>	- Public key

6.5.18 Elliptic Curve Digital Signature Algorithm (ECDSA)

Parameters

<i>GX</i>	- X-coordinate for the ECDSA curve
<i>GY</i>	- Y-coordinate for the ECDSA curve
<i>CURVE</i>	- Name of the current curve
<i>PRVKEY</i>	- Private key
<i>PUBKEY</i>	- Public key

Enumerator

DIRECTION Direction of processing (ENCAP or DECAP)

VERSION Version field for the packet either IPV4 or IPV6 (ICMP or ICMPV6) (SSL30, TLS10, TLS11, TLS12, DTLS)

MODE Mode of operation either TUNNEL or TRANSPORT.

SPI Security parameter index (SPI)

SEQNUM Sequence number for the packet.

EXTSEQNUM Extended sequence number.

ARLEN Length of the anti-replay window.

ARWIN Anti-Replay window represented as a byte array.

CIPHER Cipher for encryption.

CKEYLEN Length of the cipher key.

CIPHERKEY Cipher key for encryption.

AUTH Authentication mode.

AKEYLEN Length of the authentication key.

AUTHKEY Authentication key.

IVLEN Length fo the initialization vector (IV)

IV Initializaton Vector (IV)

SALTLEN Length of encryption salt.

SALT Encryption salt.

BYTECNT Number of bytes encrypted or decrypted by this flow.

LIFETIME Amount of time for this flow to exist.

SEQNUMOVRFLOW Allow SEQNUM overflow during ENCAP.

STATEFULFRAG Stateful FRAG bit.

BYPASSDF Enable bypass of the DONT FRAGMENT bit copy from inner to outer header.

BYPASSDSCP Enable bypass of copy from inner to outer header of DiffServices.

NAT Enable Network Address Translation (NAT) insertion.

NCHK Enable NAT checksum generation and checking.

DSECN Bits for Diff Services/Congestion bits.

TTLHOP Time-to-Live field for the packet.

FLAGS Flags field in the IPv4.

NATSRC Source port for NAT.

NATDST Destination port for NAT.

ID Identification field for IPv4.

LABEL Label field for IPv6.

FRAGOFFSET Fragment offset from first packet.

MOREFRAG Enable fragmentation of packet.

FRAGID Fragmentation ID for this segment in the overall packet.

MTU Maximum Transmission unit (MTU)

SOURCE Source Address.

DESTINATION Destination Address.

EXTHDR IPv6 Extension header(s)

NH Next header in either the extension header or the payload.

ICVLEN Length of the ICV.

HDRLEN Length of the IP headeer.

TFCLEN Length of the Traffic flow Confidentiality (TFC) padding.

USEEXT Use extended sequence number.

RANDIV Use random IV rather than user IV.

NODEJUMBO Nodal JUMBOGRAM support.

AUDIT Enable audit of IP flow.

AUDITLOG Path to the audit log.

CHECKSUM Enable IPv4 header checksum.

LENGTH Length of the packet (IPv4) or length of the packet minus the standard IPv6 header.

TYPE ICMP Type.

CODE ICMP SubType.

POINTER Pointer to error in packet.

IDENTIFIER Identifier from the invoking message.

GATEWAY Gateway Internet Address.

ORIGTIMESTAMP Timestamp from the origination node.

RXTIMESTAMP Receive timestamp.

TXTIMESTAMP Transmit timestamp.

MESSAGE Message to send.

ICMPCODE Code to send.

DATACTRL Data or Control bit.

PDUTYPE Control PDU type field.

POLLBIT Polling bit.

EXTENSION Extension bit.

RSN Reset sequence number (RSN)

SNLEN Sequence number length in bits.

HDREXT Header extension type field.

LENGTHIND Length indicator field.

HFNI Hyper frame number indicator (HFNI)

SUFI Super Field (SUFI)

FMS PDCP SN of the first Missing PDCP SDU (FMS)

BITMAPLEN Length of BitMap.

BITMAP Bitmap for LTE.

PGKINDEX Five LSB(s) of PGK identity.

PTKIDENT PTK identity.

SDUTYPE PDCP SDU type (IP, ARP, PC5SIG, NONIP, RSVD)

KDID Kd identity.

NMP Number of missing PDCP SDU(s) with associated COUNT value.

HRW PDCP SN of the PDCP SDU received on WLAN with highest associated PDCP COUNT value.

BEARER Bearer value.

FRESH Fresh value for F9 modes.

ETHERTYPE Ethernet type.

SL Short length field.

TCIAN TAG Control Information and Association Number.

PN Packet number.

XPN Extended packet number.

SCI Secure Channel Identifier.

SSCI Short Secure Channel Identifier.

SAKEY Security Association Key.

ENRECEIVE Enable reception of packets.

ENTRANSMIT Enable transmission of packets.

PROTECTFRAMES Protect packets.

INUSE MacSEC flow is in use (RX or TX)

CREATETIME Time this MacSEC was created.

STARTTIME Time this MacSEC last enabled RX or TX.

STOPTIME Time this MacSEC last disabled RX or TX.

PADDING Padding field.

CC CSRC count field.

MARKER Marker field.

PT Payload type field.

ROC Rollover Counter (ROC)

TIMESTAMP Timestamp field.

SSRC Synchronization source identifier (SSRC)

CSRC Contributing source identifier (CSRC)

BLKSIZE Block size of the encryption cipher.

MKI MKI flag.

MKILEN MKI data length.

MKIDATA MKI data.

ACKNUM Acknowledge number.

OFFSET Data offset.

RCVWINDOW Receive size window.

URGENT Urgent pointer.

STATE TCP state of operation.

OPTIONS OPTIONS.

SEGLEN SEGLEN.

PRECEDENCE PRECEDENCE.

SNDUNA SNDUNA.

SNDNXT SNDNXT.

SNDWND SNDWND.

SNDUP SNDUP.

SNDW1 SNDW1.

SNDW2 SNDW2.

ISS ISS.

RCVNXT RCVNXT.

RCVWND RCVWND.

RCVUP RCVUP.

IRS IRS.

TCPTIMEOUT TCPTIMEOUT.

EPOCH Epoch number for DTLS.

CIPHERSUITE Ciphersuite for the TLS protocol.

ENCTHENMAC Encrypt-then-MAC the TLS packet.

IVEX Implicit or Explicit IV for TLS.

FRAMECTL Frame control sequence in entirety.

CTLEXT Control Frame extension.

PROTVER Protocol version.

SUBTYPE SubType.

TDS To Data Stream.

FDS From Data Stream.

MFRAG More fragmentation.

RETRY Retry.

PWRMGMT Power management.

MDATA More data field.

WEP Wireless Encryption Protection (WEP)

ORDER Order.

KDFALG Key derivation function (KDF) algorithm.

ADDR1 First address.

ADDR2 Second address.

ADDR3 Third address.

SEQCTL Sequence control.

ADDR4 Fourth address.

QOSCTL Quality of Service field.

HTCTL HT control field.

EXTIV Extended IV flag.

KEYID Key identification.

SPPCAP SPP Cap.

FCS Enable CRC checking for packet.

HT Header Type where zero for generic MAC header.

EC Encryption Control 0=no encryption 1=encryption.

ESF Extended Subheader Field ESF=1 then ESF is present and follows MAC header.

CI CRC Indicator 0=no CRC, 1=CRC present.

EKS Encryption Key Sequence. Index of TEK and IV used to encrypt the payload.

CID Connection Identifier.

FID Flow Identifier (AGHM header only)

EH Extended Header Group Indicator (AGHM header only)

HCS Header Check Sequence 8-bit header checksum.

IKECNXT IKE Conneciton Name.

EXCHG IKE Exchange Type.

NXTPYLD IKE Next Payload Value.

MAJOR_VERSION IKE Major Revision Number.

MINOR_VERSION IKE Minor Revision Number.

MSGIDINIT IKE Message Identifier when Initiator.

MSGIDRESP IKE Message Identifier when Responder.

SPIi IKE Initiator Security Parameter Index.

SPIr IKE Responder Security Parameter Index.

Ni IKE Initiator Nonce.

Nr IKE Responder Nonce.

SKd IKE Child SA Key Generation Seed.

SKei IKE Initiator Encryption Key.

SKer IKE Responder Encryption Key.

SKsi IKE Initiator Salt Value.

SKsr IKE Responder Salt Value.

INTEG IKE Integrity Algorithm.

SKai IKE Initiator Integrity Algorithm Key.

SKar IKE Responder Integrity Algorithm Key.

PRF IKE Psuedo Random Function Algorithm.

PRFLEN IKE Psuedo Random Total Key Length.

SKpi IKE Initiator Psuedo Random Key.

SKpr IKE Responder Psuedo Random Key.

DH IKE Diffie-Hellman Group ID.

IKEAUTH IKE Signature Algorithm.

BITSIZE Size fo the Private Key.

PRVKEY Private Key.

PUBKEY Public Key.

PRIME Prime value for DSA (usually called "p")

SUBPRIME Subprime value for DSA (usually called "q")

GENERATOR Generator value for DSA (usually called "g")

GX X-coordinate for the ECDSA curve.

GY Y-coordinate for the ECDSA curve.

CURVE Name of the current curve.

6.5.18.1 enum ProtocolPP::iana_t

IANA values for Internet communication Value and their definitions can be found at www.iana.org

Enumerator

HOPOPT IPv6 Hop-by-Hop Option - 0.

ICMP Internet Control Message Protocol (ICMP) version 4 - 1.

IGMP Internet Group Management Protocol (IGMP) - 2.

GGP Gateway-to-Gateway Protocol (GGP) - 3.

IPV4 IP version 4 - 4.

ST Internet Stream Protocol (ST) - 5.

TCP Transmission Control Protocol (TCP) - 6.

CBT Core-base trees (CBT) - 7.

EGP Exterior Gateway Protocol (EGP) - 8.

IGP Interior Gateway Protocol (IGP) - 9.

BBN_RCC_MON BBN RCC Monitoring (BBN_RCC_MON) - 10.

NVP_II Network Voice Protocol (NVP_II) - 11.

PUP Xerox PUP (PUP) - 12.

ARGUS ARGUS - 13.

EMCON EMCON - 14.

XNET Cross Net Debugger (XNET) - 15.

CHAOS Chaos - 16.

UDP User Datagram Protocol (UDP) - 17.

MUX Multiplexing (MUX) - 18.

DCN_MEAS DCN Measurement Subsystems (DCN-MEAS) - 19.

HMP Host Monitoring Protocol (HMP) - 20.

PRM Packet Radio Measurement (PRM) - 21.

XNS_IDP XEROX NS IDP (XNS-IDP) - 22.

TRUNK_1 Trunk-1 - 23.

TRUNK_2 Trunk-2 - 24.

LEAF_1 Leaf-1 - 25.

LEAF_2 Leaf-2 - 26.

RDP Reliable Data Protocol (RDP) - 27.

IRTP Internet Reliable Transaction Protocol (IRTP) - 28.

ISO_TP4 ISO Transport Protocol Class 4 (ISO-TP4) - 29.

NETBLT Bulk Data Transfer Protocol (NETBLT) - 30.

MFE_NSP MFE Network Services Protocol (MFE-NSP) - 31.

MERIT_INP MERIT Internodal Protocol (MERIT-INP) - 32.

DCCP Datagram Congestion Control Protocol (DCCP) - 33.

THREE_PC Third Party Connect Protocol (3PC) - 34.

IDPR Inter-Domain Policy Routing Protocol (IDPR) - 35.

XTP Xpress Transport Protocol (XTP) - 36.

DDP Datagram Delivery Protocol (DDP) - 37.

IDPR_CMTP IDPR Control Message Transport Protocol (IDPR-CMTP) - 38.

TP_PLUS TP++ Transport Protocol (TP++) - 39.

IL IL Transport Protocol (IL) - 40.

IPV6 IP version 6 - 41.

SDRP Source Demand Routing Protocol (SDRP) - 42.

IPV6_ROUTE IPv6 Routing header - 43.

IPV6_FRAG IPv6 Fragmentation header - 44.

IDRP Inter-Domain Routing Protocol (IDRP) - 45.

RSVP Resource Reservation Protocol (RSVP) - 46.

GRE Generic Routing Encapsulation (GRE) - 47.

DSR Dynamic Source Routing Protocol (DSR) - 48.

BNA Burroughs Network Architecture (BNA) - 49.

ESP Encapsulating Security Payload - 50.

AH Authentication header - 51.

I_NLSP Integrated Net Layer Security Protocol (I-NLSP) - 52.

SWIPE SwIPE (IP with Encryption) - 53.

NARP NBMA Address Resolution Protocol (NARP) - 54.

MOBILE IP Mobility (Min Encap) - 55.

TLSP Transport Layer Security Protocol (using Kryptonet key management) - 56.

SKIP Simple Key-Management for Internet Protocol (SKIP) - 57.

ICMPV6 ICMP version 6 - 58.

IPV6_NONXT IPv6 No Next header - 59.

IPV6_OPTS IPv6 Options header - 60.

HOST_INT_PROT Any host Internet Protocol - 61.

CFTP CFTP - 62.

LOCAL_NET Any local network - 63.

SAT_EXPAK SATNET and Backroom EXPAK (SAT-EXPAK) - 64.

KRYPTOLAN Kryptolan - 65.

RVD MIT Remote Virtual Disk Protocol (RVD) - 66.

IPPC Internet Pluribus Packet Core (IPPC) - 67.

DFS Any distributed file system - 68.

SAT_MON SATNET Monitoring (SAT-MON) - 69.

VISA VISA Protocol - 70.

IPCV Internet Packet Core Utility (IPCV) - 71.

CPNX Computer Protocol Network Executive (CPNX) - 72.

CPHB Computer Protocol Heart Beat (CPHB) - 73.

WSN Wang Span Network (WSN) - 74.

PVP Packet Video Protocol (PVP) - 75.

BR_SAT_MON Backroom SATNET Monitoring (BR-SAT-MON) - 76.

SUN_ND SUN ND Protocol-Temporary (SUN-ND) - 77.

WB_MON WIDEBAND Monitoring (WB-MON) - 78.

WB_EXPAK WIDEBAND EXPAK (WB-EXPAK) - 79.

ISO_IP International Organization for Standardization Internet Protocol (ISO-IP) - 80.

VMTP Versatile Message Transaction Protocol (VMTP) - 81.

SECURE_VMTP Secure Versatile Message Transaction Protocol (SECURE-VMTP) - 82.

VINES VINES - 83.

TTP TTP - 84.

IPTM Internet Protocol Traffic Manager (IPTM) - 85.

NSFNET_IGP NSFNET-IGP - 86.

DGP Dissimilar Gateway Protocol (DGP) - 87.

TCF TCF - 88.

EIGRP EIGRP - 89.

OSPFIGP Open Shortest Path First (OSPFIGP) - 90.

SPRITE_RPC Sprite RPC Protocol (Sprite-RPC) - 91.

LARP Locus Address Resolution Protocol (LARP) - 92.

MTP Multicast Transport Protocol (MTP) - 93.

AX_25 AX.25 - 94.

MICP Mobile Internetworking Control Protocol (MICP) - 95.

SCC_SP Semaphore Communications Security Protocol (SCC-SP) - 96.

ETHERIP Ethernet-within-IP Encapsulation (ETHERIP) - 97.

IENCAP Encapsulation Header (ENCAP) - 98.

PRIV_ENCRYPT Any private encryption scheme - 99.

GMTP GMTP - 100.

IFMP Ipsilon Flow Management Protocol (IFMP) - 101.

PNNI PNNI over IP (PNNI) - 102.

PIM Protocol Independent Multicast (PIM) - 103.

ARIS IBM's ARIS (Aggregate Route IP Switching) Protocol (SCPS) - 104.

SCPS SCPS (Space Communications Protocol Standards) - 105.

QNX QNX - 106.

A_N Active Networks (A/N) - 107.

IPCOMP IP Payload Compression Protocol (IPComp) - 108.

SNP Sitara Networks Protocol (SNP) - 109.

COMPAQ_PEER Compaq Peer Protocol (Compaq-Peer) - 110.

IPX_IN_IP IPX in IP (IPX-in-IP) - 111.

VRRP Virtual Router Redundancy Protocol, Common Address Redundancy Protocol (not IANA assigned) - 112.

PGM PGM Reliable Transport Protocol (PGM) - 113.

ZERO_HOP_PROT Any 0-hop protocol - 114.

L2TP Layer Two Tunneling Protocol Version 3 (L2TP) - 115.

DDX D-II Data Exchange (DDX) - 116.

IATP Interactive Agent Transfer Protocol (IATP) - 117.

STP Schedule Transfer Protocol (STP) - 118.

SRP SpectraLink Radio Protocol (SRP) - 119.

UTI Universal Transport Interface Protocol (UTI) - 120.

SMP Simple Message Protocol (SMP) - 121.

SM Simple Multicast Protocol (SM) - 122.

PTP Performance Transparency Protocol (PTP) - 123.

ISIS_OVER_IPV4 Intermediate System to Intermediate System (IS-IS) Protocol over IPv4 - 124.

FIRE Flexible Intra-AS Routing Environment (FIRE) - 125.

CRTDP Combat Radio Transport Protocol (CRTDP) - 126.

CRUDP Combat Radio User Datagram (CRUDP) - 127.

SSCOPMCE Service-Specific Connection-Oriented Protocol in a Multilink and Connectionless Environment (SSCOPMCE) - 128.

IPLT IPLT - 129.

SPS Secure Packet Shield (SPS) - 130.

PIPE Private IP Encapsulation within IP (PIPE) - 131.

SCTP Stream Control Transmission Protocol (SCTP) - 132.

FC Fibre Channel (FC) - 133.

RSVP_E2E_IGNORE Reservation Protocol (RSVP) End-to-End Ignore - 134.

MOBILITY_HEADER Mobility Extension Header for IPv6 - 135.

UDPLITE UDP-Lite Protocol - 136.

MPLS_IN_IP Multiple Label Switching Encapsulated in IP (MPLS-in-IP) - 137.

MANET MANET Protocols (MANET) - 138.

HIP Host Identity Protocol (HIP) - 139.

SHIM6 Site Multihoming by IPv6 Intermediation (Shim6) - 140.

WESP Wrapped Encapsulating Security Protocol (WESP) - 141.

ROHC Robust Header Compression (ROHC) - 142.

JUMBOGRAM JUMBOGRAM extension header - 194.

6.5.18.2 enum ProtocolPP::icmpcode_t

ICMP Codes

Parameters

ECHORPLY	- Echo Reply
NONETWRK	- Destination network unreachable
NOHOST	- Destination host unreachable
NOPROT	- Destination protocol unreachable
NOPORT	- Destination port unreachable
FRAGRQD	- Fragmentation required and DF flag set
RTEFAIL	- Source route failed
DSTUNKWN	- Destination network unknown
DSTHSTUNKN-WN	- Destination host unknown
SRCHSTISOLT	- Source host isolated
NETWRKPROH-IB	- Network administratively prohibited
HOSTPROHIB	- Host administratively prohibited
NETWKNOTOS	- Network unreachable for ToS
COMMPROHIB	- Communication administratively prohibited
HOSTVILATE	- Host Precedence Violation
CUTOFF	- Precedence cutoff in effect
REDIRNETWK	- Redirect Datagram for the Network
REDIRHOST	- Redirect Datagram for the Host
REDIRTOSN	- Redirect Datagram for the ToS and network
REDIRTOSH	- Redirect Datagram for the ToS and host
TTLEXPIRE	- TTL expired in transit
FRAGEXPIRE	- Fragment reassembly time exceeded
PTRINDERR	- Pointer indicates the error
OPTERR	- Missing a required option
BADLENGTH	- Bad length
NOROUTE	- No destination (ICMPv6)
COMMPROHIB-V6	- Communication with destination administratively prohibited (ICMPv6)
BYNDSRCADD-R	- Beyond scope of source address (ICMPv6)
ADDRUNREAC-H	- Address unreachable (ICMPv6)
PORTUNREAC-H	- Port unreachable (ICMPv6)
SRCADDRPLCY	- Source address failed ingree/egress policy (ICMPv6)
REJECTDST	- Reject route to destination (ICMPv6)
HOPLMTEXCD	- Hop limit exceeded in transit (ICMPv6)
FRAGLTIME	- Fragment reassembly time exceeded (ICMPv6)
ERRHDRFIELD	- Erroneous header field encountered (ICMPv6)
NXTHDRERR	- Unrecognized Next Header type encountered (ICMPv6)
IPV6OPTERR	- Unrecognized IPv6 option encountered (ICMPv6)
SEQNUMRST	- Sequence number reset (ICMPv6) (ICMPv6)

Enumerator

ECHORPLY ECHO REPLY - 0.
NONETWRK DESTINATION NETWORK UNREACHABLE - 0.
NOHOST DESTINATION HOST UNREACHABLE - 1.
NOPROT DESTINATION PROTOCOL UNREACHABLE - 2.
NOPORT DESTINATION PORT UNREACHABLE - 3.
FRAGRQD FRAGMENTATION REQUIRED AND DF FLAG SET - 4.
RTEFAIL SOURCE ROUTE FAILED - 5.

DSTUNKWN DESTINATION NETWORK UNKNOWN - 6.

DSTHSTUNKWN DESTINATION HOST UNKNOWN - 7.

SRCHSTISOLT SOURCE HOST ISOLATED - 8.

NETWKPROHIB NETWORK ADMINISTRATIVELY PROHIBITED - 9.

HOSTPROHIB HOST ADMINISTRATIVELY PROHIBITED - 10.

NETWKNOTOS NETWORK UNREACHABLE FOR ToS - 11.

HOSTNOTOS HOST UNREACHABLE FOR ToS - 12.

COMMPROHIB COMMUNICATION ADMINISTRATIVELY PROHIBITED - 13.

HOSTVILATE HOST PRECEDENCE VIOLATION - 14.

CUTOFF PRECEDENCE CUTOFF IN EFFECT - 15.

REDIRNETWK REDIRECT DATAGRAM FOR THE NETWORK - 0.

REDIRHOST REDIRECT DATAGRAM FOR THE HOST - 1.

REDIRTOSN REDIRECT DATAGRAM FOR THE ToS AND NETWORK - 2.

REDIRTOSH REDIRECT DATAGRAM FOR THE ToS AND HOST - 3.

TTLEXPIRE TTL EXPIRED IN TX - 0.

FRAGEXPIRE FRAGMENT REASSEMBLY TIME EXPIRED - 1.

PTRINDERR POINTER INDICATES THE ERROR - 0.

OPTERR MISSING A REQUIRED OPTION - 1.

BADLENGTH BAD LENGTH - 2.

NOROUTE NO DESTINATION (ICMPv6) - 0.

COMMPROHIBV6 COMMUNICATION WITH DEST ADMINISTRATIVELY PROHIBITED (ICMPv6) - 1.

BYNDSRCADDR BEYOND SCOPE OF SOURCE ADDR (ICMPv6) - 2.

ADDRUNREACH ADDRESS UNREADCHABLE (ICMPv6) - 3.

PORTUNREACH PORT UNREACHABLE (ICMPv6) - 4.

SRCADDRPLCY SOURCE ADDRESS FAILED POLICY (ICMPv6) - 5.

REJECTDST REJECT ROUTE TO DESTINATION (ICMPv6) - 6.

HOPLMTEXCD HOP LIMIT EXCEEDED IN TX (ICMPv6) - 0.

FRAGTIME FRAGMENT REASSEMBLY EXCEEDED TIME (ICMPv6) - 1.

ERRHDRFIELD ERRORONEOUS HEADER FIELD (ICMPv6) - 0.

NXTHDRERR UNRECOGNIZED NEXT HEADER (ICMPv6) - 1.

IPV6OPTERR UNRECOGNIZED IPv6 OPTION (ICMPv6) - 2.

SEQNUMRST SEQUENCE NUMBER RESET (ICMPv6) - 255.

6.5.18.3 enum ProtocolPP::icmpmsg_t

ICMP Message Types

Parameters

ECHORPLYMS-G	- Echo Reply
RSVD1	- Reserved1
RSVD2	- Reserved2

<i>DESTUNRCH</i>	- Destination Unreachable
<i>SRCQNCH</i>	- Source Quench
<i>RDIRMSG</i>	- Redirect Message
<i>ALTHOST</i>	- Alternate Host Address
<i>RSVD</i>	- Reserved7
<i>ECHORQST</i>	- Echo Request
<i>RTRADVERT</i>	- Router Advertisement
<i>RTRSOLCIT</i>	- Router Solicitation
<i>TIMEXCEED</i>	- Time Exceeded
<i>BADIPHDR</i>	- Parameter Problem : Bad IP header
<i>TIMESTMP</i>	- Timestamp
<i>TIMERPLY</i>	- Timestamp Reply
<i>INFORQST</i>	- Information Request
<i>INFORPLY</i>	- Information Reply
<i>ADMSKRQST</i>	- Address Mask Request
<i>ADMSKRPLY</i>	- Address Mask Reply
<i>TRACERTE</i>	- Traceroute (ICMPv6)
<i>NODEST</i>	- Destination Unreachable (ICMPv6)
<i>PKTBIG</i>	- Packet Too Big (ICMPv6)
<i>TIMEOUT</i>	- Time Exceeded (ICMPv6)
<i>PARAM</i>	- Parameter Problem (ICMPv6)
<i>PRVTE1</i>	- Private Experimentation (ICMPv6)
<i>PRVTE2</i>	- Private Experimentation (ICMPv6)
<i>RSVDE</i>	- Reserved for expansion of ICMPv6 error messages (ICMPv6)
<i>ECHORQSTV6</i>	- Echo Request (ICMPv6)
<i>ECHORPLYV6</i>	- Echo Reply (ICMPv6)
<i>PRVTI1</i>	- Private Experimentation (ICMPv6)
<i>PRVTI2</i>	- Private Experimentation (ICMPv6)
<i>RSVDI</i>	- Reserved for expansion of ICMPv6 info messages (ICMPv6)

Enumerator

ECHORPLYMSG ECHO REPLY - 0.**RSVD1** RESERVED - 1.**RSVD2** RESERVED - 2.**DESTUNRCH** DESTINATION UNREACHABLE - 3.**SRCQNCH** SOURCE QUENCH - 4.**RDIRMSG** REDIRECT MESSAGE - 5.**ALTHOST** ALTERNATE HOST ADDRESS - 6.**RSVD** RESERVED - 7.**ECHORQST** ECHO REQUEST - 8.**RTRADVERT** ROUTER ADVERTISEMENT - 9.**RTRSOLCIT** ROUTER SOLICITATION - 10.**TIMEXCEED** TIME EXCEEDED - 11.**BADIPHDR** PARAMETER PROBLEM : BAD IP HEADER - 12.**TIMESTMP** TIMESTAMP - 13.**TIMERPLY** TIMESTAMP REPLY - 14.**INFORQST** INFORMATION REQUEST - 15.**INFORPLY** INFORMATION REPLY - 16.**ADMSKRQST** ADDRESS MASK REQUEST - 17.**ADMSKRPLY** ADDRESS MASK REPLY - 18.**TRACERTE** TRACEROUTE - 30.

NODEST DESTINATION UNREACHABLE (ICMPv6) - 1.
PKTBIG PACKET TOO BIG (ICMPv6) - 2.
TIMEOUT TIME EXCEEDED (ICMPv6) - 3.
PARAM PARAMETER PROBLEM (ICMPv6) - 4.
PRVTE1 PRIVATE EXPERIMENTATION (ICMPv6) - 100.
PRVTE2 PRIVATE EXPERIMENTATION (ICMPv6) - 101.
RSVDE RESERVED (ICMPv6) - 127.
ECHORQSTV6 ECHO REQUEST (ICMPv6) - 128.
ECHORPLYV6 ECHO REPLY (ICMPv6) - 129.
PRVTI1 PRIVATE EXPERIMENTATION (ICMPv6) - 200.
PRVTI2 PRIVATE EXPERIMENTATION (ICMPv6) - 201.
RSVDI RESERVED (ICMPv6) - 255.

6.5.18.4 enum ProtocolPP::id_type_t

IKEv2 ID_TYPE fields

Parameters

<i>ID_IPV4_ADDR</i>	- IPv4 Address ID type
<i>ID_FQDN</i>	- FQDN ID type
<i>ID_RFC822_ADDR_DR</i>	- RFC822 Address ID type
<i>ID_UNASSIGNED_ED4</i>	- Unassigned
<i>ID_IPV6_ADDR</i>	- IPv6 Address ID type
<i>ID_UNASSIGNED_ED6</i>	- Unassigned
<i>ID_UNASSIGNED_ED7</i>	- Unassigned
<i>ID_UNASSIGNED_ED8</i>	- Unassigned
<i>ID_DER ASN1_DN</i>	- ASN1 DN ID type
<i>ID_DER ASN1_GN</i>	- ASN1 GN ID type
<i>ID_KEY_ID</i>	- Key ID type
<i>ID_FC_NAME</i>	- FC Name ID type
<i>ID_NULL</i>	- NULL ID type

Enumerator

ID_IPV4_ADDR IPv4 Address ID type.
ID_FQDN FQDN ID type.
ID_RFC822_ADDR RFC822 Address ID type.
ID_UNASSIGNED4 Unassigned.
ID_IPV6_ADDR IPv6 Address ID type.
ID_UNASSIGNED6 Unassigned.
ID_UNASSIGNED7 Unassigned.
ID_UNASSIGNED8 Unassigned.
ID_DER ASN1_DN ASN1 DN ID type.

ID_DER ASN1 GN ASN1 GN ID type.
ID_KEY_ID Key ID type.
ID_FC_NAME FC Name ID type.
ID_NULL NULL ID type.

6.5.18.5 enum ProtocolPP::ike_gateway_t

IKE Gateway Identity Types

Parameters

GATEWAY_IP-V4_ADDR	- IPv4 Address of the VPN gateway
GATEWAY_IP-V6_ADDR	- IPv6 Address of the VPN gateway
GATEWAY_FQ-DN_ADDR	- FQDN Address of the VPN gateway

Enumerator

GATEWAY_IPV4_VPN_ADDR IPv4 Address of the VPN gateway.
GATEWAY_IPV6_VPN_ADDR IPv6 Address of the VPN gateway.
GATEWAY_FQDN_VPN_ADDR FQDN Address of the VPN gateway.

6.5.18.6 enum ProtocolPP::ike_hash_t

IKE Hash Algorithms

Parameters

HASH_SHA1	- SHA1 Hash Algorithm
HASH_SHA2_-256	- SHA2_256 Hash Algorithm
HASH_SHA2_-384	- SHA2_384 Hash Algorithm
HASH_SHA2_-512	- SHA2_512 Hash Algorithm
HASH_IDENTITY	- Identity Hash Algorithm

Enumerator

HASH_SHA1 SHA1 Hash Algorithm.
HASH_SHA2_256 SHA2_256 Hash Algorithm.
HASH_SHA2_384 SHA2_384 Hash Algorithm.
HASH_SHA2_512 SHA2_512 Hash Algorithm.
HASH_IDENTITY Identity Hash Algorithm.

6.5.18.7 enum ProtocolPP::ike_ipcomp_t

IKE Compression Types

Parameters

<i>IPCOMP_OUI</i>	- OUI Compression
<i>IPCOMP_DEFLATE</i>	- Deflation
<i>IPCOMP_LZS</i>	- LZS Compression
<i>IPCOMP_LZJH</i>	- LZJH Compression

Enumerator

- IPCOMP_OUI*** Configure an internal IPv4 Address.
IPCOMP_DEFLATE Configure an internal IPv4 NETMASK.
IPCOMP_LZS Configure a IPv4 DNS setting.
IPCOMP_LZJH Configure a IPv4 NBNS setting.

6.5.18.8 enum ProtocolPP::ike_pyld_t

IKEv2 Payload Type fields

Parameters

<i>PYLD_NONE</i>	- No Next Payload
<i>PYLD_SA</i>	- Security Association
<i>PYLD_KE</i>	- Key Exchange
<i>PYLD_IDI</i>	- Identification Initiator
<i>PYLD_IDR</i>	- Identification Responder
<i>PYLD_CERT</i>	- Certificate
<i>PYLD_CERTR_EQ</i>	- Certificate Request
<i>PYLD_AUTH</i>	- Authentication
<i>PYLD_NINR</i>	- Nonce
<i>PYLD_N</i>	- Notify
<i>PYLD_D</i>	- Delete
<i>PYLD_V</i>	- Vendor ID
<i>PYLD_TSI</i>	- Traffic Selector Initiator
<i>PYLD_TSR</i>	- Traffic Selector Responder
<i>PYLD_SK</i>	- Encrypted and Authenticated
<i>PYLD_CP</i>	- Configuration
<i>PYLD_EAP</i>	- Extensible Authentication
<i>PYLD_GSPM</i>	- Generic Secure Password Method
<i>PYLD_IDG</i>	- Group Identification
<i>PYLD_GSA</i>	- Group Security Association
<i>PYLD_KD</i>	- Key Download
<i>PYLD_SKF</i>	- Encrypted and Authenticated Fragment
<i>PYLD_PS</i>	- Puzzle Solution

Enumerator

- PYLD_NONE*** No Next Payload.
PYLD_SA Security Association.
PYLD_KE Key Exchange.
PYLD_IDI Identification Initiator.
PYLD_IDR Identification Responder.
PYLD_CERT Certificate.
PYLD_CERTREQ Certificate Request.

PYLD_AUTH Authentication.
PYLD_NINRNonce.
PYLD_N Notify.
PYLD_D Delete.
PYLD_V Vendor ID.
PYLD_TSI Traffic Selector Initiator.
PYLD_TSRR Traffic Selector Responder.
PYLD_SK Encrypted and Authenticated.
PYLD_CP Configuration.
PYLD_EAP Extensible Authentication.
PYLD_GSPM Generic Secure Password Method.
PYLD_IDG Group Identification.
PYLD_GSA Group Security Association.
PYLD_KD Key Download.
PYLD_SKF Encrypted and Authenticated Fragment.
PYLD_PS Puzzle Solution.

6.5.18.9 enum ProtocolPP::ike_secpass_t

IKE Secure Password Types

Parameters

SECURE_PAS-SWORD_PACE	- PACE Secure Password
SECURE_PAS-SWORD_AUGP-AKE	- AugPAKE Secure Password
SECURE_PAS-SWORD_PSK-AUTH	- Secure PSK Authentication

Enumerator

SECURE_PASSWORD_PACE PACE Secure Password.
SECURE_PASSWORD_AUGPAKE AugPAKE Secure Password.
SECURE_PASSWORD_PSK_AUTH Secure PSK Authentication.

6.5.18.10 enum ProtocolPP::ike_ts_t

IKE Traffic Selector Types

Parameters

TS_IPV4_ADD-R_RANGE	- Traffic Selector with IPv4 Address Range
TS_IPV6_ADD-R_RANGE	- Traffic Selector with IPv6 Address Range
TS_FC_ADDR-RANGE	- Traffic Selector with FC Address Range

Enumerator

TS_IPV4_ADDR_RANGE Traffic Selector with IPv4 Address Range.

TS_IPV6_ADDR_RANGE Traffic Selector with IPv6 Address Range.

TS_FC_ADDR_RANGE Traffic Selector with FC Address Range.

6.5.18.11 enum ProtocolPP::integ_id_t

IKEv2 INTEG_ID fields

Parameters

AUTH_NONE	- No Authentication
AUTH_HMAC_MD5_96	- HMAC-MD5-96 Authentication with 96-bit ICV
AUTH_HMAC_SHA1_96	- HMAC-SHA1-96 Authentication with 96-bit ICV
AUTH_DES_MAC	- DES-MAC Authentication with 64-bit ICV
AUTH_KPDK_MD5	- KPDK_MD5 Authentication with 128-bit ICV
AUTH_AES_XCBC_96	- AES-XCBC-96 Authentication with 96-bit ICV
AUTH_HMAC_MD5_128	- HMAC-MD5-128 Authentication with 128-bit ICV
AUTH_HMAC_SHA1_160	- HMAC-SHA1-160 Authentication with 160-bit ICV
AUTH_AES_CMAC_95	- AES-CMAC-96 Authentication with 96-bit ICV
AUTH_AES_128_GMAC	- AES-128-GMAC Authentication with 128-bit ICV
AUTH_AES_192_GMAC	- AES-192-GMAC Authentication with 192-bit ICV
AUTH_AES_256_GMAC	- AES-256-GMAC Authentication with 256-bit ICV
AUTH_HMAC_SHA2_256_128	- HMAC-SHA2-256 Authentication with 128-bit ICV
AUTH_HMAC_SHA2_384_192	- HMAC-SHA2-384 Authentication with 192-bit ICV
AUTH_HMAC_SHA2_512_256	- HMAC-SHA2-512 Authentication with 256-bit ICV
AUTH_NONE	- No Authentication

Enumerator

AUTH_NONE No Authentication.

AUTH_HMAC_MD5_96 HMAC-MD5-96 Authentication with 96-bit ICV.

AUTH_HMAC_SHA1_96 HMAC-SHA1-96 Authentication with 96-bit ICV.

AUTH_DES_MAC DES-MAC Authentication with 64-bit ICV.

AUTH_KPDK_MD5 KPDK_MD5 Authentication with 128-bit ICV.

AUTH_AES_XCBC_96 AES-XCBC-96 Authentication with 96-bit ICV.

AUTH_HMAC_MD5_128 HMAC-MD5-128 Authentication with 128-bit ICV.

AUTH_HMAC_SHA1_160 HMAC-SHA1-160 Authentication with 160-bit ICV.

AUTH_AES_CMAC_96 AES-CMAC-96 Authentication with 96-bit ICV.

AUTH_AES_128_GMAC AES-128-GMAC Authentication with 128-bit ICV.

AUTH_AES_192_GMAC AES-192-GMAC Authentication with 192-bit ICV.

AUTH_AES_256_GMAC AES-256-GMAC Authentication with 256-bit ICV.

AUTH_HMAC_SHA2_256_128 HMAC-SHA2-256 Authentication with 128-bit ICV.

AUTH_HMAC_SHA2_384_192 HMAC-SHA2-384 Authentication with 192-bit ICV.

AUTH_HMAC_SHA2_512_256 HMAC-SHA2-512 Authentication with 256-bit ICV.

6.5.18.12 enum ProtocolPP::ip_proto_t

IKEv2 ID PROTO fields

Parameters

<i>IP_PROTO_AN-Y</i>	- Any protocol
<i>IP_PROTO_IC-MP</i>	- ICMP protocol
<i>IP_PROTO_TC-P</i>	- Transport Control Protocol (TCP)
<i>IP_PROTO_UD-P</i>	- User Datagram Protocol (UDP)
<i>IP_PROTO_IC-MPV6</i>	- ICMP protocol version 6
<i>IP_PROTO_MH</i>	- MH Protocol

Enumerator

IP_PROTO_ANY Any Protocol.

IP_PROTO_ICMP ICMP Protocol.

IP_PROTO_TCP Transport Control Protocol (TCP)

IP_PROTO_UDP User Datagram Protocol (UDP)

IP_PROTO_ICMPV6 ICMP protocol version 6.

IP_PROTO_MH MH Protocol.

6.5.18.13 enum ProtocolPP::ipmode_t

IPsec mode of operation

Parameters

<i>TRANSPORT</i>	- Transport mode
<i>TUNNEL</i>	- Tunnel mode

Enumerator

TRANSPORT IP transport mode.

TUNNEL IP tunnel mode.

6.5.18.14 enum ProtocolPP::ipsec_mode_t

IKEv2 IPSEC MODE fields

Parameters

<i>TRANSPORT_MODE</i>	- IPsec Transport Mode
<i>TUNNEL_MODE</i>	- IPsec Tunnel Mode

Enumerator

TRANSPORT_MODE IPsec Transport Mode.

TUNNEL_MODE IPsec Tunnel Mode.

6.5.18.15 enum ProtocolPP::macsecmode_t

MACSEC Encryption Modes of Operation

Parameters

<i>AES_GCM_128</i>	- AES-GCM with 128-bit key
<i>AES_GCM_256</i>	- AES-GCM with 256-bit key
<i>AES_GCM_XP_N_128</i>	- AES-GCM with extended packet number and 128-bit key
<i>AES_GCM_XP_N_256</i>	- AES-GCM with extended packet number and 256-bit key
<i>NULL_MACSEC</i>	- Authenticate only mode AES-GMAC

Enumerator

AES_GCM_128 AES-GCM with 128-bit key.

AES_GCM_256 AES-GCM with 256-bit key.

AES_GCM_XP_N_128 AES-GCM with extended packet number and 128-bit key.

AES_GCM_XP_N_256 AES-GCM with extended packet number and 256-bit key.

NULL_MACSEC Authenticate only with AES-GMAC.

6.5.18.16 enum ProtocolPP::notify_err_t

IKE Notify Messages - Error Types NERR_UNSUPPORTED_CRIT_PAYLD - Unsupported critical payload NERR_RSVD2 - Unsupported critical payload NERR_RSVD3 - Unsupported critical payload NERR_INVALID_IKE_SPI - Invalid IKE Security Protocol Index (SPI) NERR_INVALID_MAJOR_VERSION - Invalid IKE Major Version NERR_RSVD6 NERR_INVALID_SYNTAX - Invalid syntax NERR_RSVD8 NERR_INVALID_MESSAGE_ID - ID for this message is incorrect NERR_RSVD10 NERR_INVALID_SPI - Invalid Security Protocol Index for ESP or AH NERR_RSVD12 NERR_RSVD13 NERR_NO_PROPOSAL_CHOSEN - IKE or ESP proposal was not chosen from the selection NERR_RSVD15 NERR_RSVD16 NERR_INVALID_KE_PAYLOAD - Key Exchange Payload is Invalid NERR_RSVD18 NERR_RSVD19 NERR_RSVD20 NERR_RSVD21 NERR_RSVD22 NERR_RSVD23 NERR_AUTH FAILED - Authentication of the payload failed NERR_RSVD25 NERR_RSVD26 NERR_RSVD27 NERR_RSVD28 NERR_RSVD29 NERR_RSVD30 NERR_RSVD31 NERR_RSVD32 NERR_RSVD33 NERR_SINGLE_PAIR_REQUIRED - Single Pair Required NERR_NO_ADDITIONAL_SAS - No Additional SAS NERR_INTERNAL_ADDRESSES_FAILURE - Requested Internal Address Failed NERR_FAILED_CP_REQUIRED - Failed CP Required NERR_TS_UNACCEPTABLE - Traffic Selector was not acceptable NERR_INVALID_SELECTORS - Invalid Traffic Selectors NERR_UNACCEPTABLE_ADDRESSES - Requested Addresses were not accepted NERR_UNEXPECTED_NAT_DETECTED - NAT Transversal requested but not configured NERR_USE_ASSIGNED_HOA - Use Assigned HOA NERR_TEMPORARY_FAILURE - Temporary Failure NERR_CHILD_SA_NOT_FOUND - For the connection, the Child Security Association (SA) was not found NERR_INVALID_GROUP_ID - Invalid Diffie-Hellman Group ID was requested NERR_AUTHORIZATION_FAILED - Authorization of the connection failed

Enumerator

NERR_UNSUPPORTED_CRIT_PAYLD

NERR_RSVD2
NERR_RSVD3
NERR_INVALID_IKE_SPI
NERR_INVALID_MAJOR_VERSION
NERR_RSVD6
NERR_INVALID_SYNTAX
NERR_RSVD8
NERR_INVALID_MESSAGE_ID
NERR_RSVD10
NERR_INVALID_SPI
NERR_RSVD12
NERR_RSVD13
NERR_NO_PROPOSAL_CHOSEN
NERR_RSVD15
NERR_RSVD16
NERR_INVALID_KE_PAYLOAD
NERR_RSVD18
NERR_RSVD19
NERR_RSVD20
NERR_RSVD21
NERR_RSVD22
NERR_RSVD23
NERR_AUTH_FAILED
NERR_RSVD25
NERR_RSVD26
NERR_RSVD27
NERR_RSVD28
NERR_RSVD29
NERR_RSVD30
NERR_RSVD31
NERR_RSVD32
NERR_RSVD33
NERR_SINGLE_PAIR_REQUIRED
NERR_NO_ADDITIONAL_SAS
NERR_INTERNAL_ADDRESS_FAILURE
NERR_FAILED_CP_REQUIRED
NERR_TS_UNACCEPTABLE
NERR_INVALID_SELECTORS
NERR_UNACCEPTABLE_ADDRESSES
NERR_UNEXPECTED_NAT_DETECTED
NERR_USE_ASSIGNED_HOA
NERR_TEMPORARY_FAILURE
NERR_CHILD_SA_NOT_FOUND
NERR_INVALID_GROUP_ID
NERR_AUTHORIZATION_FAILED

6.5.18.17 enum ProtocolPP::notify_status_t

IKE Notify Messages - Status Types NSTAT_INITIAL_CONTACT [RFC7296] NSTAT_SET_WINDOW_SIZE [RFC7296] NSTAT_ADDITIONAL_TS_POSSIBLE [RFC7296] NSTAT_IPCOMP_SUPPORTED [RFC7296] NSTAT_NAT_DETECTION_SOURCE_IP [RFC7296] NSTAT_NAT_DETECTION_DESTINATION_IP [RFC7296] NSTAT_COOKIE [RFC7296] NSTAT_USE_TRANSPORT_MODE [RFC7296] NSTAT_HTTP_CERT_LOOKUP_SUPPORTED [RFC7296] NSTAT_REKEY_SA [RFC7296] NSTAT_ESP_TFC_PADDING_NOT_SUPPORTED [RFC7296] NSTAT_NON_FIRST_FRAGMENTS_ALSO [RFC7296] NSTAT_MOBIKE_SUPPORTED [RFC4555] NSTAT_ADDITIONAL_IP4_ADDRESS [RFC4555] NSTAT_ADDITIONAL_IP6_ADDRESS [RFC4555] NSTAT_NO_ADDITIONAL_ADDRESSES [RFC4555] NSTAT_UPDATE_SA_ADDRESSES [RFC4555] NSTAT_COOKIE2 [RFC4555] NSTAT_NO_NATS_ALLOWED [RFC4555] NSTAT_AUTH_LIFETIME [RFC4478] NSTAT_MULTIPLE_AUTH_SUPPORTED [RFC4739] NSTAT_ANOTHER_AUTH_FOLLOWS [RFC4739] NSTAT_REDIRECT_SUPPORTED [RFC5685] NSTAT_REDIRECT [RFC5685] NSTAT_REDIRECTED_FROM [RFC5685] NSTAT_TICKET_LT_OPAQUE [RFC5723] NSTAT_TICKET_REQUEST [RFC5723] NSTAT_TICKET_ACK [RFC5723] NSTAT_TICKET_NACK [RFC5723] NSTAT_TICKET_OPAQUE [RFC5723] NSTAT_LINK_ID [RFC5739] NSTAT_USE_WESP_MODE [RFC5840] NSTAT_ROHC_SUPPORTED [RFC5857] NSTAT_EAP_ONLY_AUTHENTICATION [RFC5998] NSTAT_CHILDLESS_IKEV2_SUPPORTED [RFC6023] NSTAT_QUICK_CRASH_DETECTION [RFC6290] NSTAT_IKEV2_MESSAGE_ID_SYNC_SUPPORTED [RFC6311] NSTAT_IPSEC_REPLY_COUNTER_SYNC_SUPPORTED [RFC6311] NSTAT_IKEV2_MESSAGE_ID_SYNC [RFC6311] NSTAT_IPSEC_REPLY_COUNTER_SYNC [RFC6311] NSTAT_SECURE_PASSWORD_METHODS [RFC6467] NSTAT_PSK_PERSIST [RFC6631] NSTAT_PSK_CONFIRM [RFC6631] NSTAT_ERX_SUPPORTED [RFC6867] NSTAT_IFOM_CAPABILITY [Frederic_Firmin][3GPP TS 24.303 v10.6.0 annex B.2] NSTAT_SENDER_REQUEST_ID [draft-yeunggikev2] NSTAT_IKEV2_FRAGMENTATION_SUPPORTED [RFC7383] NSTAT_SIGNATURE_HASH_ALGORITHMS [RFC7427] NSTAT_CLONE_IKE_SA_SUPPORTED [RFC7791] NSTAT_CLONE_IKE_SA [RFC7791] NSTAT_PUZZLE [RFC8019] NSTAT_USE_PPK [draft-ietf-ipsecme-qikev2] NSTAT_PPK_IDENTITY [draft-ietf-ipsecme-qikev2] NSTAT_NO_PPK_AUTH [draft-ietf-ipsecme-qikev2]

Enumerator

- NSTAT_INITIAL_CONTACT**
- NSTAT_SET_WINDOW_SIZE**
- NSTAT_ADDITIONAL_TS_POSSIBLE**
- NSTAT_IPCOMP_SUPPORTED**
- NSTAT_NAT_DETECTION_SOURCE_IP**
- NSTAT_NAT_DETECTION_DESTINATION_IP**
- NSTAT_COOKIE**
- NSTAT_USE_TRANSPORT_MODE**
- NSTAT_HTTP_CERT_LOOKUP_SUPPORTED**
- NSTAT_REKEY_SA**
- NSTAT_ESP_TFC_PADDING_NOT_SUPPORTED**
- NSTAT_NON_FIRST_FRAGMENTS_ALSO**
- NSTAT_MOBIKE_SUPPORTED**
- NSTAT_ADDITIONAL_IP4_ADDRESS**
- NSTAT_ADDITIONAL_IP6_ADDRESS**
- NSTAT_NO_ADDITIONAL_ADDRESSES**
- NSTAT_UPDATE_SA_ADDRESSES**
- NSTAT_COOKIE2**
- NSTAT_NO_NATS_ALLOWED**
- NSTAT_AUTH_LIFETIME**
- NSTAT_MULTIPLE_AUTH_SUPPORTED**
- NSTAT_ANOTHER_AUTH_FOLLOWS**
- NSTAT_REDIRECT_SUPPORTED**

```
NSTAT_REDIRECT
NSTAT_REDIRECTED_FROM
NSTAT_TICKET_LT_OPAQUE
NSTAT_TICKET_REQUEST
NSTAT_TICKET_ACK
NSTAT_TICKET_NACK
NSTAT_TICKET_OPAQUE
NSTAT_LINK_ID
NSTAT_USE_WESP_MODE
NSTAT_ROHC_SUPPORTED
NSTAT_EAP_ONLY_AUTHENTICATION
NSTAT_CHILDLESS_IKEV2_SUPPORTED
NSTAT_QUICK_CRASH_DETECTION
NSTAT_IKEV2_MESSAGE_ID_SYNC_SUPPORTED
NSTAT_IPSEC_REPLAY_COUNTER_SYNC_SUPPORTED
NSTAT_IKEV2_MESSAGE_ID_SYNC
NSTAT_IPSEC_REPLAY_COUNTER_SYNC
NSTAT_SECURE_PASSWORD_METHODS
NSTAT_PSK_PERSIST
NSTAT_PSK_CONFIRM
NSTAT_ERX_SUPPORTED
NSTAT_IFOM_CAPABILITY
NSTAT_SENDER_REQUEST_ID
NSTAT_IKEV2_FRAGMENTATION_SUPPORTED
NSTAT_SIGNATURE_HASH_ALGORITHMS
NSTAT_CLONE_IKE_SA_SUPPORTED
NSTAT_CLONE_IKE_SA
NSTAT_PUZZLE
NSTAT_USE_PPK
NSTAT_PPK_IDENTITY
NSTAT_NO_PPK_AUTH
```

6.5.18.18 enum ProtocolPP::pad_t

padding types to generate for packet padding

Parameters

RANDOM	- generate random string of bytes for padding
INCREMENT	- generate a series of incrementing bytes for padding
ZERO	- generate a string of zeros for padding
SIZE	- generate a string of bytes with a value equal to size

Enumerator

- RANDOM** Generate random string of bytes.
- INCREMENT** Generate an incrementing string of bytes.
- ZERO** Generate a string of zeros.
- SIZE** Generate a string of bytes with value of size.
- UNKNWN** Generate unknown padding.

6.5.18.19 enum ProtocolPP::platform_t

Testbench platform, only used for jtestbench to define how the software rings are accessed and how the security associations are handled

Parameters

WASPPLAT	- Testbench uses jresponder to simulate a platform to test against. Allows randomization of read, write and processing times
SECPLAT	- Support for SEC block in QorIQ, Layerscape, and iMX* processors from NXP/Qualcomm. This testbench can program the address and size of the software rings which SEC can then read

Enumerator

WASPPLAT Testbench uses jresponder to simulate.

SECPLAT Support for SEC block.

6.5.18.20 enum ProtocolPP::prf_id_t

IKEv2 PRF_ID fields

Parameters

PRF_HMAC_M-D5	- HMAC-MD5 Pseudo Random Function with 16-byte ICV
PRF_HMAC_S-HA1	- HMAC-SHA1 Pseudo Random Function with 20-byte ICV
PRF_HMAC_TI-GER	- HMAC-TIGER Pseudo Random Function
PRF_AES128-XCBC	- AES-XCBC Pseudo Random Function with 128-bit key and 16-byte ICV
PRF_HMAC_S-HA2_256	- HMAC-SHA2-256 Authenticaiton with 32-byte ICV
PRF_HMAC_S-HA2_384	- HMAC-SHA2-384 Pseudo Random Function with 48-byte ICV
PRF_HMAC_S-HA2_512	- HMAC-SHA2-512 Pseudo Random Function with 64-byte ICV
PRF_AES128-CMAC	- AES-CMAC Pseudo Random Function with 128-bit key and 16-byte ICV

Enumerator

PRF_HMAC_MD5 HMAC-MD5 Pseudo Random Function with 16-byte ICV.

PRF_HMAC_SHA1 HMAC-SHA1 Pseudo Random Function with 20-byte ICV.

PRF_HMAC_TIGER HMAC-TIGER Pseudo Random Function.

PRF_AES128_XCBC AES-XCBC Pseudo Random Function with 128-bit key and 16-byte ICV.

PRF_HMAC_SHA2_256 HMAC-SHA2-256 Pseudo Random Function with 32-byte ICV.

PRF_HMAC_SHA2_384 HMAC-SHA2-384 Pseudo Random Function with 48-byte ICV.

PRF_HMAC_SHA2_512 HMAC-SHA2-512 Pseudo Random Function with 64-byte ICV.

PRF_AES128_CMAC AES-CMAC Psuedo Random Function with 128-bit key and 16-byte ICV.

6.5.18.21 enum ProtocolPP::protocol_id_t

IKEv2 Protocol Identifiers

Parameters

<i>PROTO_IKE</i>	- Request Internet Key Exchange Protocol (IKE)
<i>PROTO_AH</i>	- Request Authentication Header Protocol (AH)
<i>PROTO_ESP</i>	- Request Encapsulating Security Protocol (ESP)
<i>PROTO_FC_E- SP_HDR</i>	- FC ESP Header
<i>PROTO_FC_C- T_AUTH</i>	- FC CT Authentication

Enumerator

PROTO_IKE Request Internet Key Exchange Protocol (IKE)

PROTO_AH Request Authentication Header Protocol (AH)

PROTO_ESP Request Encapsulating Security Protocol (ESP)

PROTO_FC_ESP_HDR FC ESP Header.

PROTO_FC_CT_AUTH FC CT Authentication.

6.5.18.22 enum ProtocolPP::protocol_t

Protocols not found in the standard IANA values

Parameters

<i>NO_ERROR</i>	- No error in protocol
<i>PROTOCOL</i>	- Protocol++ base class
<i>MACSEC</i>	- IEEE802 Security protocol
<i>WIFI</i>	- IEEE802.11 Wireless Protocol
<i>WIGIG</i>	- IEEE802.11ad Wireless Protocol
<i>WIMAX</i>	- IEEE802.16 Wireless Protocol
<i>LTE</i>	- Long Term Evolution (LTE)/3GPP Protocols
<i>RLC</i>	- Radio Link Control Protocol (RLC)
<i>TLS</i>	- Transport Layer Security (TLS)
<i>SRTP</i>	- Secure Real Time Protocol (SRTP)
<i>UDPP</i>	- User Data Protocol (UDP)
<i>TCPP</i>	- Transport Control Protocol (TCP)
<i>ICMPP</i>	- Internet Control Message Protocol (ICMP)
<i>IP</i>	- IP protocol
<i>IPSEC</i>	- IPsec Protocol (ESP)
<i>XMLPARSER</i>	- TinyXML2 parser
<i>WASP</i>	- WASP protocolpp front end
<i>RINGDRIVER</i>	- Ring based driver
<i>DIRECTDRIVER</i>	- Direct driver
<i>DRIVER</i>	- Driver
<i>LOOPBACK</i>	- Loopback mode for the Ethernet interface
<i>ETHERNET</i>	- Ethernet interface
<i>IKEV2</i>	- Internet Key Exchange (IKE) version 2
<i>NOPROTO</i>	- No protocol selected

Enumerator

NO_ERROR No Error.

PROTOCOL JProtocol base class.

MACSEC Macsec protocol.

WIFI Wifi protocol.

WIGIG WiGig protocol.

WIMAX WiMax protocol.

LTE LTE/3GPP protocol.

RLC Radio Link Control.

TLS TLS/SSL protocol.

SRTP Secure RTP protocol.

UDPP UDP.

TCPP TCP.

ICMPP ICMP.

IP IP.

IPSEC IPsec.

XMLPARSER XML Parser.

WASP WASP front end.

RINGDRIVER Ring Driver.

DIRECTDRIVER Direct Driver.

DRIVER Driver.

LOOPBACK Ethernet LoopBack mode.

ETHERNET Ethernet.

IKEV2 Internet Key Exchange (IKE) version 2.

NOPROTO No protocol selected.

6.5.18.23 enum ProtocolIPP::replay_t

Anti-Replay sequence number types

Parameters

NORMAL	- Sequence number incremented (seqnum = seqnum + 1)
SHIFT	- Sequence number in the future (seqnum = seqnum + random("2..50"))
WINDOW	- Sequence number found in the window
REPLAY	- Sequence number is replay
LATE	- Sequence number is outside the window (LATE)
OVERFLOW	- Sequence number overflow
UNDERFLOW	- Sequence number underflow

Enumerator

NORMAL Sequence number incremented (seqnum = seqnum + 1)

SHIFT Sequence number in the future, shifts window left (seqnum = seqnum + random("2..31"))

WINDOW Sequence number found in the window.

REPLAY Sequence number is replay.

LATE Sequence number is outside the window (LATE)

ROLLOVER Sequence number overflow.

ROLLUNDER Sequence number underflow.

6.5.18.24 enum ProtocolIPP::rohc_attr_t

IKE ROHC Attribute Type

Parameters

<i>ROHC_MAX_CID</i>	- ROHC Maximum CID
<i>ROHC_PROFILE</i>	- ROHC Profile
<i>ROHC_INTEG</i>	- ROHC Integrity Algorithm
<i>ROHC_ICV_LEN</i>	- ROHC ICV Length
<i>ROHC_MRRU</i>	- ROHC MRRU

Enumerator

ROHC_MAX_CID ROHC Maximum CID.

ROHC_PROFILE ROHC Profile.

ROHC_INTEG ROHC Integrity Algorithm.

ROHC_ICV_LEN ROHC ICV Length.

ROHC_MRRU ROHC MRRU.

6.5.18.25 enum ProtocolPP::role_id_t

IKEv2 ROLE_ID fields

Parameters

<i>ROLE_INITIATOR</i>	- Initiator role
<i>ROLE_RESPONDER</i>	- Responder role
<i>ROLE_ANY</i>	- Both an initiator and receiver

Enumerator

ROLE_INITIATOR Initiator role.

ROLE_RESPONDER Responder role.

ROLE_ANY Both an initiator and receiver.

6.5.18.26 enum ProtocolPP::rsapadtype_t

RSA Pad formats

Parameters

<i>PKCS15</i>	- PKCS v1.5 padding
<i>PSS</i>	- Secure padding

Enumerator

PKCS15 PKCS v1.5 padding.

PSS Secure padding.

6.5.18.27 enum ProtocolPP::srtpcipher_t

SRTP ciphers

Parameters

<code>NULL_SRTP</code>	- Null encryption and authentication
<code>AES_CTR_SH-A1</code>	- AES Counter mode encryption with SHA1 authentication
<code>AEAD_AES-128_GCM</code>	- AES Galois Counter Mode with 16-byte key and 16-byte ICV
<code>AEAD_AES-256_GCM</code>	- AES Galois Counter Mode with 32-byte key and 16-byte ICV
<code>AEAD_AES-128_GCM_8</code>	- AES Galois Counter Mode with 16-byte key and 8-byte ICV
<code>AEAD_AES-256_GCM_8</code>	- AES Galois Counter Mode with 32-byte key and 8-byte ICV
<code>AEAD_AES-128_GCM_12</code>	- AES Galois Counter Mode with 16-byte key and 12-byte ICV
<code>AEAD_AES-256_GCM_12</code>	- AES Galois Counter Mode with 32-byte key and 12-byte ICV
<code>AEAD_AES-128_CCM</code>	- AES CCM Mode with 16-byte key and 16-byte ICV
<code>AEAD_AES-256_CCM</code>	- AES CCM Mode with 32-byte key and 16-byte ICV
<code>AEAD_AES-128_CCM_8</code>	- AES CCM Mode with 16-byte key and 8-byte ICV
<code>AEAD_AES-256_CCM_8</code>	- AES CCM Mode with 32-byte key and 8-byte ICV
<code>AEAD_AES-128_CCM_12</code>	- AES CCM Mode with 16-byte key and 12-byte ICV
<code>AEAD_AES-256_CCM_12</code>	- AES CCM Mode with 32-byte key and 12-byte ICV
<code>ARIA_128_CTR-HMAC_SHA1-80</code>	- ARIA-CTR with 16-byte key and 10-byte ICV
<code>ARIA_128_CTR-HMAC_SHA1-32</code>	- ARIA-CTR with 16-byte key and 4-byte ICV
<code>ARIA_192_CTR-HMAC_SHA1-80</code>	- ARIA-CTR with 24-byte key and 10-byte ICV
<code>ARIA_192_CTR-HMAC_SHA1-32</code>	- ARIA-CTR with 24-byte key and 4-byte ICV
<code>ARIA_256_CTR-HMAC_SHA1-80</code>	- ARIA-CTR with 32-byte key and 10-byte ICV
<code>ARIA_256_CTR-HMAC_SHA1-32</code>	- ARIA-CTR with 32-byte key and 4-byte ICV

AEAD_ARIA_- 128_GCM	- ARIA Galois Counter Mode with 16-byte key and 16-byte ICV
AEAD_ARAI_- 256_GCM	- ARIA Galois Counter Mode with 32-byte key and 16-byte ICV

Enumerator

NULL_SRTP NULL encryption.**AES_CTR_SHA1** AES-CTR counter mode with 20-byte ICV.**AEAD_AES_128_GCM** AES-GCM with 16-byte key and 16-byte ICV.**AEAD_AES_256_GCM** AES-GCM with 32-byte key and 16-byte ICV.**AEAD_AES_128_GCM_8** AES-GCM with 16-byte key and 8-byte ICV.**AEAD_AES_256_GCM_8** AES-GCM with 32-byte key and 8-byte ICV.**AEAD_AES_128_GCM_12** AES-GCM with 16-byte key and 12-byte ICV.**AEAD_AES_256_GCM_12** AES-GCM with 32-byte key and 12-byte ICV.**AEAD_AES_128_CCM** AES-CCM with 16-byte key and 16-byte ICV.**AEAD_AES_256_CCM** AES-CCM with 32-byte key and 16-byte ICV.**AEAD_AES_128_CCM_8** AES-CCM with 16-byte key and 8-byte ICV.**AEAD_AES_256_CCM_8** AES-CCM with 32-byte key and 8-byte ICV.**AEAD_AES_128_CCM_12** AES-CCM with 16-byte key and 12-byte ICV.**AEAD_AES_256_CCM_12** AES-CCM with 32-byte key and 12-byte ICV.**ARIA_128_CTR_HMAC_SHA1_80** ARIA-CTR with 16-byte key and 10-byte ICV.**ARIA_128_CTR_HMAC_SHA1_32** ARIA-CTR with 16-byte key and 4-byte ICV.**ARIA_192_CTR_HMAC_SHA1_80** ARIA-CTR with 24-byte key and 10-byte ICV.**ARIA_192_CTR_HMAC_SHA1_32** ARIA-CTR with 24-byte key and 4-byte ICV.**ARIA_256_CTR_HMAC_SHA1_80** ARIA-CTR with 32-byte key and 10-byte ICV.**ARIA_256_CTR_HMAC_SHA1_32** ARIA-CTR with 32-byte key and 4-byte ICV.**AEAD_ARIA_128_GCM** ARIA-GCM with 16-byte key and 16-byte ICV.**AEAD_ARIA_256_GCM** ARIA-GCM with 32-byte key and 16-byte ICV.

6.5.18.28 enum ProtocolPP::tls_ciphersuite_t

Ciphersuite values for TLS/SSL

Parameters

TLS_NULL_WI- TH_NULL_NULL	- 0x0000
TLS_RSA_WIT- H_NULL_MD5	- 0x0001
TLS_RSA_WIT- H_NULL_SHA	- 0x0002
TLS_RSA_EXP- ORT_WITH_R- C4_40_MD5	- 0x0003

<i>TLS_RSA_WIT-H_RC4_128_M-D5</i>	- 0x0004
<i>TLS_RSA_WIT-H_RC4_128_S-HA</i>	- 0x0005
<i>TLS_RSA_EXPORT_WITH_RC2_CBC_40_M-D5</i>	- 0x0006
<i>TLS_RSA_WIT-H_IDEA_CBC_SHA</i>	- 0x0007
<i>TLS_RSA_EXPORT_WITH_DES40_CBC_SHA</i>	- 0x0008
<i>TLS_RSA_WIT-H DES_CBC_S-HA</i>	- 0x0009
<i>TLS_RSA_WIT-H_3DES_EDE_CBC_SHA</i>	- 0x000A
<i>TLS_DH_DSS_EXPORT_WITH_DES40_CBC_SHA</i>	- 0x000B
<i>TLS_DH_DSS_WITH_DES_CBC_SHA</i>	- 0x000C
<i>TLS_DH_DSS_WITH_3DES_EDE_CBC_SHA</i>	- 0x000D
<i>TLS_DH_RSA_EXPORT_WITH_DES40_CBC_SHA</i>	- 0x000E
<i>TLS_DH_RSA_WITH_DES_CBC_SHA</i>	- 0x000F
<i>TLS_DH_RSA_WITH_3DES_EDE_CBC_SHA</i>	- 0x0010
<i>TLS_DHE_DSS_EXPORT_WITH_DES40_CBC_SHA</i>	- 0x0011
<i>TLS_DHE_DSS_WITH_DES_CBC_SHA</i>	- 0x0012
<i>TLS_DHE_DSS_WITH_3DES_EDE_CBC_SHA</i>	- 0x0013

<i>TLS_DHE_RSA_EXPORT_WITH_DES40_CBC_SHA</i>	- 0x0014
<i>TLS_DHE_RSA_WITH_DES_CBC_SHA</i>	- 0x0015
<i>TLS_DHE_RSA_WITH_3DES_EDE_CBC_SHA</i>	- 0x0016
<i>TLS_DH_ANON_EXPORT_WITH_RC4_40_MD5</i>	- 0x0017
<i>TLS_DH_ANON_WITH_RC4_128_MD5</i>	- 0x0018
<i>TLS_DH_ANON_EXPORT_WITH_DES_CBC_SHA</i>	- 0x0019
<i>TLS_DH_ANON_WITH_DES_CBC_SHA</i>	- 0x001A
<i>TLS_DH_ANON_WITH_3DES_EDE_CBC_SHA</i>	- 0x001B
<i>TLS_KRB5_WITH_DES_CBC_SHA</i>	- 0x001E
<i>TLS_KRB5_WITH_3DES_EDE_CBC_SHA</i>	- 0x001F
<i>TLS_KRB5_WITH_RC4_128_SHA</i>	- 0x0020
<i>TLS_KRB5_WITH_IDEA_CBC_SHA</i>	- 0x0021
<i>TLS_KRB5_WITH_DES_CBC_MD5</i>	- 0x0022
<i>TLS_KRB5_WITH_3DES_EDE_CBC_MD5</i>	- 0x0023
<i>TLS_KRB5_WITH_RC4_128_MD5</i>	- 0x0024
<i>TLS_KRB5_WITH_IDEA_CBC_MD5</i>	- 0x0025

<i>TLS_KRB5_EX-PORT_WITH_D-ES_CBC_40_S-HA</i>	- 0x0026
<i>TLS_KRB5_EX-PORT_WITH_R-C2_CBC_40_S-HA</i>	- 0x0027
<i>TLS_KRB5_EX-PORT_WITH_R-C4_40_SHA</i>	- 0x0028
<i>TLS_KRB5_EX-PORT_WITH_D-ES_CBC_40_M-D5</i>	- 0x0029
<i>TLS_KRB5_EX-PORT_WITH_R-C2_CBC_40_M-D5</i>	- 0x002A
<i>TLS_KRB5_EX-PORT_WITH_R-C4_40_MD5</i>	- 0x002B
<i>TLS_PSK_WIT-H_NULL_SHA</i>	- 0x002C
<i>TLS_DHE_PSK-_WITH_NULL-_SHA</i>	- 0x002D
<i>TLS_RSA_PSK-_WITH_NULL-_SHA</i>	- 0x002E
<i>TLS_RSA_WIT-H_AES_128_C-BC_SHA</i>	- 0x002F
<i>TLS_DH_DSS-_WITH_AES-_128_CBC_SHA</i>	- 0x0030
<i>TLS_DH_RSA-_WITH_AES-_128_CBC_SHA</i>	- 0x0031
<i>TLS_DHE_DSS-_WITH_AES-_128_CBC_SHA</i>	- 0x0032
<i>TLS_DHE_RSA-_WITH_AES-_128_CBC_SHA</i>	- 0x0033
<i>TLS_DH_ANO-N_WITH_AES-_128_CBC_SHA</i>	- 0x0034
<i>TLS_RSA_WIT-H_AES_256_C-BC_SHA</i>	- 0x0035

<i>TLS_DH_DSS_- WITH_AES_- 256_CBC_SHA</i>	- 0x0036
<i>TLS_DH_RSA_- WITH_AES_- 256_CBC_SHA</i>	- 0x0037
<i>TLS_DHE_DSS- _WITH_AES_- 256_CBC_SHA</i>	- 0x0038
<i>TLS_DHE_RSA- _WITH_AES_- 256_CBC_SHA</i>	- 0x0039
<i>TLS_DH_ANO- N_WITH_AES_- 256_CBC_SHA</i>	- 0x003A
<i>TLS_RSA_WIT- H_NULL_SH- A256</i>	- 0x003B
<i>TLS_RSA_WIT- H_AES_128_C- BC_SHA256</i>	- 0x003C
<i>TLS_RSA_WIT- H_AES_256_C- BC_SHA256</i>	- 0x003D
<i>TLS_DH_DSS_- WITH_AES_- 128_CBC_SH- A256</i>	- 0x003E
<i>TLS_DH_RSA_- WITH_AES_- 128_CBC_SH- A256</i>	- 0x003F
<i>TLS_DHE_DSS- _WITH_AES_- 128_CBC_SH- A256</i>	- 0x0040
<i>TLS_RSA_WIT- H_CAMELLIA_- 128_CBC_SHA</i>	- 0x0041
<i>TLS_DH_DSS_- WITH_CAMELL- IA_128_CBC_S- HA</i>	- 0x0042
<i>TLS_DH_RSA_- WITH_CAMELL- IA_128_CBC_S- HA</i>	- 0x0043
<i>TLS_DHE_DSS- _WITH_CAMEL- LIA_128_CBC_- SHA</i>	- 0x0044

<i>TLS_DHE_RSA_WITH_CAMELLIA_128_CBC_SHA</i>	- 0x0045
<i>TLS_DH_ANON_WITH_CAMELLIA_128_CBC_SHA</i>	- 0x0046
<i>TLS_DHE_RSA_WITH_AES_128_CBC_SHA256</i>	- 0x0067
<i>TLS_DH_DSS_WITH_AES_256_CBC_SHA256</i>	- 0x0068
<i>TLS_DH_RSA_WITH_AES_256_CBC_SHA256</i>	- 0x0069
<i>TLS_DHE_DSS_WITH_AES_256_CBC_SHA256</i>	- 0x006A
<i>TLS_DHE_RSA_WITH_AES_256_CBC_SHA256</i>	- 0x006B
<i>TLS_DH_ANON_WITH_AES_128_CBC_SHA256</i>	- 0x006C
<i>TLS_DH_ANON_WITH_AES_256_CBC_SHA256</i>	- 0x006D
<i>TLS_RSA_WITH_CAMELLIA_256_CBC_SHA</i>	- 0x0084
<i>TLS_DH_DSS_WITH_CAMELLIA_256_CBC_SHA</i>	- 0x0085
<i>TLS_DH_RSA_WITH_CAMELLIA_256_CBC_SHA</i>	- 0x0086
<i>TLS_DHE_DSS_WITH_CAMELLIA_256_CBC_SHA</i>	- 0x0087

<i>TLS_DHE_RSA_WITH_CAMELIA_256_CBC_SHA</i>	- 0x0088
<i>TLS_DH_ANON_WITH_CAMELLIA_256_CBC_SHA</i>	- 0x0089
<i>TLS_PSK_WITH_RC4_128_SHA</i>	- 0x008A
<i>TLS_PSK_WITH_3DES_EDE_CBC_SHA</i>	- 0x008B
<i>TLS_PSK_WITH_AES_128_CBC_SHA</i>	- 0x008C
<i>TLS_PSK_WITH_AES_256_CBC_SHA</i>	- 0x008D
<i>TLS_DHE_PSK_WITH_RC4_128_SHA</i>	- 0x008E
<i>TLS_DHE_PSK_WITH_3DES_EDE_CBC_SHA</i>	- 0x008F
<i>TLS_DHE_PSK_WITH_AES_128_CBC_SHA</i>	- 0x0090
<i>TLS_DHE_PSK_WITH_AES_256_CBC_SHA</i>	- 0x0091
<i>TLS_RSA_PSK_WITH_RC4_128_SHA</i>	- 0x0092
<i>TLS_RSA_PSK_WITH_3DES_EDE_CBC_SHA</i>	- 0x0093
<i>TLS_RSA_PSK_WITH_AES_128_CBC_SHA</i>	- 0x0094
<i>TLS_RSA_PSK_WITH_AES_256_CBC_SHA</i>	- 0x0095
<i>TLS_RSA_WITH_SEED_CBC_SHA</i>	- 0x0096
<i>TLS_DH_DSS_WITH_SEED_CBC_SHA</i>	- 0x0097

<code>TLS_DH_RSA_- WITH_SEED_- CBC_SHA</code>	- 0x0098
<code>TLS_DHE_DSS- _WITH_SEED_- CBC_SHA</code>	- 0x0099
<code>TLS_DHE_RSA- _WITH_SEED_- CBC_SHA</code>	- 0x009A
<code>TLS_DH_ANO- N_WITH_SEED- _CBC_SHA</code>	- 0x009B
<code>TLS_RSA_WIT- H_AES_128_G- CM_SHA256</code>	- 0x009C
<code>TLS_RSA_WIT- H_AES_256_G- CM_SHA384</code>	- 0x009D
<code>TLS_DHE_RSA- _WITH_AES_- 128_GCM_SH- A256</code>	- 0x009E
<code>TLS_DHE_RSA- _WITH_AES_- 256_GCM_SH- A384</code>	- 0x009F
<code>TLS_DH_RSA_- WITH_AES_- 128_GCM_SH- A256</code>	- 0x00A0
<code>TLS_DH_RSA_- WITH_AES_- 256_GCM_SH- A384</code>	- 0x00A1
<code>TLS_DHE_DSS- _WITH_AES_- 128_GCM_SH- A256</code>	- 0x00A2
<code>TLS_DHE_DSS- _WITH_AES_- 256_GCM_SH- A384</code>	- 0x00A3
<code>TLS_DH_DSS_- WITH_AES_- 128_GCM_SH- A256</code>	- 0x00A4
<code>TLS_DH_DSS_- WITH_AES_- 256_GCM_SH- A384</code>	- 0x00A5

<i>TLS_DH_ANO-N_WITH_AES_-128_GCM_SH-A256</i>	- 0x00A6
<i>TLS_DH_ANO-N_WITH_AES_-256_GCM_SH-A384</i>	- 0x00A7
<i>TLS_PSK_WIT-H_AES_128_G-CM_SHA256</i>	- 0x00A8
<i>TLS_PSK_WIT-H_AES_256_G-CM_SHA384</i>	- 0x00A9
<i>TLS_DHE_PSK-_WITH_AES_-128_GCM_SH-A256</i>	- 0x00AA
<i>TLS_DHE_PSK-_WITH_AES_-256_GCM_SH-A384</i>	- 0x00AB
<i>TLS_RSA_PSK-_WITH_AES_-128_GCM_SH-A256</i>	- 0x00AC
<i>TLS_RSA_PSK-_WITH_AES_-256_GCM_SH-A384</i>	- 0x00AD
<i>TLS_PSK_WIT-H_AES_128_C-BC_SHA256</i>	- 0x00AE
<i>TLS_PSK_WIT-H_AES_256_C-BC_SHA384</i>	- 0x00AF
<i>TLS_PSK_WIT-H_NULL_SH-A256</i>	- 0x00B0
<i>TLS_PSK_WIT-H_NULL_SH-A384</i>	- 0x00B1
<i>TLS_DHE_PSK-_WITH_AES_-128_CBC_SH-A256</i>	- 0x00B2
<i>TLS_DHE_PSK-_WITH_AES_-256_CBC_SH-A384</i>	- 0x00B3

<code>TLS_DHE_PSK- _WITH_NULL_- SHA256</code>	- 0x00B4
<code>TLS_DHE_PSK- _WITH_NULL_- SHA384</code>	- 0x00B5
<code>TLS_RSA_PSK- _WITH_AES_- 128_CBC_SH- A256</code>	- 0x00B6
<code>TLS_RSA_PSK- _WITH_AES_- 256_CBC_SH- A384</code>	- 0x00B7
<code>TLS_RSA_PSK- _WITH_NULL_- SHA256</code>	- 0x00B8
<code>TLS_RSA_PSK- _WITH_NULL_- SHA384</code>	- 0x00B9
<code>TLS_RSA_WIT- H_CAMELLIA_- 128_CBC_SH- A256</code>	- 0x00BA
<code>TLS_DH_DSS_- WITH_CAMELL- IA_128_CBC_S- HA256</code>	- 0x00BB
<code>TLS_DH_RSA_- WITH_CAMELL- IA_128_CBC_S- HA256</code>	- 0x00BC
<code>TLS_DHE_DSS- _WITH_CAMEL- LIA_128_CBC_- SHA256</code>	- 0x00BD
<code>TLS_DHE_RSA- _WITH_CAMEL- LIA_128_CBC_- SHA256</code>	- 0x00BE
<code>TLS_DH_ANO- N_WITH_CAM- ELLIA_128_CB- C_SHA256</code>	- 0x00BF
<code>TLS_RSA_WIT- H_CAMELLIA_- 256_CBC_SH- A256</code>	- 0x00C0
<code>TLS_DH_DSS_- WITH_CAMELL- IA_256_CBC_S- HA256</code>	- 0x00C1

<i>TLS_DH_RSA_WITH_CAMELLIA_256_CBC_SHA256</i>	- 0x00C2
<i>TLS_DHE_DSS_WITH_CAMELLIA_256_CBC_SHA256</i>	- 0x00C3
<i>TLS_DHE_RSA_WITH_CAMELLIA_256_CBC_SHA256</i>	- 0x00C4
<i>TLS_DH_ANON_WITH_CAMELLIA_256_CBC_SHA256</i>	- 0x00C5
<i>TLS_ECDH_ECDSA_WITH_NULL_SHA</i>	- 0xC001
<i>TLS_ECDH_ECDSA_WITH_RC4_128_SHA</i>	- 0xC002
<i>TLS_ECDH_ECDSA_WITH_3DES_EDE_CBC_SHA</i>	- 0xC003
<i>TLS_ECDH_ECDSA_WITH_AESE_128_CBC_SHA</i>	- 0xC004
<i>TLS_ECDH_ECDSA_WITH_AESE_256_CBC_SHA</i>	- 0xC005
<i>TLS_ECDHE_ECDSA_WITH_NULL_SHA</i>	- 0xC006
<i>TLS_ECDHE_ECDSA_WITH_RC4_128_SHA</i>	- 0xC007
<i>TLS_ECDHE_ECDSA_WITH_3DES_EDE_CBC_SHA</i>	- 0xC008
<i>TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA</i>	- 0xC009
<i>TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA</i>	- 0xC00A

<code>TLS_ECDH_RS-A_WITH_NULL_SHA</code>	- 0xC00B
<code>TLS_ECDH_RS-A_WITH_RC4-128_SHA</code>	- 0xC00C
<code>TLS_ECDH_RS-A_WITH_3DES-EDE_CBC_SHA</code>	- 0xC00D
<code>TLS_ECDH_RS-A_WITH_AES-128_CBC_SHA</code>	- 0xC00E
<code>TLS_ECDH_RS-A_WITH_AES-256_CBC_SHA</code>	- 0xC00F
<code>TLS_ECDHE_RSA_WITH_NULL_SHA</code>	- 0xC010
<code>TLS_ECDHE_RSA_WITH_RC4-128_SHA</code>	- 0xC011
<code>TLS_ECDHE_RSA_WITH_3DES_EDE_CBC_SHA</code>	- 0xC012
<code>TLS_ECDHE_RSA_WITH_AES-128_CBC_SHA</code>	- 0xC013
<code>TLS_ECDHE_RSA_WITH_AES-256_CBC_SHA</code>	- 0xC014
<code>TLS_ECDH_ANON_WITH_NULL_SHA</code>	- 0xC015
<code>TLS_ECDH_ANON_WITH_RC4-128_SHA</code>	- 0xC016
<code>TLS_ECDH_ANON_WITH_3DES_EDE_CBC_SHA</code>	- 0xC017
<code>TLS_ECDH_ANON_WITH_AES-128_CBC_SHA</code>	- 0xC018
<code>TLS_ECDH_ANON_WITH_AES-256_CBC_SHA</code>	- 0xC019
<code>TLS_SRP_SHA-WITH_3DES-EDE_CBC_SHA</code>	- 0xC01A

<i>TLS_SR_P_SHA-RSA_WITH_3-DES_EDE_CBC_SHA</i>	- 0xC01B
<i>TLS_SR_P_SHA-DSS_WITH_3-DES_EDE_CBC_SHA</i>	- 0xC01C
<i>TLS_SR_P_SHA-WITH_AES-128_CBC_SHA</i>	- 0xC01D
<i>TLS_SR_P_SHA-RSA_WITH_AES_128_CBC_SHA</i>	- 0xC01E
<i>TLS_SR_P_SHA-DSS_WITH_AES_128_CBC_SHA</i>	- 0xC01F
<i>TLS_SR_P_SHA-WITH_AES-256_CBC_SHA</i>	- 0xC020
<i>TLS_SR_P_SHA-RSA_WITH_AES_256_CBC_SHA</i>	- 0xC021
<i>TLS_SR_P_SHA-DSS_WITH_AES_256_CBC_SHA</i>	- 0xC022
<i>TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256</i>	- 0xC023
<i>TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384</i>	- 0xC024
<i>TLS_ECDH_ECDSA_WITH_AES_128_CBC_SHA256</i>	- 0xC025
<i>TLS_ECDH_ECDSA_WITH_AES_256_CBC_SHA384</i>	- 0xC026
<i>TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256</i>	- 0xC027

<i>TLS_ECDHE_R-SA_WITH_AES_-256_CBC_SH-A384</i>	- 0xC028
<i>TLS_ECDH_RS-A_WITH_AES_-128_CBC_SH-A256</i>	- 0xC029
<i>TLS_ECDH_RS-A_WITH_AES_-256_CBC_SH-A384</i>	- 0xC02A
<i>TLS_ECDHE_E-CDSA_WITH_AES_128_GCM_SHA256</i>	- 0xC02B
<i>TLS_ECDHE_E-CDSA_WITH_AES_256_GCM_SHA384</i>	- 0xC02C
<i>TLS_ECDH_EC-DSA_WITH_AES_128_GCM_SHA256</i>	- 0xC02D
<i>TLS_ECDH_EC-DSA_WITH_AES_256_GCM_SHA384</i>	- 0xC02E
<i>TLS_ECDHE_R-SA_WITH_AES_-128_GCM_SH-A256</i>	- 0xC02F
<i>TLS_ECDHE_R-SA_WITH_AES_-256_GCM_SH-A384</i>	- 0xC030
<i>TLS_ECDH_RS-A_WITH_AES_-128_GCM_SH-A256</i>	- 0xC031
<i>TLS_ECDH_RS-A_WITH_AES_-256_GCM_SH-A384</i>	- 0xC032
<i>TLS_ECDHE_P-SK_WITH_RC4_-128_SHA</i>	- 0xC033
<i>TLS_ECDHE_P-SK_WITH_3DE-S_EDE_CBC_SH-A</i>	- 0xC034

<i>TLS_ECDHE_P-SK_WITH_AES-128_CBC_SHA</i>	- 0xC035
<i>TLS_ECDHE_P-SK_WITH_AES-256_CBC_SHA</i>	- 0xC036
<i>TLS_ECDHE_P-SK_WITH_AES-128_CBC_SHA256</i>	- 0xC037
<i>TLS_ECDHE_P-SK_WITH_AES-256_CBC_SHA256</i>	- 0xC038
<i>TLS_ECDHE_P-SK_WITH_NUL_L_SHA</i>	- 0xC039
<i>TLS_ECDHE_P-SK_WITH_NUL_L_SHA256</i>	- 0xC03A
<i>TLS_ECDHE_P-SK_WITH_NUL_L_SHA384</i>	- 0xC03B
<i>TLS_RSA_WITH_ARIA_128_CBC_SHA256</i>	- 0xC03C
<i>TLS_RSA_WITH_ARIA_256_CBC_SHA384</i>	- 0xC03D
<i>TLS_DH_DSS_WITH_ARIA_128_CBC_SHA256</i>	- 0xC03E
<i>TLS_DH_DSS_WITH_ARIA_256_CBC_SHA384</i>	- 0xC03F
<i>TLS_DH_RSA_WITH_ARIA_128_CBC_SHA256</i>	- 0xC040
<i>TLS_DH_RSA_WITH_ARIA_256_CBC_SHA384</i>	- 0xC041
<i>TLS_DHE_DSS_WITH_ARIA_128_CBC_SHA256</i>	- 0xC042

<i>TLS_DHE_DSS_-_WITH_ARIA_-_256_CBC_SH-A384</i>	- 0xC043
<i>TLS_DHE_RSA_-_WITH_ARIA_-_128_CBC_SH-A256</i>	- 0xC044
<i>TLS_DHE_RSA_-_WITH_ARIA_-_256_CBC_SH-A384</i>	- 0xC045
<i>TLS_DH_ANO-N_WITH_ARIA_-_128_CBC_SH-A256</i>	- 0xC046
<i>TLS_DH_ANO-N_WITH_ARIA_-_256_CBC_SH-A384</i>	- 0xC047
<i>TLS_ECDHE_E-CDSA_WITH_ARIA_128_CBC_-SHA256</i>	- 0xC048
<i>TLS_ECDHE_E-CDSA_WITH_ARIA_256_CBC_-SHA384</i>	- 0xC049
<i>TLS_ECDH_EC-DSA_WITH_ARIA_128_CBC_S-HA256</i>	- 0xC04A
<i>TLS_ECDH_EC-DSA_WITH_ARIA_256_CBC_S-HA384</i>	- 0xC04B
<i>TLS_ECDHE_RSA_WITH_ARIA_128_CBC_S-HA256</i>	- 0xC04C
<i>TLS_ECDHE_RSA_WITH_ARIA_256_CBC_S-HA384</i>	- 0xC04D
<i>TLS_ECDH_RSA_WITH_ARIA_-_128_CBC_SH-A256</i>	- 0xC04E
<i>TLS_ECDH_RSA_WITH_ARIA_-_256_CBC_SH-A384</i>	- 0xC04F

<code>TLS_RSA_WITH_ARIA_128_GCM_SHA256</code>	- 0xC050
<code>TLS_RSA_WITH_ARIA_256_GCM_SHA384</code>	- 0xC051
<code>TLS_DHE_RSA_WITH_ARIA_128_GCM_SHA256</code>	- 0xC052
<code>TLS_DHE_RSA_WITH_ARIA_256_GCM_SHA384</code>	- 0xC053
<code>TLS_DH_RSA_WITH_ARIA_128_GCM_SHA256</code>	- 0xC054
<code>TLS_DH_RSA_WITH_ARIA_256_GCM_SHA384</code>	- 0xC055
<code>TLS_DHE_DSS_WITH_ARIA_128_GCM_SHA256</code>	- 0xC056
<code>TLS_DHE_DSS_WITH_ARIA_256_GCM_SHA384</code>	- 0xC057
<code>TLS_DH_DSS_WITH_ARIA_128_GCM_SHA256</code>	- 0xC058
<code>TLS_DH_DSS_WITH_ARIA_256_GCM_SHA384</code>	- 0xC059
<code>TLS_DH_ANON_WITH_ARIA_128_GCM_SHA256</code>	- 0xC05A
<code>TLS_DH_ANON_WITH_ARIA_256_GCM_SHA384</code>	- 0xC05B
<code>TLS_ECDHE_ECDSA_WITH_ARIA_128_GCM_SHA256</code>	- 0xC05C

<i>TLS_ECDHE_E- CDSA_WITH_AR- IA_256_GCM- SHA384</i>	- 0xC05D
<i>TLS_ECDH_EC- DSA_WITH_AR- IA_128_GCM- SHA256</i>	- 0xC05E
<i>TLS_ECDH_EC- DSA_WITH_AR- IA_256_GCM- SHA384</i>	- 0xC05F
<i>TLS_ECDHE_RS- SA_WITH_ARI- A_128_GCM_S- HA256</i>	- 0xC060
<i>TLS_ECDHE_RS- SA_WITH_ARI- A_256_GCM_S- HA384</i>	- 0xC061
<i>TLS_ECDH_RS- A_WITH_ARIA_- 128_GCM_SH- A256</i>	- 0xC062
<i>TLS_ECDH_RS- A_WITH_ARIA_- 256_GCM_SH- A384</i>	- 0xC063
<i>TLS_PSK_WIT- H_ARIA_128_C- BC_SHA256</i>	- 0xC064
<i>TLS_PSK_WIT- H_ARIA_256_C- BC_SHA384</i>	- 0xC065
<i>TLS_DHE_PSK- _WITH_ARIA_- 128_CBC_SH- A256</i>	- 0xC066
<i>TLS_DHE_PSK- _WITH_ARIA_- 256_CBC_SH- A384</i>	- 0xC067
<i>TLS_RSA_PSK- _WITH_ARIA_- 128_CBC_SH- A256</i>	- 0xC068
<i>TLS_RSA_PSK- _WITH_ARIA_- 256_CBC_SH- A384</i>	- 0xC069

<i>TLS_PSK_WIT-H_ARIA_128_G-CM_SHA256</i>	- 0xC06A
<i>TLS_PSK_WIT-H_ARIA_256_G-CM_SHA384</i>	- 0xC06B
<i>TLS_DHE_PSK-WITH_ARIA-128_GCM_SHA256</i>	- 0xC06C
<i>TLS_DHE_PSK-WITH_ARIA-256_GCM_SHA384</i>	- 0xC06D
<i>TLS_RSA_PSK-WITH_ARIA-128_GCM_SHA256</i>	- 0xC06E
<i>TLS_RSA_PSK-WITH_ARIA-256_GCM_SHA384</i>	- 0xC06F
<i>TLS_ECDHE_PSK_WITH_ARIA_128_CBC_SHA256</i>	- 0xC070
<i>TLS_ECDHE_PSK_WITH_ARIA_256_CBC_SHA384</i>	- 0xC071
<i>TLS_ECDHE_ECDDSA_WITH_CAMELLIA_128_CBC_SHA256</i>	- 0xC072
<i>TLS_ECDHE_ECDDSA_WITH_CAMELLIA_256_CBC_SHA384</i>	- 0xC073
<i>TLS_ECDH_ECDSA_WITH_CAMELLIA_128_CBC_SHA256</i>	- 0xC074
<i>TLS_ECDH_ECDSA_WITH_CAMELLIA_256_CBC_SHA384</i>	- 0xC075
<i>TLS_ECDHE_RSA_WITH_CAMELLIA_128_CBC_SHA256</i>	- 0xC076

<i>TLS_ECDHE_RSA_WITH_CAMELLIA_256_CBC_SHA384</i>	- 0xC077
<i>TLS_ECDH_RSA_WITH_CAMELLIA_128_CBC_SHA256</i>	- 0xC078
<i>TLS_ECDH_RSA_WITH_CAMELLIA_256_CBC_SHA384</i>	- 0xC079
<i>TLS_RSA_WITH_CAMELLIA_128_GCM_SHA256</i>	- 0xC07A
<i>TLS_RSA_WITH_CAMELLIA_256_GCM_SHA384</i>	- 0xC07B
<i>TLS_DHE_RSA_WITH_CAMELLIA_128_GCM_SHA256</i>	- 0xC07C
<i>TLS_DHE_RSA_WITH_CAMELLIA_256_GCM_SHA384</i>	- 0xC07D
<i>TLS_DH_RSA_WITH_CAMELLIA_128_GCM_SHA256</i>	- 0xC07E
<i>TLS_DH_RSA_WITH_CAMELLIA_256_GCM_SHA384</i>	- 0xC07F
<i>TLS_DHE_DSS_WITH_CAMELLIA_128_GCM_SHA256</i>	- 0xC080
<i>TLS_DHE_DSS_WITH_CAMELLIA_256_GCM_SHA384</i>	- 0xC081
<i>TLS_DH_DSS_WITH_CAMELLIA_128_GCM_SHA256</i>	- 0xC082
<i>TLS_DH_DSS_WITH_CAMELLIA_256_GCM_SHA384</i>	- 0xC083

<i>TLS_DH_ANO-N_WITH_CAM-ELLIA_128_GCM_SHA256</i>	- 0xC084
<i>TLS_DH_ANO-N_WITH_CAM-ELLIA_256_GCM_SHA384</i>	- 0xC085
<i>TLS_ECDHE_E-CDSA_WITH_C-AMELLIA_128_-GCM_SHA256</i>	- 0xC086
<i>TLS_ECDHE_E-CDSA_WITH_C-AMELLIA_256_-GCM_SHA384</i>	- 0xC087
<i>TLS_ECDH_EC-DSA_WITH_CA-MELLIA_128_G-CM_SHA256</i>	- 0xC088
<i>TLS_ECDH_EC-DSA_WITH_CA-MELLIA_256_G-CM_SHA384</i>	- 0xC089
<i>TLS_ECDHE_RSA_WITH_CAM-ELLIA_128_GCM_SHA256</i>	- 0xC08A
<i>TLS_ECDHE_RSA_WITH_CAM-ELLIA_256_GCM_SHA384</i>	- 0xC08B
<i>TLS_ECDH_RSA_WITH_CAMELLIA_128_GCM_SHA256</i>	- 0xC08C
<i>TLS_ECDH_RSA_WITH_CAMELLIA_256_GCM_SHA384</i>	- 0xC08D
<i>TLS_PSK_WITH_CAMELLIA_128_GCM_SHA256</i>	- 0xC08E
<i>TLS_PSK_WITH_CAMELLIA_256_GCM_SHA384</i>	- 0xC08F
<i>TLS_DHE_PSK_WITH_CAMELLIA_128_GCM_SHA256</i>	- 0xC090

<code>TLS_DHE_PSK- _WITH_CAMEL- LIA_256_GCM_- SHA384</code>	- 0xC091
<code>TLS_RSA_PSK- _WITH_CAMEL- LIA_128_GCM_- SHA256</code>	- 0xC092
<code>TLS_RSA_PSK- _WITH_CAMEL- LIA_256_GCM_- SHA384</code>	- 0xC093
<code>TLS_PSK_WIT- H_CAMELLIA_- 128_CBC_SH- A256</code>	- 0xC094
<code>TLS_PSK_WIT- H_CAMELLIA_- 256_CBC_SH- A384</code>	- 0xC095
<code>TLS_DHE_PSK- _WITH_CAMEL- LIA_128_CBC_- SHA256</code>	- 0xC096
<code>TLS_DHE_PSK- _WITH_CAMEL- LIA_256_CBC_- SHA384</code>	- 0xC097
<code>TLS_RSA_PSK- _WITH_CAMEL- LIA_128_CBC_- SHA256</code>	- 0xC098
<code>TLS_RSA_PSK- _WITH_CAMEL- LIA_256_CBC_- SHA384</code>	- 0xC099
<code>TLS_ECDHE_P- SK_WITH_CAM- ELLIA_128_CB- C_SHA256</code>	- 0xC09A
<code>TLS_ECDHE_P- SK_WITH_CAM- ELLIA_256_CB- C_SHA384</code>	- 0xC09B
<code>TLS_RSA_WIT- H_AES_128_C- CM</code>	- 0xC09C
<code>TLS_RSA_WIT- H_AES_256_C- CM</code>	- 0xC09D

<i>TLS_DHE_RSA- _WITH_AES_- 128_CCM</i>	- 0xC09E
<i>TLS_DHE_RSA- _WITH_AES_- 256_CCM</i>	- 0xC09F
<i>TLS_RSA_WIT- H_AES_128_C- CM_8</i>	- 0xC0A0
<i>TLS_RSA_WIT- H_AES_256_C- CM_8</i>	- 0xC0A1
<i>TLS_DHE_RSA- _WITH_AES_- 128_CCM_8</i>	- 0xC0A2
<i>TLS_DHE_RSA- _WITH_AES_- 256_CCM_8</i>	- 0xC0A3
<i>TLS_PSK_WIT- H_AES_128_C- CM</i>	- 0xC0A4
<i>TLS_PSK_WIT- H_AES_256_C- CM</i>	- 0xC0A5
<i>TLS_DHE_PSK- _WITH_AES_- 128_CCM</i>	- 0xC0A6
<i>TLS_DHE_PSK- _WITH_AES_- 256_CCM</i>	- 0xC0A7
<i>TLS_PSK_WIT- H_AES_128_C- CM_8</i>	- 0xC0A8
<i>TLS_PSK_WIT- H_AES_256_C- CM_8</i>	- 0xC0A9
<i>TLS_PSK_DHE- _WITH_AES_- 128_CCM_8</i>	- 0xC0AA
<i>TLS_PSK_DHE- _WITH_AES_- 256_CCM_8</i>	- 0xC0AB
<i>TLS_ECDHE_E- CDSA_WITH_A- ES_128_CCM</i>	- 0xC0AC
<i>TLS_ECDHE_E- CDSA_WITH_A- ES_256_CCM</i>	- 0xC0AD
<i>TLS_ECDHE_E- CDSA_WITH_A- ES_128_CCM_8</i>	- 0xC0AE

<code>TLS_ECDHE_E-CDSA_WITH_AES_256_CCM_8</code>	- 0xC0AF
<code>TLS_ECDHE_RSA_WITH_CHACHA20_POLY1305_SHA256</code>	- 0xCCA8
<code>TLS_ECDHE_E-CDSA_WITH_CHACHA20_POLY1305_SHA256</code>	- 0CCA9
<code>TLS_DHE_RSA_WITH_CHACHA20_POLY1305_SHA256</code>	- 0CCAA
<code>TLS_PSK_WITH_CHACHA20_POLY1305_SHA256</code>	- 0CCAB
<code>TLS_ECDHE_PSK_WITH_CHACHA20_POLY1305_SHA256</code>	- 0CCAC
<code>TLS_DHE_PSK_WITH_CHACHA20_POLY1305_SHA256</code>	- 0CCAD
<code>TLS_RSA_PSK_WITH_CHACHA20_POLY1305_SHA256</code>	- 0CCAE

Enumerator

`TLS_NULL_WITH_NULL_NULL` NULL Encryption and NULL Authentication.

`TLS_RSA_WITH_NULL_MD5` RSA Key Negotiation with NULL Encryption and MD5 Authentication and PRF.

`TLS_RSA_WITH_NULL_SHA` RSA Key Negotiation with NULL Encryption and SHA1 Authentication and PRF.

`TLS_RSA_EXPORT_WITH_RC4_40_MD5` RSA Key Negotiation with RC4 Encryption 40-bit Key and MD5 Authentication PRF.

`TLS_RSA_WITH_RC4_128_MD5` RSA Key Negotiation with RC4 Encryption 128-bit Key and MD5 Authentication PRF.

`TLS_RSA_WITH_RC4_128_SHA` RSA Key Negotiation with RC4 Encryption 128-bit Key and SHA1 Authentication PRF.

`TLS_RSA_EXPORT_WITH_DES40_CBC_SHA` RSA Key Negotiation with RC2-CBC Encryption 40-bit Key and MD5 Authentication PRF.

`TLS_RSA_WITH DES_CBC_SHA`

`TLS_RSA_WITH_3DES_EDE_CBC_SHA`

`TLS_DH_DSS_EXPORT_WITH_DES40_CBC_SHA`

`TLS_DH_DSS_WITH_DES_CBC_SHA`

`TLS_DH_DSS_WITH_3DES_EDE_CBC_SHA`

`TLS_DH_RSA_EXPORT_WITH_DES40_CBC_SHA`

`TLS_DH_RSA_WITH_DES_CBC_SHA`

*TLS_DH_RSA_WITH_3DES_EDE_CBC_SHA
TLS_DHE_DSS_EXPORT_WITH_DES40_CBC_SHA
TLS_DHE_DSS_WITH_DES_CBC_SHA
TLS_DHE_DSS_WITH_3DES_EDE_CBC_SHA
TLS_DHE_RSA_EXPORT_WITH_DES40_CBC_SHA
TLS_DHE_RSA_WITH_DES_CBC_SHA
TLS_DHE_RSA_WITH_3DES_EDE_CBC_SHA
TLS_DH_ANON_EXPORT_WITH_RC4_40_MD5
TLS_DH_ANON_WITH_RC4_128_MD5
TLS_DH_ANON_EXPORT_WITH_DES40_CBC_SHA
TLS_DH_ANON_WITH_DES_CBC_SHA
TLS_DH_ANON_WITH_3DES_EDE_CBC_SHA
TLS_KRB5_WITH_DES_CBC_SHA
TLS_KRB5_WITH_3DES_EDE_CBC_SHA
TLS_KRB5_WITH_RC4_128_SHA
TLS_KRB5_WITH_DES_CBC_MD5
TLS_KRB5_WITH_3DES_EDE_CBC_MD5
TLS_KRB5_WITH_RC4_128_MD5
TLS_KRB5_EXPORT_WITH_DES_CBC_40_SHA
TLS_KRB5_EXPORT_WITH_RC4_40_SHA
TLS_KRB5_EXPORT_WITH_DES_CBC_40_MD5
TLS_KRB5_EXPORT_WITH_RC4_40_MD5
TLS_PSK_WITH_NULL_SHA
TLS_DHE_PSK_WITH_NULL_SHA
TLS_RSA_PSK_WITH_NULL_SHA
TLS_RSA_WITH_AES_128_CBC_SHA
TLS_DH_DSS_WITH_AES_128_CBC_SHA
TLS_DH_RSA_WITH_AES_128_CBC_SHA
TLS_DHE_DSS_WITH_AES_128_CBC_SHA
TLS_DHE_RSA_WITH_AES_128_CBC_SHA
TLS_DH_ANON_WITH_AES_128_CBC_SHA
TLS_RSA_WITH_AES_256_CBC_SHA
TLS_DH_DSS_WITH_AES_256_CBC_SHA
TLS_DH_RSA_WITH_AES_256_CBC_SHA
TLS_DHE_DSS_WITH_AES_256_CBC_SHA
TLS_DHE_RSA_WITH_AES_256_CBC_SHA
TLS_DH_ANON_WITH_AES_256_CBC_SHA
TLS_RSA_WITH_NULL_SHA256
TLS_RSA_WITH_AES_128_CBC_SHA256
TLS_RSA_WITH_AES_256_CBC_SHA256
TLS_DH_DSS_WITH_AES_128_CBC_SHA256
TLS_DH_RSA_WITH_AES_128_CBC_SHA256
TLS_DHE_DSS_WITH_AES_128_CBC_SHA256
TLS_RSA_WITH_CAMELLIA_128_CBC_SHA
TLS_DH_DSS_WITH_CAMELLIA_128_CBC_SHA*

*TLS_DH_RSA_WITH_CAMELLIA_128_CBC_SHA
TLS_DHE_DSS_WITH_CAMELLIA_128_CBC_SHA
TLS_DHE_RSA_WITH_CAMELLIA_128_CBC_SHA
TLS_DH_ANON_WITH_CAMELLIA_128_CBC_SHA
TLS_DHE_RSA_WITH_AES_128_CBC_SHA256
TLS_DH_DSS_WITH_AES_256_CBC_SHA256
TLS_DH_RSA_WITH_AES_256_CBC_SHA256
TLS_DHE_DSS_WITH_AES_256_CBC_SHA256
TLS_DHE_RSA_WITH_AES_256_CBC_SHA256
TLS_DH_ANON_WITH_AES_128_CBC_SHA256
TLS_DH_ANON_WITH_AES_256_CBC_SHA256
TLS_RSA_WITH_CAMELLIA_256_CBC_SHA
TLS_DH_DSS_WITH_CAMELLIA_256_CBC_SHA
TLS_DH_RSA_WITH_CAMELLIA_256_CBC_SHA
TLS_DHE_DSS_WITH_CAMELLIA_256_CBC_SHA
TLS_DHE_RSA_WITH_CAMELLIA_256_CBC_SHA
TLS_DH_ANON_WITH_CAMELLIA_256_CBC_SHA
TLS_PSK_WITH_RC4_128_SHA
TLS_PSK_WITH_3DES_EDE_CBC_SHA
TLS_PSK_WITH_AES_128_CBC_SHA
TLS_PSK_WITH_AES_256_CBC_SHA
TLS_DHE_PSK_WITH_RC4_128_SHA
TLS_DHE_PSK_WITH_3DES_EDE_CBC_SHA
TLS_DHE_PSK_WITH_AES_128_CBC_SHA
TLS_DHE_PSK_WITH_AES_256_CBC_SHA
TLS_RSA_PSK_WITH_RC4_128_SHA
TLS_RSA_PSK_WITH_3DES_EDE_CBC_SHA
TLS_RSA_PSK_WITH_AES_128_CBC_SHA
TLS_RSA_PSK_WITH_AES_256_CBC_SHA
TLS_RSA_WITH_SEED_CBC_SHA
TLS_DH_DSS_WITH_SEED_CBC_SHA
TLS_DH_RSA_WITH_SEED_CBC_SHA
TLS_DHE_DSS_WITH_SEED_CBC_SHA
TLS_DHE_RSA_WITH_SEED_CBC_SHA
TLS_DH_ANON_WITH_SEED_CBC_SHA
TLS_RSA_WITH_AES_128_GCM_SHA256
TLS_RSA_WITH_AES_256_GCM_SHA384
TLS_DHE_RSA_WITH_AES_128_GCM_SHA256
TLS_DHE_RSA_WITH_AES_256_GCM_SHA384
TLS_DH_RSA_WITH_AES_128_GCM_SHA256
TLS_DH_RSA_WITH_AES_256_GCM_SHA384
TLS_DHE_DSS_WITH_AES_128_GCM_SHA256
TLS_DHE_DSS_WITH_AES_256_GCM_SHA384
TLS_DH_DSS_WITH_AES_128_GCM_SHA256
TLS_DH_DSS_WITH_AES_256_GCM_SHA384*

*TLS_DH_ANON_WITH_AES_128_GCM_SHA256
TLS_DH_ANON_WITH_AES_256_GCM_SHA384
TLS_PSK_WITH_AES_128_GCM_SHA256
TLS_PSK_WITH_AES_256_GCM_SHA384
TLS_DHE_PSK_WITH_AES_128_GCM_SHA256
TLS_DHE_PSK_WITH_AES_256_GCM_SHA384
TLS_RSA_PSK_WITH_AES_128_GCM_SHA256
TLS_RSA_PSK_WITH_AES_256_GCM_SHA384
TLS_PSK_WITH_AES_128_CBC_SHA256
TLS_PSK_WITH_AES_256_CBC_SHA384
TLS_PSK_WITH_NULL_SHA256
TLS_PSK_WITH_NULL_SHA384
TLS_DHE_PSK_WITH_AES_128_CBC_SHA256
TLS_DHE_PSK_WITH_AES_256_CBC_SHA384
TLS_DHE_PSK_WITH_NULL_SHA256
TLS_DHE_PSK_WITH_NULL_SHA384
TLS_RSA_PSK_WITH_AES_128_CBC_SHA256
TLS_RSA_PSK_WITH_AES_256_CBC_SHA384
TLS_RSA_PSK_WITH_NULL_SHA256
TLS_RSA_PSK_WITH_NULL_SHA384
TLS_RSA_WITH_CAMELLIA_128_CBC_SHA256
TLS_DH_DSS_WITH_CAMELLIA_128_CBC_SHA256
TLS_DH_RSA_WITH_CAMELLIA_128_CBC_SHA256
TLS_DHE_DSS_WITH_CAMELLIA_128_CBC_SHA256
TLS_DHE_RSA_WITH_CAMELLIA_128_CBC_SHA256
TLS_DH_ANON_WITH_CAMELLIA_128_CBC_SHA256
TLS_RSA_WITH_CAMELLIA_256_CBC_SHA256
TLS_DH_DSS_WITH_CAMELLIA_256_CBC_SHA256
TLS_DH_RSA_WITH_CAMELLIA_256_CBC_SHA256
TLS_DHE_DSS_WITH_CAMELLIA_256_CBC_SHA256
TLS_DHE_RSA_WITH_CAMELLIA_256_CBC_SHA256
TLS_DH_ANON_WITH_CAMELLIA_256_CBC_SHA256
TLS_ECDH_ECDSA_WITH_NULL_SHA
TLS_ECDH_ECDSA_WITH_RC4_128_SHA
TLS_ECDH_ECDSA_WITH_3DES_EDE_CBC_SHA
TLS_ECDH_ECDSA_WITH_AES_128_CBC_SHA
TLS_ECDH_ECDSA_WITH_AES_256_CBC_SHA
TLS_ECDHE_ECDSA_WITH_NULL_SHA
TLS_ECDHE_ECDSA_WITH_RC4_128_SHA
TLS_ECDHE_ECDSA_WITH_3DES_EDE_CBC_SHA
TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA
TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA
TLS_ECDH_RSA_WITH_NULL_SHA
TLS_ECDH_RSA_WITH_RC4_128_SHA
TLS_ECDH_RSA_WITH_3DES_EDE_CBC_SHA*

*TLS_ECDH_RSA_WITH_AES_128_CBC_SHA
TLS_ECDH_RSA_WITH_AES_256_CBC_SHA
TLS_ECDHE_RSA_WITH_NULL_SHA
TLS_ECDHE_RSA_WITH_RC4_128_SHA
TLS_ECDHE_RSA_WITH_3DES_EDE_CBC_SHA
TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA
TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA
TLS_ECDH_ANON_WITH_NULL_SHA
TLS_ECDH_ANON_WITH_RC4_128_SHA
TLS_ECDH_ANON_WITH_3DES_EDE_CBC_SHA
TLS_ECDH_ANON_WITH_AES_128_CBC_SHA
TLS_ECDH_ANON_WITH_AES_256_CBC_SHA
TLS_SRP_SHA_WITH_3DES_EDE_CBC_SHA
TLS_SRP_SHA_RSA_WITH_3DES_EDE_CBC_SHA
TLS_SRP_SHA_DSS_WITH_3DES_EDE_CBC_SHA
TLS_SRP_SHA_WITH_AES_128_CBC_SHA
TLS_SRP_SHA_RSA_WITH_AES_128_CBC_SHA
TLS_SRP_SHA_DSS_WITH_AES_128_CBC_SHA
TLS_SRP_SHA_WITH_AES_256_CBC_SHA
TLS_SRP_SHA_RSA_WITH_AES_256_CBC_SHA
TLS_SRP_SHA_DSS_WITH_AES_256_CBC_SHA
TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256
TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384
TLS_ECDH_ECDSA_WITH_AES_128_CBC_SHA256
TLS_ECDH_ECDSA_WITH_AES_256_CBC_SHA384
TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256
TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384
TLS_ECDH_RSA_WITH_AES_128_CBC_SHA256
TLS_ECDH_RSA_WITH_AES_256_CBC_SHA384
TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256
TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384
TLS_ECDH_ECDSA_WITH_AES_128_GCM_SHA256
TLS_ECDH_ECDSA_WITH_AES_256_GCM_SHA384
TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256
TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384
TLS_ECDH_RSA_WITH_AES_128_GCM_SHA256
TLS_ECDH_RSA_WITH_AES_256_GCM_SHA384
TLS_ECDHE_PSK_WITH_RC4_128_SHA
TLS_ECDHE_PSK_WITH_3DES_EDE_CBC_SHA
TLS_ECDHE_PSK_WITH_AES_128_CBC_SHA
TLS_ECDHE_PSK_WITH_AES_256_CBC_SHA
TLS_ECDHE_PSK_WITH_AES_128_CBC_SHA256
TLS_ECDHE_PSK_WITH_AES_256_CBC_SHA384
TLS_ECDHE_PSK_WITH_NULL_SHA
TLS_ECDHE_PSK_WITH_NULL_SHA256*

TLS_ECDHE_PSK_WITH_NULL_SHA384
TLS_RSA_WITH_ARIA_128_CBC_SHA256
TLS_RSA_WITH_ARIA_256_CBC_SHA384
TLS_DH_DSS_WITH_ARIA_128_CBC_SHA256
TLS_DH_DSS_WITH_ARIA_256_CBC_SHA384
TLS_DH_RSA_WITH_ARIA_128_CBC_SHA256
TLS_DH_RSA_WITH_ARIA_256_CBC_SHA384
TLS_DHE_DSS_WITH_ARIA_128_CBC_SHA256
TLS_DHE_DSS_WITH_ARIA_256_CBC_SHA384
TLS_DHE_RSA_WITH_ARIA_128_CBC_SHA256
TLS_DHE_RSA_WITH_ARIA_256_CBC_SHA384
TLS_DH_ANON_WITH_ARIA_128_CBC_SHA256
TLS_DH_ANON_WITH_ARIA_256_CBC_SHA384
TLS_ECDHE_ECDSA_WITH_ARIA_128_CBC_SHA256
TLS_ECDHE_ECDSA_WITH_ARIA_256_CBC_SHA384
TLS_ECDH_ECDSA_WITH_ARIA_128_CBC_SHA256
TLS_ECDH_ECDSA_WITH_ARIA_256_CBC_SHA384
TLS_ECDHE_RSA_WITH_ARIA_128_CBC_SHA256
TLS_ECDHE_RSA_WITH_ARIA_256_CBC_SHA384
TLS_ECDH_RSA_WITH_ARIA_128_CBC_SHA256
TLS_ECDH_RSA_WITH_ARIA_256_CBC_SHA384
TLS_RSA_WITH_ARIA_128_GCM_SHA256
TLS_RSA_WITH_ARIA_256_GCM_SHA384
TLS_DHE_RSA_WITH_ARIA_128_GCM_SHA256
TLS_DHE_RSA_WITH_ARIA_256_GCM_SHA384
TLS_DH_RSA_WITH_ARIA_128_GCM_SHA256
TLS_DH_RSA_WITH_ARIA_256_GCM_SHA384
TLS_DHE_DSS_WITH_ARIA_128_GCM_SHA256
TLS_DHE_DSS_WITH_ARIA_256_GCM_SHA384
TLS_DH_ANON_WITH_ARIA_128_GCM_SHA256
TLS_DH_ANON_WITH_ARIA_256_GCM_SHA384
TLS_ECDHE_ECDSA_WITH_ARIA_128_GCM_SHA256
TLS_ECDHE_ECDSA_WITH_ARIA_256_GCM_SHA384
TLS_ECDH_ECDSA_WITH_ARIA_128_GCM_SHA256
TLS_ECDH_ECDSA_WITH_ARIA_256_GCM_SHA384
TLS_ECDHE_RSA_WITH_ARIA_128_GCM_SHA256
TLS_ECDHE_RSA_WITH_ARIA_256_GCM_SHA384
TLS_ECDH_RSA_WITH_ARIA_128_GCM_SHA256
TLS_ECDH_RSA_WITH_ARIA_256_GCM_SHA384
TLS_PSK_WITH_ARIA_128_CBC_SHA256
TLS_PSK_WITH_ARIA_256_CBC_SHA384
TLS_DHE_PSK_WITH_ARIA_128_CBC_SHA256
TLS_DHE_PSK_WITH_ARIA_256_CBC_SHA384

TLS_RSA_PSK_WITH_ARIA_128_CBC_SHA256
TLS_RSA_PSK_WITH_ARIA_256_CBC_SHA384
TLS_PSK_WITH_ARIA_128_GCM_SHA256
TLS_PSK_WITH_ARIA_256_GCM_SHA384
TLS_DHE_PSK_WITH_ARIA_128_GCM_SHA256
TLS_DHE_PSK_WITH_ARIA_256_GCM_SHA384
TLS_RSA_PSK_WITH_ARIA_128_GCM_SHA256
TLS_RSA_PSK_WITH_ARIA_256_GCM_SHA384
TLS_ECDHE_PSK_WITH_ARIA_128_CBC_SHA256
TLS_ECDHE_PSK_WITH_ARIA_256_CBC_SHA384
TLS_ECDHE_ECDSA_WITH_CAMELLIA_128_CBC_SHA256
TLS_ECDHE_ECDSA_WITH_CAMELLIA_256_CBC_SHA384
TLS_ECDH_ECDSA_WITH_CAMELLIA_128_CBC_SHA256
TLS_ECDH_ECDSA_WITH_CAMELLIA_256_CBC_SHA384
TLS_ECDHE_RSA_WITH_CAMELLIA_128_CBC_SHA256
TLS_ECDHE_RSA_WITH_CAMELLIA_256_CBC_SHA384
TLS_ECDH_RSA_WITH_CAMELLIA_128_CBC_SHA256
TLS_ECDH_RSA_WITH_CAMELLIA_256_CBC_SHA384
TLS_RSA_WITH_CAMELLIA_128_GCM_SHA256
TLS_RSA_WITH_CAMELLIA_256_GCM_SHA384
TLS_DHE_RSA_WITH_CAMELLIA_128_GCM_SHA256
TLS_DHE_RSA_WITH_CAMELLIA_256_GCM_SHA384
TLS_DH_RSA_WITH_CAMELLIA_128_GCM_SHA256
TLS_DH_RSA_WITH_CAMELLIA_256_GCM_SHA384
TLS_DHE_DSS_WITH_CAMELLIA_128_GCM_SHA256
TLS_DHE_DSS_WITH_CAMELLIA_256_GCM_SHA384
TLS_DH_DSS_WITH_CAMELLIA_128_GCM_SHA256
TLS_DH_DSS_WITH_CAMELLIA_256_GCM_SHA384
TLS_DH_ANON_WITH_CAMELLIA_128_GCM_SHA256
TLS_DH_ANON_WITH_CAMELLIA_256_GCM_SHA384
TLS_ECDHE_ECDSA_WITH_CAMELLIA_128_GCM_SHA256
TLS_ECDHE_ECDSA_WITH_CAMELLIA_256_GCM_SHA384
TLS_ECDH_ECDSA_WITH_CAMELLIA_128_GCM_SHA256
TLS_ECDH_ECDSA_WITH_CAMELLIA_256_GCM_SHA384
TLS_ECDHE_RSA_WITH_CAMELLIA_128_GCM_SHA256
TLS_ECDHE_RSA_WITH_CAMELLIA_256_GCM_SHA384
TLS_ECDH_RSA_WITH_CAMELLIA_128_GCM_SHA256
TLS_ECDH_RSA_WITH_CAMELLIA_256_GCM_SHA384
TLS_PSK_WITH_CAMELLIA_128_GCM_SHA256
TLS_PSK_WITH_CAMELLIA_256_GCM_SHA384
TLS_DHE_PSK_WITH_CAMELLIA_128_GCM_SHA256
TLS_DHE_PSK_WITH_CAMELLIA_256_GCM_SHA384
TLS_RSA_PSK_WITH_CAMELLIA_128_GCM_SHA256
TLS_RSA_PSK_WITH_CAMELLIA_256_GCM_SHA384
TLS_PSK_WITH_CAMELLIA_128_CBC_SHA256

TLS_PSK_WITH_CAMELLIA_256_CBC_SHA384
TLS_DHE_PSK_WITH_CAMELLIA_128_CBC_SHA256
TLS_DHE_PSK_WITH_CAMELLIA_256_CBC_SHA384
TLS_RSA_PSK_WITH_CAMELLIA_128_CBC_SHA256
TLS_RSA_PSK_WITH_CAMELLIA_256_CBC_SHA384
TLS_ECDHE_PSK_WITH_CAMELLIA_128_CBC_SHA256
TLS_ECDHE_PSK_WITH_CAMELLIA_256_CBC_SHA384
TLS_RSA_WITH_AES_128_CCM
TLS_RSA_WITH_AES_256_CCM
TLS_DHE_RSA_WITH_AES_128_CCM
TLS_DHE_RSA_WITH_AES_256_CCM
TLS_RSA_WITH_AES_128_CCM_8
TLS_RSA_WITH_AES_256_CCM_8
TLS_DHE_RSA_WITH_AES_128_CCM_8
TLS_DHE_RSA_WITH_AES_256_CCM_8
TLS_PSK_WITH_AES_128_CCM
TLS_PSK_WITH_AES_256_CCM
TLS_DHE_PSK_WITH_AES_128_CCM
TLS_DHE_PSK_WITH_AES_256_CCM
TLS_PSK_WITH_AES_128_CCM_8
TLS_PSK_WITH_AES_256_CCM_8
TLS_PSK_DHE_WITH_AES_128_CCM_8
TLS_PSK_DHE_WITH_AES_256_CCM_8
TLS_ECDHE_ECDSA_WITH_AES_128_CCM
TLS_ECDHE_ECDSA_WITH_AES_256_CCM
TLS_ECDHE_ECDSA_WITH_AES_128_CCM_8
TLS_ECDHE_ECDSA_WITH_AES_256_CCM_8
TLS_ECDHE_RSA_WITH_CHACHA20_POLY1305_SHA256
TLS_ECDHE_ECDSA_WITH_CHACHA20_POLY1305_SHA256
TLS_DHE_RSA_WITH_CHACHA20_POLY1305_SHA256
TLS_PSK_WITH_CHACHA20_POLY1305_SHA256
TLS_ECDHE_PSK_WITH_CHACHA20_POLY1305_SHA256
TLS_DHE_PSK_WITH_CHACHA20_POLY1305_SHA256
TLS_RSA_PSK_WITH_CHACHA20_POLY1305_SHA256

6.5.18.29 enum ProtocolPP::tls_error_t

ERROR types for TLS/SSL

Parameters

CLOSE	- Close the connection
MESSAGE	FATAL - Message was fatal

<i>BAD</i>	RECORD MAC - MAC for record was incorrect
<i>DECRYPT_FAT-AL</i>	- Decryption error
<i>RECORD</i>	OVERFLOW - Record overflow
<i>DECOMPRESS</i>	FAIL - Decompression failed
<i>HANDSHAKE</i>	FAIL - Handshake failed
<i>NO</i>	CERTIFICATE - Client has no certificate
<i>BAD</i>	CERTIFICATE - Client has bad certificate
<i>UNSUPPORTE-D</i>	CERTIFICATE - Wrong certificate type
<i>CERTIFICATE</i>	REVOKED - Certificate revoked
<i>CERTIFICATE</i>	EXPIRED - Certificate expired
<i>CERTIFICATE</i>	UNKNOWN - Unknown certificate used
<i>ILLEGAL</i>	PARAMETER - Illegal parameter in the packet
<i>UNKNOWN</i>	CA - Unknown certificate agency
<i>ACCESS</i>	DENIED - Access not allowed
<i>DECODE</i>	ERROR - Unable to decode the packet
<i>DECRYPT</i>	ERROR - Error during decryption
<i>EXPORT</i>	RESTRICTION - Export restriction on this packet
<i>PROTOCOL</i>	VERSION - Incorrect protocol version
<i>INSUFFICIENT</i>	SECURITY - Packet security was insufficient
<i>INTERNAL</i>	ERROR - Internal error during operation
<i>USER</i>	CANCELED - User cancelled connection
<i>NO</i>	RENEGOTIATION - Renegotiation failed
<i>UNSUPPORTE-D</i>	EXTENSION - Extension requested is unsupported
<i>CERTIFICATE</i>	UNOBTAINABLE - Required certificate was unobtainable
<i>UNRECOGNIZ-ED</i>	NAME - Unrecognized name
<i>BAD</i>	CERTIFICATE STATUS RESPONSE - Certificate status took too long
<i>BAD</i>	CERTIFICATE HASH VALUE - Certificate has been tampered with
<i>UNKNOWN</i>	PSK IDENTITY - ID unknown
<i>NO</i>	APPLICATION PROTOCOL - Incorrect Application protocol

Enumerator

```

CLOSE 0
MESSAGE_FATAL 10
BAD_RECORD_MAC 20
DECRYPT_FATAL 21
RECORD_OVERFLOW 22
DECOMPRESS_FAIL 30
HANDSHAKE_FAIL 40
NO_CERTIFICATE 41
BAD_CERTIFICATE 42
UNSUPPORTED_CERTIFICATE 43
CERTIFICATE_REVOKED 44
CERTIFICATE_EXPIRED 45
CERTIFICATE_UNKNOWN 46
ILLEGAL_PARAMETER 47
UNKNOWN_CA 48
ACCESS_DENIED 49
DECODE_ERROR 50

```

DECRYPT_ERROR 51
EXPORT_RESTRICTION 60
PROTOCOL_VERSION 70
INSUFFICIENT_SECURITY 71
INTERNAL_ERROR 80
USER_CANCELED 90
NO_RENEGOTIATION 100
UNSUPPORTED_EXTENSION 110
CERTIFICATE_UNOBTAINABLE 111
UNRECOGNIZED_NAME 112
BAD_CERTIFICATE_STATUS_RESPONSE 113
BAD_CERTIFICATE_HASH_VALUE 114
UNKNOWN_PSK_IDENTITY 115
NO_APPLICATION_PROTOCOL 120

6.5.18.30 enum ProtocolPP::tls_handshake_t

HANDSHAKE types for TLS/SSL

Parameters

<i>HELLO_REQUEST</i>	- Server request to open connection
<i>CLIENT_HELLO</i>	- Client request to open connection
<i>SERVER_HELLO</i>	- Server request to open connection
<i>NEW_SESSION_TICKET</i>	- Request to create new session
<i>CERTIFICATE</i>	- Certificate exchange
<i>SERVER_KEY_EXCHANGE</i>	- Server request to exchange keys
<i>CERTIFICATE_REQUEST</i>	- Request for certificate
<i>SERVER_HELLO_DONE</i>	- Server terminates connection request
<i>CERTIFICATE_VERIFY</i>	- Request to verify certificate
<i>CLIENT_KEY_EXCHANGE</i>	- Client request to exchange keys
<i>FINISHED</i>	- Request to end connection

Enumerator

HELLO_REQUEST Hello Request - 0.
CLIENT_HELLO Client Hello - 1.
SERVER_HELLO Server Hello - 2.
NEW_SESSION_TICKET New Session Ticket - 4.
CERTIFICATE Certificate - 11.
SERVER_KEY_EXCHANGE Server Key Exchange - 12.
CERTIFICATE_REQUEST Certificate Request - 13.
SERVER_HELLO_DONE Server Hello Done - 14.
CERTIFICATE_VERIFY Certificate Verify - 15.
CLIENT_KEY_EXCHANGE Client Key Exchange - 16.
FINISHED Finished - 20.

6.5.18.31 enum ProtocolPP::tlslvl_t

ALERT level values for TLS/SSL

Parameters

WARNING	- Warning level of ALERT for TLS
FATAL	- Fatal level of ALERT for TLS

Enumerator

TLSLVLO Reserved.

WARNING Warning alert - 2.

FATAL Fatal alert - 2.

6.5.18.32 enum ProtocolPP::tlstype_t

Types of packets for TLS/SSL

Parameters

CHG_CIPHER_- SPEC	- Change cipher spec
ALERT	- Alert packet for TLS
HANDSHAKE	- TLS handshake packet
APPLICATION	- TLS application packet
HEARTBEAT	- TLS keep alive packet

Enumerator

CHG_CIPHER_SPEC Change Cipher Spec - 0x14.

ALERT Alert message - 0x15.

HANDSHAKE Handshake message - 0x16.

APPLICATION Application message - 0x17.

HEARTBEAT Keep alive message - 0x18.

6.5.18.33 enum ProtocolPP::tlsver_t

Versions for TLS/SSL

Parameters

SSL30	- Secure Socket Layer v3.0
TLS10	- Transport Layer Security v1.0
TLS11	- Transport Layer Security v1.1
TLS12	- Transport Layer Security v1.2
DTLS	- Datagram Transport Layer Security

Enumerator

SSL30 Secure Socket Layer v3.0 - 0x3000.

TLS10 Transport Layer Security v1.0 - 0x0301.

TLS11 Transport Layer Security v1.1 - 0x0302.

TLS12 Transport Layer Security v1.2 - 0x0303.

DTLS Datagram Transport Layer Security - 0xFFFFE.

6.5.18.34 enum ProtocolPP::transform_attr_t

IKEv2 TRANSFORM ATTRIBUTE TYPES fields

Parameters

ATTR_KEY_LENGTH	- Key Length Attribute
------------------------	------------------------

Enumerator

ATTR_KEY_LENGTH Key Length Attribugte.

6.5.18.35 enum ProtocolPP::transform_type_t

IKEv2 TRANSFORM TYPES fields

Parameters

IKE_ENCR	- Encryption transform
IKE_INTEG	- Authentication transform
IKE_DH	- Diffie-Hellman transform
IKE_PRF	- Pseudo Random Function transform
IKE_ESN	- Extended Sequence Number transform

Enumerator

IKE_ENCR Encryption transform.

IKE_PRF Psuedo Random Function transform.

IKE_INTEG Authentication transform.

IKE_DH Diffie-Hellman transform.

IKE_ESN Extended Sequence Number transform.

6.5.18.36 enum ProtocolPP::wifictext_t

Wifi Control field extensions

Parameters

WCTRL0	- Reserved
SSW	- Wifi SSW extension
GRANT	- Wifi GRANT extension
WCTRL3	- Reserved
POLL	- Wifi POLL extension
SSWACK	- Wifi SSWACK extension
DMGCTS	- Wifi DMGCTS extension
WCTRL7	- Reserved
WCTRL8	- Reserved
SSWFDBCK	- Wifi SSWFDBCK extension
DMGACK	- Wifi DMGACK extension
WCTRL11	- Reserved
SPR	- Wifi SPR extension
WCTRL13	- Reserved
GRANTACK	- Wifi GRANTACK extension

Enumerator

WCTRL0 Reserved.

SSW Wifi SSW extension.

GRANT Wifi GRANT extension.

WCTRL3 Reserved.

POLL Wifi POLL extension.

SSWACK Wifi SSWACK extension.

DMGDT5 Wifi DMGDT5 extension.

WCTRL7 Reserved.

WCTRL8 Reserved.

SSWFDBCK Wifi SSWFDBCK extension.

DMGCTS Wifi DMGCTS extension.

WCTRL11 Reserved.

SPR Wifi SPR extension.

WCTRL13 Reserved.

GRANTACK Wifi GRANTACK extension.

6.5.18.37 enum ProtocolPP::wifisub_t

Wifi subtype field NOTE : bit ordering is opposite to spec (i.e., B4_B5_B6_B7) according to how they appear in the packet header such that WIFI1=0x8, WIFI5=0x2, etc. For example, if it's a management packet with ATM subtype you would use WIFI9. A control packet with RTS subtype would use WIFI11, etc

Due to the various meanings of the subtype based on the type, generic enums are provided for the subtype field

Parameters

WIFI0	- Management(WIFI0) Association Request / Control - RSVD / Data - Data
WIFI8	- Management(WIFI8) Beacon / Control - Block ACK Request (BlockAckReq) / Data - QoS Data
WIFI4	- Management(WIFI4) Probe Request / Control - RSVD / Data - Null (no data)
WIFI12	- Management(WIFI12) Deauthentication / Control - CTS / Data - QoS Null (no data)
WIFI2	- Management(WIFI2) Reassociation Request / Control - RSVD / Data - Data + CF-Poll
WIFI10	- Management(WIFI10) Disassociation / Control - PS-Poll / Data - QoS Data + CF-Poll
WIFI6	- Management(WIFI6) Timing Advertisement / Control - RSVD / Data - CF-Poll (no data)
WIFI14	- Management(WIFI14) Action No ACK / Control - CF-End / Data - QoS CF-Poll (no data)
WIFI1	- Management(WIFI1) Association Response / Control - RSVD / Data - Data + CF-Ack
WIFI9	- Management(WIFI9) ATM / Control - Block ACK (BlockAck) / Data - QoS Data + CF-Ack
WIFI5	- Management(WIFI5) Probe Response / Control - RSVD / Data - CF-Ack (no data)
WIFI13	- Management(WIFI13) Action / Control - ACK / Data - RSVD
WIFI3	- Management(WIFI3) Reassociation Response / Control - RSVD / Data - Data + CF-Ack + CF-Poll
WIFI11	- Management(WIFI11) Authentication / Control - RTS / Data - QoS Data + CF-Ack + CF-Poll
WIFI7	- Management(WIFI7) RSVD / Control - Control Wrapper / Data - CF-Ack + CF-Poll (no data)
WIFI15	- Management(WIFI15) RSVD / Control - CF-End + CF-Ack / Data - QoS CF-Ack + CF-Poll (no data)

Enumerator

WIFI0 Management(WIFI0) - Association Request / Control - RSVD / Data - Data.

WIFI8 Management(WIFI8) - Beacon / Control - Block ACK Request (BlockAckReq) / Data - QoS Data.

WIFI4 Management(WIFI4) - Probe Request / Control - RSVD / Data - Null (no data)

WIFI12 Management(WIFI12) - Deauthentication / Control - CTS / Data - QoS Null (no data)

WIFI2 Management(WIFI2) - Reassociation Request / Control - RSVD / Data - Data + CF-Poll.

WIFI10 Management(WIFI10) - Disassociation / Control - PS-Poll / Data - QoS Data + CF-Poll.

WIFI6 Management(WIFI6) - Timing Advertisement / Control - RSVD / Data - CF-Poll (no data)

WIFI14 Management(WIFI14) - Action No ACK / Control - CF-End / Data - QoS CF-Poll (no data)

- WIFI1** Management(WIFI1) - Association Response / Control - RSVD / Data - Data + CF-Ack.
- WIFI9** Management(WIFI9) - ATM / Control - Block ACK (BlockAck) / Data - QoS Data + CF-Ack.
- WIFI5** Management(WIFI5) - Probe Response / Control - RSVD / Data - CF-Ack (no data)
- WIFI13** Management(WIFI13) - Action / Control - ACK / Data - RSVD.
- WIFI3** Management(WIFI3) - Reassociation Response / Control - RSVD / Data - Data + CF-Ack + CF-Poll.
- WIFI11** Management(WIFI11) - Authentication / Control - RTS / Data - QoS Data + CF-Ack + CF-Poll.
- WIFI7** Management(WIFI7) - RSVD / Control - Control Wrapper / Data - CF-Ack + CF-Poll (no data)
- WIFI15** Management(WIFI15) - RSVD / Control - CF-End + CF-Ack / Data - QoS CF-Ack + CF-Poll (no data)

6.5.18.38 enum ProtocolPP::wifitype_t

Wifi type field

Parameters

WIFIMNG	- Wifi Management frame type
WIFIDAT	- Wifi Data frame type
WIFICTL	- Wifi Control frame type

Enumerator

WIFIMNG Management frame.

WIFIDAT Data frame.

WIFICTL Control frame.

6.5.18.39 enum ProtocolPP::wimaxmode_t

Wimax fields

Parameters

OFDM	- Orthogonal frequency division multiplexing
OFDMA	- Orthogonal frequency division multiple access
AGMH_OFDM	- Advanced GMH with OFDM
AGMH_OFDMA	- Advanced GMH with OFDMA

Enumerator

OFDM Orthogonal frequency division multiplexing.

OFDMA Orthogonal frequency division multiple access.

AGMH_OFDM Advanced GMH header with OFDM.

AGMH_OFDMA Advanced GMH header with OFDMA.

6.5.19 Function Documentation

6.5.19.1 ProtocolPP::Begin_Enum_String (field_t)

Direction of processing (ENCAP or DECAP)

Version field for the packet either IPV4 or IPV6 (ICMP or ICMPV6) (SSL30, TLS10, TLS11, TLS12, DTLS)

Diff services and ECN field

ID field for IPv4

Label field from IPv6
Fragmentation offset for IPv4
Next header (NH) field
Time-to-Live field
Checksum field in IPv4
Flags field in IPv4
Length field
Sequence number
Extended sequence number
IPv6 Extension header
Nodal JUMBOGRAM support
ICMP Type
ICMP SubType
Pointer to error in packet
Identifier from the invoking message
Gateway Internet Address
Maximum transmission unit of the next-hop link
Timestamp from the origination node
Receive timestamp
Transmit timestamp
Source Address
Destination Address
Message to send
Code to send
Data or Control bit
Control PDU type field
Polling bit
Extension bit
Reset sequence number (RSN)
Sequence number length in bits
Header extension type field
Length indicator field
Hyper frame number indicator (HFNI)
Super Field (SUFI)
PDCP SN of the first Missing PDCP SDU (FMS)
Length of BitMap
Bitmap for LTE
Five LSB(s) of PGK identity
PTK identity
PDCP SDU type (IP, [Enum_String\(ARP\)](#); [Enum_String\(PC5SIG\)](#); [Enum_String\(NONIP\)](#); RSVD)

Kd identity
Number of missing PDCP SDU(s) with associated COUNT value
PDCP SN of the PDCP SDU received on WLAN with highest associated PDCP COUNT value
Bearer value
Fresh value for F9 modes
Ethernet type
Short length field
TAG Control Information and Association Number
Packet number
Extended packet number
Secure Channel Identifier
Short Secure Channel Identifier
MacSEC security association key
Enable reception of packets
Enable transmission of packets
Protect MacSEC frames
MacSEC SA in use
Time of creation of this MacSEC SA
Time of last enable of RX or TX in this MacSEC SA
Time of last disable of RX or TX in this MacSEC SA
Padding field
CSRC count field
Marker field
Payload type field
Rollover Counter (ROC)
Timestamp field
Synchronization source identifier (SSRC)
Contributing source identifier (CSRC)
Acknowledge number
Data offset
Receive size window
Urgent pointer
TCP state of operation
OPTIONS
SEGLEN
PRECEDENCE
SNDUNA
SNDNXT
SNDWND
SNDUP

SNDW1
SNDW2
ISS
RCVNXT
RCVWND
RCVUP
IRS
TCPTIMEOUT
Epoch number for DTLS
Frame control sequence in entirety
Control Frame extension
Protocol version
SubType
To Data Stream
From Data Stream
More fragmentation
Retry
Power management
More data field
Wireless Encryption Protection (WEP)
Order
First address
Key derivation function (KDF) algorithm
Second address
Third address
Sequence control
Fourth address
Quality of Service field
HT control field
Extended IV flag
Key identification
Length of the header according to option bits
Header Type where zero for generic MAC header
Encryption Control 0=no encryption 1=encryption
Extended Subheader Field ESF=1 then ESF is present and follows MAC header
CRC Indicator 0=no [Enum_String\(CRC\)](#); 1=CRC present
Encryption Key Sequence. Index of TEK and IV used to encrypt the payload
Connection Identifier
Flow Identifier (AGHM header only)
Extended Header Group Indicator (AGHM header only)

Header Check Sequence 8-bit header checksum

IKE Conneciton Name

IKE Exchange Type

IKE Next Payload Value

IKE Major Revision Number

IKE Minor Revision Number

IKE Message Identifier when Initiator

IKE Message Identifier when Responder

IKE Initiator Security Parameter Index

IKE Responder Security Parameter Index

IKE Initiator Nonce

IKE Responder Nonce

IKE Child SA Key Generation Seed

IKE Initiator Encryption Key

IKE Responder Encryption Key

IKE Initiator Salt Value

IKE Responder Salt Value

IKE Integrity Algorithm

IKE Initiator Integrity Algorithm Key

IKE Responder Integrity Algorithm Key

IKE Psuedo Random Function Algorithm

IKE Psuedo Random Total Key Length

IKE Initiator Psuedo Random Key

IKE Responder Psuedo Random Key

IKE Diffie-Hellman Group ID

IKE Signature Algorithm

Bit size of the Private Key

Private Key

Public Key

Prime value for DSA (usually called "p")

Subprime value for DSA (usually called "q")

Generator value for DSA (usually called "g")

X-coordinate for the ECDSA curve

Y-coordinate for the ECDSA curve

Name of the current curve

6.5.19.2 ProtocolPP::Begin_Enum_String (endian_t)

6.5.19.3 ProtocolPP::Begin_Enum_String (platform_t)

6.5.19.4 ProtocolPP::Begin_Enum_String (pad_t)

6.5.19.5 ProtocolPP::Begin_Enum_String (protocol_t)

6.5.19.6 * param ETH_P PPP_MP Dummy type for PPP MP frames (0x0008) * @param ETH_P_LOCALTALK - Localtalk pseudo type (0x0009) * @param ETH_P_CAN - CAN ProtocolPP::Begin_Enum_String (ethid_t)

Ethernet ID Type for use with drivers

Parameters

<i>ETH_P_LOOP</i>	- Ethernet Loopback packet (0x0060)
<i>ETH_P_PUP</i>	- Xerox PUP packet (0x0200)
<i>ETH_P_PUPAT</i>	- Xerox PUP Addr Trans packet (0x0201)
<i>ETH_P_IP</i>	- Internet Protocol packet (0x0800)
<i>ETH_P_X25</i>	- CCITT X.25 (0x0805)
<i>ETH_P_ARP</i>	- Address Resolution packet (0x0806)
<i>ETH_P_BPQ</i>	- G8BPQ AX.25 Ethernet Packet [NOT AN OFFICIALLY REGISTERED ID] (0x08FF)
<i>ETH_P_IEEEP-UP</i>	- Xerox IEEE802.3 PUP packet (0xa00)
<i>ETH_P_IEEEP-UPAT</i>	- Xerox IEEE802.3 PUP Addr Trans packet (0xa01)
<i>ETH_P_BATMAN-N</i>	- B.A.T.M.A.N.-Advanced packet [NOT AN OFFICIALLY REGISTERED ID] (0x4305)
<i>ETH_P_DEC</i>	- DEC Assigned proto (0x6000)
<i>ETH_P_DNA_DL</i>	- DEC DNA Dump/Load (0x6001)
<i>ETH_P_DNA_RC</i>	- DEC DNA Remote Console (0x6002)
<i>ETH_P_DNA_RT</i>	- DEC DNA Routing (0x6003)
<i>ETH_P_LAT</i>	- DEC LAT (0x6004)
<i>ETH_P_DIAG</i>	- DEC Diagnostics (0x6005)
<i>ETH_P_CUST</i>	- DEC Customer use (0x6006)
<i>ETH_P_SCA</i>	- DEC Systems Comms Arch (0x6007)
<i>ETH_P_TEB</i>	- Trans Ether Bridging (0x6558)
<i>ETH_P_RARP</i>	- Reverse Addr Res packet (0x8035)
<i>ETH_P_ATALK</i>	- Appletalk DDP (0x809B)
<i>ETH_P_AARP</i>	- Appletalk AARP (0x80F3)
<i>ETH_P_8021Q</i>	- 802.1Q VLAN Extended Header (0x8100)
<i>ETH_P_IPX</i>	- IPX over DIX (0x8137)
<i>ETH_P_IPV6</i>	- IPv6 over bluebook (0x86DD)
<i>ETH_P_PAUSE</i>	- IEEE Pause frames. See 802.3 31B (0x8808)
<i>ETH_P_SLOW</i>	- Slow Protocol. See 802.3ad 43B (0x8809)
<i>ETH_P_WCCP</i>	- Web-cache coordination protocol defined in draft-wilson-wrec-wccp-v2-00.txt (0x883E)
<i>ETH_P_PPP_DISC</i>	- PPPoE discovery messages (0x8863)
<i>ETH_P_PPP_SESSION</i>	- PPPoE session messages (0x8864)
<i>ETH_P_MPLS_UC</i>	- MPLS Unicast traffic (0x8847)

<i>ETH_P_MPLS_MC</i>	- MPLS Multicast traffic (0x8848)
<i>ETH_P_ATMM_POA</i>	- MultiProtocol Over ATM (0x884c)
<i>ETH_P_LINK_CTL</i>	- HPNA, wlan link local tunnel (0x886c)
<i>ETH_P_ATMFA_TE</i>	- Frame-based ATM Transport over Ethernet (0x8884)
<i>ETH_P_PAE</i>	- Port Access Entity (IEEE 802.1X) (0x888E)
<i>ETH_P_AOE</i>	- ATA over Ethernet (0x88A2)
<i>ETH_P_8021AD</i>	- 802.1ad Service VLAN (0x88A8)
<i>ETH_P_802_E_X1</i>	- 802.1 Local Experimental 1. (0x88B5)
<i>ETH_P_TIPC</i>	- TIPC (0x88CA)
<i>ETH_P_MACSEC</i>	- 802.1ae MACsec (0x88E5)
<i>ETH_P_8021AH</i>	- 802.1ah Backbone Service Tag (0x88E7)
<i>ETH_P_MVRP</i>	- 802.1Q MVRP (0x88F5)
<i>ETH_P_1588</i>	- IEEE 1588 Timesync (0x88F7)
<i>ETH_P_FCOE</i>	- Fibre Channel over Ethernet (0x8906)
<i>ETH_P_IBOE</i>	- Infiniband over Ethernet (0x8915)
<i>ETH_P_TDLS</i>	- TDLS (0x890D)
<i>ETH_P_FIP</i>	- FCoE Initialization Protocol (0x8914)
<i>ETH_P_80221</i>	- IEEE 802.21 Media Independent Handover Protocol (0x8917)
<i>ETH_P_NSH</i>	- Network Service Header (0x894F)
<i>ETH_P_LOOPBACK_ACK</i>	- Ethernet loopback packet, per IEEE 802.3 (0x9000)
<i>ETH_P_QINQ1</i>	- deprecated QinQ VLAN [NOT AN OFFICIALLY REGISTERED ID] (0x9100)
<i>ETH_P_QINQ2</i>	- deprecated QinQ VLAN [NOT AN OFFICIALLY REGISTERED ID] (0x9200)
<i>ETH_P_QINQ3</i>	- deprecated QinQ VLAN [NOT AN OFFICIALLY REGISTERED ID] (0x9300)
<i>ETH_P_EDSA</i>	- Ethertype DSA [NOT AN OFFICIALLY REGISTERED ID] (0xDADA)
<i>ETH_P_IFE</i>	- ForCES inter-FE LFB type (0xED3E)
<i>ETH_P_AF_IU_CV</i>	- IBM af_iucv [NOT AN OFFICIALLY REGISTERED ID] (0xFBFB)
<i>ETH_P_802_3_MINUS_MIN</i>	- If the value in the ethernet type is less than this value then the frame is Ethernet II. Else it is 802.3 (0x0600)
<i>ETH_P_802_3</i>	- Dummy type for 802.3 frames (0x0001)
<i>ETH_P_AX25</i>	- Dummy protocol id for AX.25 (0x0002)
<i>ETH_P_ALL</i>	- Every packet (be careful!!!!) (0x0003)
<i>ETH_P_802_2</i>	- 802.2 frames (0x0004)
<i>ETH_P_SNAP</i>	- Internal only (0x0005)
<i>ETH_P_DDCMP</i>	- DEC DDCMP: Internal only (0x0006)
<i>ETH_P_WAN_PPP</i>	- Dummy type for WAN PPP frames

6.5.19.7 `ProtocolPP::Begin_Enum_String(direction_t)`

6.5.19.8 `ProtocolPP::Begin_Enum_String(cipher_t)`

6.5.19.9 `ProtocolPP::Begin_Enum_String(auth_t)`

6.5.19.10 `ProtocolPP::Begin_Enum_String(iana_t)`

6.5.19.11 `ProtocolPP::Begin_Enum_String(err_t)`

- 6.5.19.12 ProtocolPP::Begin_Enum_String(`replay_t`)
- 6.5.19.13 ProtocolPP::Begin_Enum_String(`ipmode_t`)
- 6.5.19.14 ProtocolPP::Begin_Enum_String(`icmpmsg_t`)
- 6.5.19.15 ProtocolPP::Begin_Enum_String(`icmpcode_t`)
- 6.5.19.16 ProtocolPP::Begin_Enum_String(`tlsver_t`)
- 6.5.19.17 ProtocolPP::Begin_Enum_String(`tlstype_t`)
- 6.5.19.18 ProtocolPP::Begin_Enum_String(`tlslvl_t`)
- 6.5.19.19 ProtocolPP::Begin_Enum_String(`tls_handshake_t`)
- 6.5.19.20 ProtocolPP::Begin_Enum_String(`tls_error_t`)
- 6.5.19.21 ProtocolPP::Begin_Enum_String(`srpcipher_t`)
- 6.5.19.22 ProtocolPP::Begin_Enum_String(`tls_ciphersuite_t`)
- 6.5.19.23 ProtocolPP::Begin_Enum_String(`wifitype_t`)
- 6.5.19.24 ProtocolPP::Begin_Enum_String(`wifisub_t`)
- 6.5.19.25 ProtocolPP::Begin_Enum_String(`wifictext_t`)
- 6.5.19.26 ProtocolPP::Begin_Enum_String(`wimaxmode_t`)
- 6.5.19.27 ProtocolPP::Begin_Enum_String(`macsecmode_t`)
- 6.5.19.28 ProtocolPP::Begin_Enum_String(`dir_id_t`)
- 6.5.19.29 ProtocolPP::Begin_Enum_String(`role_id_t`)
- 6.5.19.30 ProtocolPP::Begin_Enum_String(`ipsec_mode_t`)
- 6.5.19.31 ProtocolPP::Begin_Enum_String(`exchg_t`)
- 6.5.19.32 ProtocolPP::Begin_Enum_String(`ike_pyld_t`)
- 6.5.19.33 ProtocolPP::Begin_Enum_String(`transform_type_t`)
- 6.5.19.34 ProtocolPP::Begin_Enum_String(`transform_attr_t`)
- 6.5.19.35 ProtocolPP::Begin_Enum_String(`encr_id_t`)
- 6.5.19.36 ProtocolPP::Begin_Enum_String(`prf_id_t`)
- 6.5.19.37 ProtocolPP::Begin_Enum_String(`integ_id_t`)
- 6.5.19.38 ProtocolPP::Begin_Enum_String(`dh_id_t`)
- 6.5.19.39 ProtocolPP::Begin_Enum_String(`esn_id_t`)

- 6.5.19.40 `ProtocolPP::Begin_Enum_String(id_type_t)`
- 6.5.19.41 `ProtocolPP::Begin_Enum_String(encoding_t)`
- 6.5.19.42 `ProtocolPP::Begin_Enum_String(auth_method_t)`
- 6.5.19.43 `ProtocolPP::Begin_Enum_String(notify_err_t)`
- 6.5.19.44 `ProtocolPP::Begin_Enum_String(notify_status_t)`
- 6.5.19.45 `ProtocolPP::Begin_Enum_String(ike_ipcomp_t)`
- 6.5.19.46 `ProtocolPP::Begin_Enum_String(ip_proto_t)`
- 6.5.19.47 `ProtocolPP::Begin_Enum_String(protocol_id_t)`
- 6.5.19.48 `ProtocolPP::Begin_Enum_String(ike_ts_t)`
- 6.5.19.49 `ProtocolPP::Begin_Enum_String(cfg_resp_t)`
- 6.5.19.50 `ProtocolPP::Begin_Enum_String(cfg_attr_t)`
- 6.5.19.51 `ProtocolPP::Begin_Enum_String(ike_gateway_t)`
- 6.5.19.52 `ProtocolPP::Begin_Enum_String(rohc_attr_t)`
- 6.5.19.53 `ProtocolPP::Begin_Enum_String(ike_secpass_t)`
- 6.5.19.54 `ProtocolPP::Begin_Enum_String(ike_hash_t)`
- 6.5.19.55 `ProtocolPP::Begin_Enum_String(rsapadtype_t)`
- 6.5.19.56 `static void ProtocolPP::do_recursion(w128_t *r, w128_t *a, w128_t *b, w128_t *c, w128_t *d) [inline], [static]`

This function represents the recursion formula.

Parameters

<i>r</i>	output
<i>a</i>	a 128-bit part of the internal state array
<i>b</i>	a 128-bit part of the internal state array
<i>c</i>	a 128-bit part of the internal state array
<i>d</i>	a 128-bit part of the internal state array

- 6.5.19.57 `static void ProtocolPP::gen_rand_array(w128_t *array, int size) [static]`

This function fills the user-specified array with pseudorandom integers.

Parameters

<i>array</i>	an 128-bit array to be filled by pseudorandom numbers.
<i>size</i>	number of 128-bit pseudorandom numbers to be generated.

6.5.19.58 static void ProtocolPP::gen_rand_array (w128_t * array, int size) [static]

This function fills the user-specified array with pseudorandom integers.

Parameters

<i>array</i>	an 128-bit array to be filled by pseudorandom numbers.
<i>size</i>	number of 128-bit pseudorandom numbers to be generated.

6.5.19.59 static void ProtocolPP::gen_rand_array (w128_t * *array*, int *size*) [inline], [static]

This function fills the user-specified array with pseudorandom integers.

Parameters

<i>array</i>	an 128-bit array to be filled by pseudorandom numbers.
<i>size</i>	number of 128-bit pesudorandom numbers to be generated.

6.5.19.60 static void ProtocolPP::gen_rand_array (w128_t * *array*, int *size*) [static]

This function fills the user-specified array with pseudorandom integers.

Parameters

<i>array</i>	an 128-bit array to be filled by pseudorandom numbers.
<i>size</i>	number of 128-bit pseudorandom numbers to be generated.

6.5.19.61 static void ProtocolPP::jpoly1305_blocks (jpoly1305_state_internal_t * *st*, const unsigned char * *m*, size_t *bytes*) [static]

6.5.19.62 static void ProtocolPP::jpoly1305_blocks (jpoly1305_state_internal_t * *st*, const unsigned char * *m*, size_t *bytes*) [static]

6.5.19.63 static void ProtocolPP::lshift128 (w128_t * *out*, w128_t const * *in*, int *shift*) [inline], [static]

This function simulates SIMD 128-bit left shift by the standard C. The 128-bit integer given in *in* is shifted by (*shift* * 8) bits. This function simulates the LITTLE ENDIAN SIMD.

Parameters

<i>out</i>	the output of this function
<i>in</i>	the 128-bit data to be shifted
<i>shift</i>	the shift value

6.5.19.64 static void ProtocolPP::mm_recursion (__m128i * *r*, __m128i *a*, __m128i *b*, __m128i *c*, __m128i *d*) [inline], [static]

This function represents the recursion formula.

Parameters

<i>r</i>	an output
<i>a</i>	a 128-bit part of the interal state array
<i>b</i>	a 128-bit part of the interal state array
<i>c</i>	a 128-bit part of the interal state array
<i>d</i>	a 128-bit part of the interal state array

```
6.5.19.65 static void ProtocolPP::mm_recursion ( __m128i * r, __m128i a, __m128i b, __m128i c, __m128i * d )  
[inline], [static]
```

This function represents the recursion formula.

Parameters

<i>r</i>	an output
<i>a</i>	a 128-bit part of the interal state array
<i>b</i>	a 128-bit part of the interal state array
<i>c</i>	a 128-bit part of the interal state array
<i>d</i>	a 128-bit part of the interal state array

6.5.19.66 static void ProtocolPP::neon_recursion (`uint32x4_t * r, uint32x4_t a, uint32x4_t b, uint32x4_t c, uint32x4_t d`) [inline], [static]

This function represents the recursion formula.

Parameters

<i>r</i>	an output
<i>a</i>	a 128-bit part of the interal state array
<i>b</i>	a 128-bit part of the interal state array
<i>c</i>	a 128-bit part of the interal state array
<i>d</i>	a 128-bit part of the interal state array

6.5.19.67 static void ProtocolPP::rshift128 (`w128_t * out, w128_t const * in, int shift`) [inline], [static]

This function simulates SIMD 128-bit right shift by the standard C. The 128-bit integer given in *in* is shifted by (*shift* * 8) bits. This function simulates the LITTLE ENDIAN SIMD.

Parameters

<i>out</i>	the output of this function
<i>in</i>	the 128-bit data to be shifted
<i>shift</i>	the shift value

6.5.19.68 void ProtocolPP::sfmt_gen_rand_all ()

This function fills the internal state array with pseudorandom integers.

6.5.19.69 static void ProtocolPP::swap (`w128_t * array, int size`) [inline], [static]

This function swaps high and low 32-bit of 64-bit integers in user specified array.

Parameters

<i>array</i>	an 128-bit array to be swaped.
<i>size</i>	size of 128-bit array.

6.5.19.70 static void ProtocolPP::U16TO8 (`unsigned char * p, unsigned short v`) [static]

6.5.19.71 static void ProtocolPP::U32TO8 (`unsigned char * p, unsigned long v`) [static]

6.5.19.72 static void ProtocolPP::U64TO8 (`unsigned char * p, unsigned long long v`) [static]

6.5.19.73 static unsigned short ProtocolPP::U8TO16 (`const unsigned char * p`) [static]

6.5.19.74 static unsigned long ProtocolPP::U8TO32 (`const unsigned char * p`) [static]

6.5.19.75 static unsigned long long ProtocolPP::U8TO64 (const unsigned char * *p*) [static]

6.5.19.76 static vector<unsigned int> ProtocolPP::vec_recursion (vector<unsigned int> *a*, vector<unsigned int> *b*, vector<unsigned int> *c*, vector<unsigned int> *d*) [inline], [static]

This function represents the recursion formula in AltiVec and BIG ENDIAN.

Parameters

<i>a</i>	a 128-bit part of the interal state array
<i>b</i>	a 128-bit part of the interal state array
<i>c</i>	a 128-bit part of the interal state array
<i>d</i>	a 128-bit part of the interal state array

Returns

output

6.5.20 Variable Documentation

6.5.20.1 ProtocolPP::End_Enum_String

6.6 tinyxml2 Namespace Reference

Classes

- class [StrPair](#)
- class [DynArray](#)
- class [MemPool](#)
- class [MemPoolIT](#)
- class [XMLVisitor](#)
- class [XMLUtil](#)
- class [XMLNode](#)
- class [XMLText](#)
- class [XMLComment](#)
- class [XMLDeclaration](#)
- class [XMLUnknown](#)
- class [XmlAttribute](#)
- class [XMLElement](#)
- class [XMLDocument](#)
- class [XMLHandle](#)
- class [XMLConstHandle](#)
- class [XMLPrinter](#)

Enumerations

- enum [XMLError](#) {

 XML_SUCCESS = 0, [XML_NO_ATTRIBUTE](#), [XML_WRONG_ATTRIBUTE_TYPE](#), [XML_ERROR_FILE_NOT_FOUND](#),

 [XML_ERROR_FILE_COULD_NOT_BE_OPENED](#), [XML_ERROR_FILE_READ_ERROR](#), [XML_ERROR_PARSING_ELEMENT](#), [XML_ERROR_PARSING_ATTRIBUTE](#),

 [XML_ERROR_PARSING_TEXT](#), [XML_ERROR_PARSING_CDATA](#), [XML_ERROR_PARSING_COMMENT](#), [XML_ERROR_PARSING_DECLARATION](#),

 [XML_ERROR_PARSING_UNKNOWN](#), [XML_ERROR_EMPTY_DOCUMENT](#), [XML_ERROR_MISMATCHED_ELEMENT](#), [XML_ERROR_PARSING](#),

 [XML_CAN_NOT_CONVERT_TEXT](#), [XML_NO_TEXT_NODE](#), [XML_ELEMENT_DEPTH_EXCEEDED](#), [XML_ERROR_COUNT](#) }

- enum `Whitespace` { `PRESERVE_WHITESPACE`, `COLLAPSE_WHITESPACE` }

6.6.1 Enumeration Type Documentation

6.6.1.1 enum tinyxml2::Whitespace

Enumerator

`PRESERVE_WHITESPACE`
`COLLAPSE_WHITESPACE`

6.6.1.2 enum tinyxml2::XMLError

Enumerator

`XML_SUCCESS`
`XML_NO_ATTRIBUTE`
`XML_WRONG_ATTRIBUTE_TYPE`
`XML_ERROR_FILE_NOT_FOUND`
`XML_ERROR_FILE_COULD_NOT_BE_OPENED`
`XML_ERROR_FILE_READ_ERROR`
`XML_ERROR_PARSING_ELEMENT`
`XML_ERROR_PARSING_ATTRIBUTE`
`XML_ERROR_PARSING_TEXT`
`XML_ERROR_PARSING_CDATA`
`XML_ERROR_PARSING_COMMENT`
`XML_ERROR_PARSING_DECLARATION`
`XML_ERROR_PARSING_UNKNOWN`
`XML_ERROR_EMPTY_DOCUMENT`
`XML_ERROR_MISMATCHED_ELEMENT`
`XML_ERROR_PARSING`
`XML_CAN_NOT_CONVERT_TEXT`
`XML_NO_TEXT_NODE`
`XML_ELEMENT_DEPTH_EXCEEDED`
`XML_ERROR_COUNT`

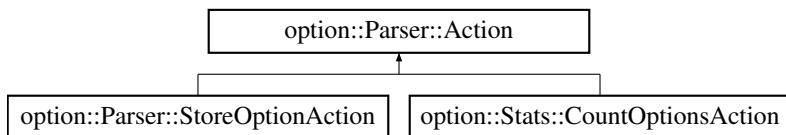
Chapter 7

Class Documentation

7.1 option::Parser::Action Struct Reference

```
#include <optionparser.h>
```

Inheritance diagram for option::Parser::Action:



Public Member Functions

- virtual bool [perform \(Option &\)](#)
Called by Parser::workhorse() for each Option that has been successfully parsed (including unknown options if they have a Descriptor whose Descriptor::check_arg does not return ARG_ILLEGAL).
- virtual bool [finished \(int numargs, const char **args\)](#)
Called by Parser::workhorse() after finishing the parse.

7.1.1 Member Function Documentation

7.1.1.1 virtual bool option::Parser::Action::finished (int numargs, const char ** args) [inline], [virtual]

Called by Parser::workhorse() after finishing the parse.

Parameters

<code>numargs</code>	the number of non-option arguments remaining
<code>args</code>	pointer to the first remaining non-option argument (if numargs > 0).

Returns

`false` iff a fatal error has occurred.

Reimplemented in [option::Parser::StoreOptionAction](#).

7.1.1.2 virtual bool option::Parser::Action::perform(Option &) [inline], [virtual]

Called by Parser::workhorse() for each [Option](#) that has been successfully parsed (including unknown options if they have a [Descriptor](#) whose [Descriptor::check_arg](#) does not return [ARG_ILLEGAL](#).

Returns `false` iff a fatal error has occurred and the parse should be aborted.

Reimplemented in [option::Parser::StoreOptionAction](#), and [option::Stats::CountOptionsAction](#).

The documentation for this struct was generated from the following file:

- [jtestbench/include/optionparser.h](#)

7.2 aead_chacha_poly1305 Class Reference

```
#include "include/aead_chacha_poly1305.h"
```

7.2.1 Detailed Description

See IETF <https://tools.ietf.org/pdf/draft-nir-cfrg-chacha20-poly1305-01.pdf>

7.2.2 ChaCha20 and Poly1305 for IETF protocols

This document defines the ChaCha20 stream cipher, as well as the use of the Poly1305 authenticator, both as stand-alone algorithms, and as a "combined mode", or Authenticated Encryption with Additional Data (AEAD) algorithm.

AEAD Construction

Note: Much of the content of this document, including this AEAD construction is taken from Adam Langley's draft ([[agl-draft](#)]) for the use of these algorithms in TLS. The AEAD construction described here is called AEAD_CHACHA20-POLY1305

AEAD_CHACHA20-POLY1305 is an authenticated encryption with additional data algorithm. The inputs to AEAD_CHACHA20-POLY1305 are:

- A 256-bit key
- A 96-bit nonce - different for each invocation with the same key.
- An arbitrary length plaintext
- Arbitrary length additional data

The ChaCha20 and Poly1305 primitives are combined into an AEAD that takes a 256-bit key and 64-bit IV as follows:

- First the 96-bit nonce is constructed by prepending a 32-bit constant value to the IV. This could be set to zero, or could be derived from keying material, or could be assigned to a sender. It is up to the specific protocol to define the source for that 32-bit value
- Next, a Poly1305 one-time key is generated from the 256-bit key and nonce using the procedure described in Section 2.6
- The ChaCha20 encryption function is called to encrypt the plaintext, using the same key and nonce, and with the initial counter set to 1
- The Poly1305 function is called with the Poly1305 key calculated above, and a message constructed as a concatenation of the following:
- The additional data

- The length of the additional data in octets (as a 64-bit little-endian integer). TBD: bit count rather than octets? network order?
- The ciphertext
- The length of the ciphertext in octets (as a 64-bit little-endian integer). TBD: bit count rather than octets? network order?

Decryption is pretty much the same thing

The output from the AEAD is twofold:

- A ciphertext of the same length as the plaintext
- A 128-bit tag, which is the output of the Poly1305 function

A few notes about this design:

1. The amount of encrypted data possible in a single invocation is $2^{32} - 1$ blocks of 64 bytes each, for a total of 247,877,906,880 bytes, or nearly 256 GB. This should be enough for traffic protocols such as IPsec and TLS, but may be too small for file and/or disk encryption. For such uses, we can return to the original design, reduce the nonce to 64 bits, and use the integer at position 13 as the top 32 bits of a 64-bit block counter, increasing the total message size to over a million petabytes (1,180,591,620,717,411,303,360 bytes to be exact)
2. Despite the previous item, the ciphertext length field in the construction of the buffer on which Poly1305 runs limits the ciphertext (and hence, the plaintext) size to 2^{64} bytes, or sixteen thousand petabytes (18,446,744,073,709,551,616 bytes to be exact)

Implementation Advice

Each block of ChaCha20 involves 16 move operations and one increment operation for loading the state, 80 each of XOR, addition and Roll operations for the rounds, 16 more add operations and 16 XOR operations for protecting the plaintext

Section 2.3 describes the ChaCha block function as "adding the original input words". This implies that before starting the rounds on the ChaCha state, it is copied aside only to be added in later. This would be correct, but it saves a few operations to instead copy the state and do the work on the copy. This way, for the next block you don't need to recreate the state, but only to increment the block counter. This saves approximately 5.5% of the cycles

It is NOT RECOMMENDED to use a generic big number library such as the one in OpenSSL for the arithmetic operations in Poly1305. Such libraries use dynamic allocation to be able to handle any-sized integer, but that flexibility comes at the expense of performance as well as side-channel security. More efficient implementations that run in constant time are available, one of them in DJB's own library, NaCl ([NaCl]).

Security Considerations

The ChaCha20 cipher is designed to provide 256-bit security. The Poly1305 authenticator is designed to ensure that forged messages are rejected with a probability of $1 - (\frac{n}{2^{102}})$ for a $16n$ -byte message, even after sending 2^{64} legitimate messages, so it is SUF-CMA in the terminology of [AE].

Proving the security of either of these is beyond the scope of this document. Such proofs are available in the referenced academic papers. The most important security consideration in implementing this draft is the uniqueness of the nonce used in ChaCha20. Counters and LFSRs are both acceptable ways of generating unique nonces, as is encrypting a counter using a 64-bit cipher such as DES. Note that it is not acceptable to use a truncation of a counter encrypted with a 128-bit or 256-bit cipher, because such a truncation may repeat after a short time

The Poly1305 key MUST be unpredictable to an attacker. Randomly generating the key would fulfill this requirement, except that Poly1305 is often used in communications protocols, so the receiver should know the key. Pseudo-random number generation such as by encrypting a counter is acceptable. Using ChaCha with a secret key and a nonce is also acceptable

The algorithms presented here were designed to be easy to implement in constant time to avoid side-channel vulnerabilities. The operations used in ChaCha20 are all additions, XORs, and fixed rotations. All of these can and

should be implemented in constant time. Access to offsets into the ChaCha state and the number of operations do not depend on any property of the key, eliminating the chance of information about the key leaking through the timing of cache misses

For Poly1305, the operations are addition, multiplication and modulus, all on >128 -bit numbers. This can be done in constant time, but a naive implementation (such as using some generic big number library) will not be constant time. For example, if the multiplication is performed as a separate operation from the modulus, the result will sometimes be under 2^{256} and some times be above 2^{256} . Implementers should be careful about timing side-channels for Poly1305 by using the appropriate implementation of these operations

For API Documentation:

See Also

[ProtocolPP::chacha20](#)
[ProtocolPP::poly1305](#)
[ProtocolPP::jmodes](#)
[ProtocolPP::ciphers](#)

For Additional Documentation:

See Also

[chacha20](#)
[poly1305](#)
[jmodes](#)
[ciphers](#)

See the license for POLY1305 and CHACHA20 at:

- [MIT](#) or PUBLIC DOMAIN

Protocol++ modified by : John Peter Greninger © John Peter Greninger 2015-2019 All Rights Reserved

All copyrights and trademarks are the property of their respective owners

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer
- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution
- Any and all modifications must be returned to John Peter Greninger at GitHub.com <https://github.com/jpgreninger/protocolpp> for evaluation. Inclusion of modifications in the source code shall be determined solely by John Peter Greninger. Failure to provide modifications shall render this license NULL and VOID and revoke any rights to use of Protocol++
- Commercial use requires a fee-based license obtainable at www.protocolpp.com
- Academic use requires written and notarized permission from John Peter and Sheila Rocha Greninger
- US Copyrights at [https://www.copyright.gov/](http://www.copyright.gov/)
 - TXu002059872 (Version 1.0.0)
 - TXu002066632 (Version 1.2.7)
 - TXu002082674 (Version 1.4.0)
 - TXu002097880 (Version 2.0.0)

The names of its contributors may not be used to endorse or promote products derived from this software without specific prior written permission

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE

The documentation for this class was generated from the following file:

- [include/aead_chacha_poly1305.h](#)

7.3 ProtocolPP::aead_chacha_poly1305 Class Reference

```
#include <aead_chacha_poly1305.h>
```

Public Types

- enum [dir_t](#) { [ENC](#), [DEC](#) }
- Direction of processing for dual mode.*

Public Member Functions

- [aead_chacha_poly1305](#) ([dir_t](#) direction, const unsigned char *key, unsigned int keylen, const unsigned char *nonce, unsigned int noncelen, unsigned int icvlen=0)
- virtual [~aead_chacha_poly1305](#) ()
Standard Deconstructor for AEAD CHACHA20-POLY1305 algorithm.
- void [ProcessData](#) (const unsigned char *input, unsigned char *output, unsigned int [length](#), const unsigned char *aad, unsigned int aadlen)
- void [result](#) (unsigned char *icv, unsigned int [length](#)=16)

7.3.1 Member Enumeration Documentation

7.3.1.1 enum ProtocolPP::aead_chacha_poly1305::dir_t

Direction of processing for dual mode.

Enumerator

- ENC** Encryption.
DEC Decryption.

7.3.2 Constructor & Destructor Documentation

7.3.2.1 ProtocolPP::aead_chacha_poly1305::aead_chacha_poly1305 ([dir_t](#) [direction](#), const unsigned char * [key](#), unsigned int [keylen](#), const unsigned char * [nonce](#), unsigned int [noncelen](#), unsigned int [icvlen](#) = 0)

Constructor for AEAD CHACHA20-POLY1305 algorithm

Parameters

<i>direction</i>	- direction of processing
<i>key</i>	- initialization key
<i>keylen</i>	- length of the key in bytes
<i>nonce</i>	- nonce for computation
<i>noncelen</i>	- length of the nonce in bytes
<i>icvlen</i>	- for decryption, length of the ICV tag

7.3.2.2 virtual ProtocolPP::aead_chacha_poly1305::~aead_chacha_poly1305() [inline], [virtual]

Standard Deconstructor for AEAD CHACHA20-POLY1305 algorithm.

7.3.3 Member Function Documentation

7.3.3.1 void ProtocolPP::aead_chacha_poly1305::ProcessData (const unsigned char * *input*, unsigned char * *output*, unsigned int *length*, const unsigned char * *aad*, unsigned int *aadlen*)

Calculates the ciphertext and MAC for AEAD CHACHA20-POLY1305 algorithm

Parameters

<i>input</i>	- data to process
<i>output</i>	- result of the computation
<i>length</i>	- length of the data
<i>aad</i>	- authentication only data
<i>aadlen</i>	- length of the AAD data

7.3.3.2 void ProtocolPP::aead_chacha_poly1305::result (unsigned char * *icv*, unsigned int *length* = 16)

Returns the MAC using for AEAD CHACHA20-POLY1305 algorithm

Parameters

<i>icv</i>	- ICV tag to retrieve
<i>length</i>	- length of ICV to retrieve

The documentation for this class was generated from the following file:

- [include/aead_chacha_poly1305.h](#)

7.4 option::Arg Struct Reference

Functions for checking the validity of option arguments.

```
#include <optionparser.h>
```

Static Public Member Functions

- static [ArgStatus None](#) (const [Option](#) &, bool)

For options that don't take an argument: Returns ARG_NONE.
- static [ArgStatus Required](#) (const [Option](#) &option, bool msg)

For options that require an argument: Returns ARG_OK.
- static [ArgStatus Optional](#) (const [Option](#) &option, bool)

Returns ARG_OK if the argument is attached and ARG_IGNORE otherwise.

7.4.1 Detailed Description

Functions for checking the validity of option arguments.

Every [Option](#) has such a function assigned in its [Descriptor](#).

```
* Descriptor usage[] = { {UNKNOWN, 0, "", "", Arg::None, ""}, ... };
```

A CheckArg function has the following signature:

```
ArgStatus CheckArg(const Option& option, bool msg);
```

It is used to check if a potential argument would be acceptable for the option. It will even be called if there is no argument. In that case `option.arg` will be NULL.

If `msg` is true and the function determines that an argument is not acceptable and that this is a fatal error, it should output a message to the user before returning [ARG_ILLEGAL](#). If `msg` is false the function should remain silent (or you will get duplicate messages).

See [ArgStatus](#) for the meaning of the return values.

While you can provide your own functions, often the following pre-defined checks (which never return [ARG_ILLEGAL](#)) will suffice:

- [Arg::None](#) For options that don't take an argument: Returns ARG_NONE.
- [Arg::Optional](#) Returns ARG_OK if the argument is attached and ARG_IGNORE otherwise.

The following example code can serve as starting place for writing your own more complex CheckArg functions:

```
* struct Arg: public option::Arg
* {
*     static void printError(const char* msg1, const option::Option& opt, const char* msg2)
*     {
*         fprintf(stderr, "ERROR: %s", msg1);
*         fwrite(opt.name, opt.namelen, 1, stderr);
*         fprintf(stderr, "%s", msg2);
*     }
*
*     static option::ArgStatus Unknown(const option::Option& option, bool msg)
*     {
*         if (msg) printError("Unknown option '", option, "'\n");
*         return option::ARG_ILLEGAL;
*     }
*
*     static option::ArgStatus Required(const
*         option::Option& option, bool msg)
*     {
*         if (option.arg != 0)
*             return option::ARG_OK;
*
*         if (msg) printError("Option '", option, "' requires an argument\n");
*         return option::ARG_ILLEGAL;
*     }
*
*     static option::ArgStatus NonEmpty(const option::Option& option, bool msg
*     )
*     {
*         if (option.arg != 0 && option.arg[0] != 0)
*             return option::ARG_OK;
*
*         if (msg) printError("Option '", option, "' requires a non-empty argument\n");
*         return option::ARG_ILLEGAL;
*     }
*
*     static option::ArgStatus Numeric(const option::Option& option, bool msg)
*     {
*         char* endptr = 0;
*         if (option.arg != 0 && strtol(option.arg, &endptr, 10)) {};
*     }
}
```

```

*     if (endptr != option.arg && *endptr == 0)
*         return option::ARG_OK;
*
*     if (msg) printError("Option '", option, "' requires a numeric argument\n");
*     return option::ARG_ILLEGAL;
* }
* };
*

```

7.4.2 Member Function Documentation

7.4.2.1 static ArgStatus option::Arg::None (const Option &, bool) [inline], [static]

For options that don't take an argument: Returns ARG_NONE.

7.4.2.2 static ArgStatus option::Arg::Optional (const Option & option, bool) [inline], [static]

Returns ARG_OK if the argument is attached and ARG_IGNORE otherwise.

7.4.2.3 static ArgStatus option::Arg::Required (const Option & option, bool msg) [inline], [static]

For options that require an argument: Returns ARG_OK.

The documentation for this struct was generated from the following file:

- jtestbench/include/optionparser.h

7.5 chacha20 Class Reference

```
#include "include/chacha20.h"
```

7.5.1 Detailed Description

LICENSE:

This is free and unencumbered software released into the public domain. Anyone is free to copy, modify, publish, use, compile, sell, or distribute this software, either in source code form or as a compiled binary, for any purpose, commercial or non-commercial, and by any means.

In jurisdictions that recognize copyright laws, the author or authors of this software dedicate any and all copyright interest in the software to the public domain. We make this dedication for the benefit of the public at large and to the detriment of our heirs and successors. We intend this dedication to be an overt act of relinquishment in perpetuity of all present and future rights to this software under copyright law.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

For more information, please refer to <http://unlicense.org>

Protocol++ modified by : John Peter Greninger © John Peter Greninger 2015-2019 All Rights Reserved

All copyrights and trademarks are the property of their respective owners

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer
- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution
- Any and all modifications must be returned to John Peter Greninger at GitHub.com <https://github.com/jpgreninger/protocolpp> for evaluation. Inclusion of modifications in the source code shall be determined solely by John Peter Greninger. Failure to provide modifications shall render this license NULL and VOID and revoke any rights to use of Protocol++
- Commercial use requires a fee-based license obtainable at www.protocolpp.com
- Academic use requires written and notarized permission from John Peter and Sheila Rocha Greninger
- US Copyrights at <https://www.copyright.gov/>
 - TXu002059872 (Version 1.0.0)
 - TXu002066632 (Version 1.2.7)
 - TXu002082674 (Version 1.4.0)
 - TXu002097880 (Version 2.0.0)

The names of its contributors may not be used to endorse or promote products derived from this software without specific prior written permission

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE

The documentation for this class was generated from the following file:

- [include/chacha20.h](#)

7.6 ProtocolPP::chacha20 Class Reference

```
#include <chacha20.h>
```

Public Member Functions

- [chacha20](#) (const uint8_t *key, const uint8_t *nonce, uint64_t counter)
 - [chacha20](#) (const uint8_t *ctx, unsigned int [length](#))
 - virtual [~chacha20](#) ()
- Standard Deconstructor.*
- void [ProcessData](#) (const uint8_t *input, uint8_t *output, unsigned int [length](#))
 - void [context](#) (uint8_t *context, unsigned int [length](#)=32)

7.6.1 Constructor & Destructor Documentation

7.6.1.1 `ProtocolPP::chacha20::chacha20 (const uint8_t * key, const uint8_t * nonce, uint64_t counter)`

Constructor for CHACHA20 algorithm

Parameters

<i>key</i>	- key for encryption
<i>nonce</i>	- initial value for the NONCE
<i>counter</i>	- initial counter value

7.6.1.2 ProtocolPP::chacha20::chacha20 (*const uint8_t * ctx, unsigned int length*)

Constructor for CHACHA20 algorithm with context

Parameters

<i>ctx</i>	- Context to re-initialize the engine with
<i>length</i>	- length of the context in bytes (must be 64)

7.6.1.3 virtual ProtocolPP::chacha20::~chacha20() [inline], [virtual]

Standard Deconstructor.

7.6.2 Member Function Documentation

7.6.2.1 void ProtocolPP::chacha20::context (*uint8_t * context, unsigned int length = 32*)

Function to return CHACHA20 state

Parameters

<i>context</i>	- state of the engine
<i>length</i>	- length up to 64 bytes of state. Length of 32 is needed for AEAD_CHACHA20_POLY1305

7.6.2.2 void ProtocolPP::chacha20::ProcessData (*const uint8_t * input, uint8_t * output, unsigned int length*)

Function to encrypt data with CHACHA20

Parameters

<i>input</i>	- plain/ciphertext
<i>output</i>	- resulting cipher/plaintext
<i>length</i>	- length of the data

The documentation for this class was generated from the following file:

- [include/chacha20.h](#)

7.7 ciphers Class Reference

#include <ciphers.h>

7.7.1 Detailed Description

This class is a wrapper for the Crypto++ library to simplify its interface and hide some of the more esoteric aspects of the interface and returns a static shared_ptr to access the encryption/authentication/crc engine.

The implementation of the modes of operation are found in the class [jmodes](#) as well as any implementation for authentication or CRC32

For API Documentation:

See Also

[ProtocolPP::jmodes](#)

For Addtional Documentation:

See Also

[jmodes](#)

Protocol++ modified by : John Peter Greninger © John Peter Greninger 2015-2019 All Rights Reserved

All copyrights and trademarks are the property of their respective owners

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer
- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution
- Any and all modifications must be returned to John Peter Greninger at GitHub.com <https://github.com/jpgreninger/protocolpp> for evaluation. Inclusion of modifications in the source code shall be determined solely by John Peter Greninger. Failure to provide modifications shall render this license NULL and VOID and revoke any rights to use of Protocol++
- Commercial use requires a fee-based license obtainable at www.protocolpp.com
- Academic use requires written and notarized permission from John Peter and Sheila Rocha Greninger
- US Copyrights at <https://www.copyright.gov/>
 - TXu002059872 (Version 1.0.0)
 - TXu002066632 (Version 1.2.7)
 - TXu002082674 (Version 1.4.0)
 - TXu002097880 (Version 2.0.0)

The names of its contributors may not be used to endorse or promote products derived from this software without specific prior written permission

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE

The documentation for this class was generated from the following file:

- [include/ciphers.h](#)

7.8 ProtocolPP::ciphers Class Reference

```
#include <ciphers.h>
```

Static Public Member Functions

- static std::shared_ptr<jmodes> **get_cipher** (jmodes::cipher_t cipher, jmodes::dir_t direction, jmodes::mode_t mode, uint8_t *key, unsigned int keylen, uint8_t *iv=NULL, unsigned int ivlen=0, unsigned int icvlen=16, uint32_t count=0, uint8_t bearer=0)
- static std::shared_ptr<jmodes> **get_auth** (auth_t auth, uint8_t *authkey=NULL, unsigned int authkeylen=0, jmodes::dir_t dir=jmodes::ENC, uint32_t count=0, uint32_t fresh=0)
- static std::shared_ptr<jmodes> **get_crc32** ()

Factory method to create an instance of CRC32.

7.8.1 Member Function Documentation

7.8.1.1 static std::shared_ptr<jmodes> ProtocolPP::ciphers::get_auth (auth_t auth, uint8_t * authkey = NULL, unsigned int authkeylen = 0, jmodes::dir_t dir = jmodes::ENC, uint32_t count = 0, uint32_t fresh = 0) [static]

Factory method to create an instance of authenticator

Parameters

<i>auth</i>	- POLY1305, SNOWA, ZUCA, MD5, SHA1, SHA224, SHA3_224, SHA256, SHA3_256, SHA384, SHA3_384, SHA512, SHA3_512
<i>authkey</i>	- Key for the authentication engine
<i>authkeylen</i>	- Length of the key for the authentication engine
<i>dir</i>	- direction of processing for SNOW3G and ZUC (UPLINK or DOWNLINK)
<i>count</i>	- Count value for SNOW3G and ZUC
<i>fresh</i>	- Fresh value for SNOW3G and ZUC

7.8.1.2 static std::shared_ptr<jmodes> ProtocolPP::ciphers::get_cipher (jmodes::cipher_t cipher, jmodes::dir_t direction, jmodes::mode_t mode, uint8_t * key, unsigned int keylen, uint8_t * iv = NULL, unsigned int ivlen = 0, unsigned int icvlen = 16, uint32_t count = 0, uint8_t bearer = 0) [static]

Factory method to create an instance of a cipher algorithm

Parameters

<i>cipher</i>	- DES, DES_EDE3, AES, ARIA, Camellia, SM4, SEED, CHACHA20, SNOWE, ZUCE, ARC4
<i>direction</i>	- Direction of processing (ENC or DEC)
<i>mode</i>	- ECB, CBC, CTR, GCM, CCM, CMAC, XCBC_MAC, AUTH, AEAD, STREAM
<i>key</i>	- Encryption key
<i>keylen</i>	- Length of the encryption key
<i>iv</i>	- Initialization vector (IV)
<i>ivlen</i>	- Length of the IV
<i>icvlen</i>	- ICV length
<i>count</i>	- Count value for SNOW3G and ZUC
<i>bearer</i>	- Bearer value for SNOW3G and ZUC

7.8.1.3 static std::shared_ptr<jmodes> ProtocolPP::ciphers::get_crc32() [static]

Factory method to create an instance of CRC32.

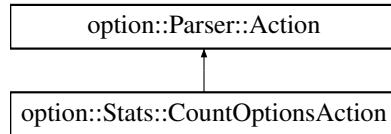
The documentation for this class was generated from the following file:

- include/ciphers.h

7.9 option::Stats::CountOptionsAction Class Reference

```
#include <optionparser.h>
```

Inheritance diagram for option::Stats::CountOptionsAction:



Public Member Functions

- [CountOptionsAction](#) (`unsigned *buffer_max_`)
- `bool perform (Option &)`

Called by Parser::workhorse() for each Option that has been successfully parsed (including unknown options if they have a Descriptor whose Descriptor::check_arg does not return ARG_ILLEGAL).

7.9.1 Constructor & Destructor Documentation

7.9.1.1 `option::Stats::CountOptionsAction::CountOptionsAction (unsigned * buffer_max_) [inline]`

Creates a new [CountOptionsAction](#) that will increase `*buffer_max_` for each parsed [Option](#).

7.9.2 Member Function Documentation

7.9.2.1 `bool option::Stats::CountOptionsAction::perform (Option &) [inline], [virtual]`

Called by Parser::workhorse() for each [Option](#) that has been successfully parsed (including unknown options if they have a [Descriptor](#) whose [Descriptor::check_arg](#) does not return [ARG_ILLEGAL](#)).

Returns `false` iff a fatal error has occurred and the parse should be aborted.

Reimplemented from [option::Parser::Action](#).

The documentation for this class was generated from the following file:

- jtestbench/include/[optionparser.h](#)

7.10 option::Descriptor Struct Reference

Describes an option, its help text (usage) and how it should be parsed.

```
#include <optionparser.h>
```

Public Attributes

- `const unsigned index`

Index of this option's linked list in the array filled in by the parser.

- const int **type**
Used to distinguish between options with the same Protocol++ (ProtocolIPP) Protocol, Encryption, and Authentication Library with Testbench and Drivers. See [Protocol++ \(ProtocolIPP\) Protocol, Encryption, and Authentication Library with Testbench and Drivers](#) for details.
- const char *const **shortopt**
Each char in this string will be accepted as a short option character.
- const char *const **longopt**
The long option name (without the leading -).
- const [CheckArg](#) **check_arg**
For each option that matches [shortopt](#) or [longopt](#) this function will be called to check a potential argument to the option.
- const char * **help**
The usage text associated with the options in this [Descriptor](#).

7.10.1 Detailed Description

Describes an option, its help text (usage) and how it should be parsed.

The main input when constructing an [option::Parser](#) is an array of Descriptors.

Example:

```
* enum OptionIndex {CREATE, ...};
* enum OptionType {DISABLE, ENABLE, OTHER};
*
* const option::Descriptor usage[] = {
*     { CREATE,                                // index
*       OTHER,                                 // type
*       "c",                                   // shortopt
*       "create",                             // longopt
*       Arg::None,                            // check_arg
*       "--create  Tells the program to create something." // help
*   }
*   , ...
* };
*
```

7.10.2 Member Data Documentation

7.10.2.1 const CheckArg option::Descriptor::check_arg

For each option that matches [shortopt](#) or [longopt](#) this function will be called to check a potential argument to the option.

This function will be called even if there is no potential argument. In that case it will be passed NULL as [arg](#) parameter. Do not confuse this with the empty string.

See [CheckArg](#) for more information.

7.10.2.2 const char* option::Descriptor::help

The usage text associated with the options in this [Descriptor](#).

You can use [option::printUsage\(\)](#) to format your usage message based on the [help](#) texts. You can use dummy Descriptors where [shortopt](#) and [longopt](#) are both the empty string to add text to the usage that is not related to a specific option.

See [option::printUsage\(\)](#) for special formatting characters you can use in [help](#) to get a column layout.

Attention

Must be UTF-8-encoded. If your compiler supports C++11 you can use the "u8" prefix to make sure string literals are properly encoded.

7.10.2.3 const unsigned option::Descriptor::index

Index of this option's linked list in the array filled in by the parser.

Command line options whose Descriptors have the same index will end up in the same linked list in the order in which they appear on the command line. If you have multiple long option aliases that refer to the same option, give their descriptors the same `index`.

If you have options that mean exactly opposite things (e.g. `-enable-foo` and `-disable-foo`), you should also give them the same `index`, but distinguish them through different values for `type`. That way they end up in the same list and you can just take the last element of the list and use its type. This way you get the usual behaviour where switches later on the command line override earlier ones without having to code it manually.

Tip:

Use an enum rather than plain ints for better readability, as shown in the example at [Descriptor](#).

7.10.2.4 const char* const option::Descriptor::longopt

The long option name (without the leading `-`).

If this [Descriptor](#) should not have a long option name, use the empty string `" "`. NULL is not permitted here!

While `shortopt` allows multiple short option characters, each [Descriptor](#) can have only a single long option name. If you have multiple long option names referring to the same option use separate Descriptors that have the same [Protocol++ \(ProtocolPP\) Protocol, Encryption, and Authentication Library with Testbench and Drivers](#) and `type`. You may repeat short option characters in such an alias [Descriptor](#) but there's no need to.

Dummy Descriptors:

You can use dummy Descriptors with an empty string for both `shortopt` and `longopt` to add text to the usage that is not related to a specific option. See [help](#). The first dummy [Descriptor](#) will be used for unknown options (see below).

Unknown Option Descriptor:

The first dummy [Descriptor](#) in the list of Descriptors, whose `shortopt` and `longopt` are both the empty string, will be used as the [Descriptor](#) for unknown options. An unknown option is a string in the argument vector that is not a lone minus '`-`' but starts with a minus character and does not match any [Descriptor](#)'s `shortopt` or `longopt`.

Note that the dummy descriptor's `check_arg` function *will* be called and its return value *will* be evaluated as usual. I.e. if it returns `ARG_ILLEGAL` the parsing will be aborted with `Parser::error() == true`.

If `check_arg` does not return `ARG_ILLEGAL` the descriptor's [Protocol++ \(ProtocolPP\) Protocol, Encryption, and Authentication Library with Testbench and Drivers](#) *will* be used to pick the linked list into which to put the unknown option.

If there is no dummy descriptor, unknown options will be dropped silently.

7.10.2.5 const char* const option::Descriptor::shortopt

Each char in this string will be accepted as a short option character.

The string must not include the minus character '`-`' or you'll get undefined behaviour.

If this [Descriptor](#) should not have short option characters, use the empty string `" "`. NULL is not permitted here!

See [longopt](#) for more information.

7.10.2.6 const int option::Descriptor::type

Used to distinguish between options with the same [Protocol++ \(ProtocolPP\) Protocol, Encryption, and Authentication Library with Testbench and Drivers](#). See [Protocol++ \(ProtocolPP\) Protocol, Encryption, and Authentication Library with Testbench and Drivers](#) for details.

It is recommended that you use an enum rather than a plain int to make your code more readable.

The documentation for this struct was generated from the following file:

- jtestbench/include/optionparser.h

7.11 tinyxml2::DynArray< T, INITIAL_SIZE > Class Template Reference

```
#include <tinyxml2.h>
```

Public Member Functions

- [DynArray \(\)](#)
- [~DynArray \(\)](#)
- [void Clear \(\)](#)
- [void Push \(T t\)](#)
- [T * PushArr \(int count\)](#)
- [T Pop \(\)](#)
- [void PopArr \(int count\)](#)
- [bool Empty \(\) const](#)
- [T & operator\[\] \(int i\)](#)
- [const T & operator\[\] \(int i\) const](#)
- [const T & PeekTop \(\) const](#)
- [int Size \(\) const](#)
- [int Capacity \(\) const](#)
- [void SwapRemove \(int i\)](#)
- [const T * Mem \(\) const](#)
- [T * Mem \(\)](#)

7.11.1 Constructor & Destructor Documentation

7.11.1.1 `template<class T, int INITIAL_SIZE> tinyxml2::DynArray< T, INITIAL_SIZE >::DynArray () [inline]`

7.11.1.2 `template<class T, int INITIAL_SIZE> tinyxml2::DynArray< T, INITIAL_SIZE >::~DynArray () [inline]`

7.11.2 Member Function Documentation

7.11.2.1 `template<class T, int INITIAL_SIZE> int tinyxml2::DynArray< T, INITIAL_SIZE >::Capacity () const [inline]`

7.11.2.2 `template<class T, int INITIAL_SIZE> void tinyxml2::DynArray< T, INITIAL_SIZE >::Clear () [inline]`

7.11.2.3 `template<class T, int INITIAL_SIZE> bool tinyxml2::DynArray< T, INITIAL_SIZE >::Empty () const [inline]`

7.11.2.4 `template<class T, int INITIAL_SIZE> const T* tinyxml2::DynArray< T, INITIAL_SIZE >::Mem () const [inline]`

7.11.2.5 `template<class T, int INITIAL_SIZE> T* tinyxml2::DynArray< T, INITIAL_SIZE >::Mem () [inline]`

7.11.2.6 `template<class T, int INITIAL_SIZE> T& tinyxml2::DynArray< T, INITIAL_SIZE >::operator[] (int i) [inline]`

- 7.11.2.7 template<class T, int INITIAL_SIZE> const T& tinyxml2::DynArray< T, INITIAL_SIZE >::operator[] (int *i*) const [inline]
- 7.11.2.8 template<class T, int INITIAL_SIZE> const T& tinyxml2::DynArray< T, INITIAL_SIZE >::PeekTop () const [inline]
- 7.11.2.9 template<class T, int INITIAL_SIZE> T tinyxml2::DynArray< T, INITIAL_SIZE >::Pop () [inline]
- 7.11.2.10 template<class T, int INITIAL_SIZE> void tinyxml2::DynArray< T, INITIAL_SIZE >::PopArr (int *count*) [inline]
- 7.11.2.11 template<class T, int INITIAL_SIZE> void tinyxml2::DynArray< T, INITIAL_SIZE >::Push (T *t*) [inline]
- 7.11.2.12 template<class T, int INITIAL_SIZE> T* tinyxml2::DynArray< T, INITIAL_SIZE >::PushArr (int *count*) [inline]
- 7.11.2.13 template<class T, int INITIAL_SIZE> int tinyxml2::DynArray< T, INITIAL_SIZE >::Size () const [inline]
- 7.11.2.14 template<class T, int INITIAL_SIZE> void tinyxml2::DynArray< T, INITIAL_SIZE >::SwapRemove (int *i*) [inline]

The documentation for this class was generated from the following file:

- [include/tinyxml2.h](#)

7.12 EnumString Class Reference

```
#include "include/EnumString.h"
```

7.12.1 Detailed Description

[EnumString](#) - A utility to provide stringizing support for C++ enums Author: Francis Xavier Joseph Pulikotil

This code is free software: you can do whatever you want with it, although I would appreciate if you gave credit where it's due.

This code is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.

```
// Usage example

// WeekEnd enumeration
enum WeekEnd
{
    Sunday = 1,
    Saturday = 7
};

// String support for WeekEnd
Begin_Enum_String( WeekEnd )
{
    Enum_String( Sunday );
    Enum_String( Saturday );
}
End_Enum_String;

// Convert from WeekEnd to string
const std::string &str = EnumString<WeekEnd>::From( Saturday );
// str should now be "Saturday"
```

```
// Convert from string to WeekEnd
WeekEnd w;
EnumString<WeekEnd>::To( w, "Sunday" );
// w should now be Sunday
```

Protocol++ Modified by : John Peter Greninger June 26, 2016 © John Peter Greninger 2015-2019

All copyrights and trademarks are the property of their respective owners

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer
- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution
- Any and all modifications must be returned to John Peter Greninger at GitHub.com <https://github.com/jpgreninger/protocolpp> for evaluation. Inclusion of modifications in the source code shall be determined solely by John Peter Greninger. Failure to provide modifications shall render this license NULL and VOID and revoke any rights to use of Protocol++
- Commercial use requires a fee-based license obtainable at www.protocolpp.com
- Academic use requires written and notarized permission from John Peter and Sheila Rocha Greninger
- US Copyrights at <https://www.copyright.gov/>
 - TXu002059872 (Version 1.0.0)
 - TXu002066632 (Version 1.2.7)
 - TXu002082674 (Version 1.4.0)
 - TXu002097880 (Version 2.0.0)

The names of its contributors may not be used to endorse or promote products derived from this software without specific prior written permission

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE

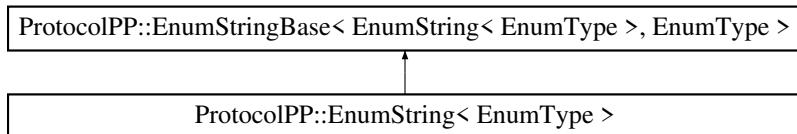
The documentation for this class was generated from the following file:

- include/EnumString.h

7.13 ProtocolPP::EnumString< EnumType > Struct Template Reference

```
#include <EnumString.h>
```

Inheritance diagram for ProtocolPP::EnumString< EnumType >:



Static Public Member Functions

- static void [RegisterEnumerators \(\)](#)

Additional Inherited Members

7.13.1 Member Function Documentation

7.13.1.1 template<class `EnumType`> static void `ProtocolPP::EnumString< EnumType >::RegisterEnumerators ()`
[static]

The documentation for this struct was generated from the following file:

- include/[EnumString.h](#)

7.14 `ProtocolPP::EnumStringBase< DerivedType, EnumType >` Class Template Reference

#include <`EnumString.h`>

Static Public Member Functions

- static const std::string & [From](#) (const `EnumType` e)
- static const bool [To](#) (`EnumType` &e, const std::string &str)
- static `EnumType` [Convert](#) (const std::string &str)
- static int [size \(\)](#)
- static `EnumType` [GetIdx](#) (int index)

Protected Types

- typedef std::map< std::string,
`EnumType` > [AssocMap](#)

Protected Member Functions

- [EnumStringBase \(\)](#)
- [~EnumStringBase \(\)](#)

Static Protected Member Functions

- static void [RegisterEnumerator](#) (const `EnumType` e, const std::string &eStr)

7.14.1 Member Typedef Documentation

7.14.1.1 `template<class DerivedType, class EnumType> typedef std::map<std::string, EnumType> ProtocolPP::EnumStringBase<DerivedType, EnumType>::AssocMap [protected]`

7.14.2 Constructor & Destructor Documentation

7.14.2.1 `template<class DerivedType, class EnumType> ProtocolPP::EnumStringBase<DerivedType, EnumType>::EnumStringBase() [explicit], [protected]`

7.14.2.2 `template<class DerivedType, class EnumType> ProtocolPP::EnumStringBase<DerivedType, EnumType>::~EnumStringBase() [protected]`

7.14.3 Member Function Documentation

7.14.3.1 `template<class D, class E> E ProtocolPP::EnumStringBase<D, E>::Convert(const std::string & str) [static]`

7.14.3.2 `template<class D, class E> const std::string & ProtocolPP::EnumStringBase<D, E>::From(const E e) [static]`

7.14.3.3 `template<class D, class E> E ProtocolPP::EnumStringBase<D, E>::GetIdx(int index) [static]`

7.14.3.4 `template<class D, class E> void ProtocolPP::EnumStringBase<D, E>::RegisterEnumerator(const E e, const std::string & eStr) [static], [protected]`

7.14.3.5 `template<class DerivedType, class EnumType> static int ProtocolPP::EnumStringBase<DerivedType, EnumType>::size() [inline], [static]`

7.14.3.6 `template<class D, class E> const bool ProtocolPP::EnumStringBase<D, E>::To(E & e, const std::string & str) [static]`

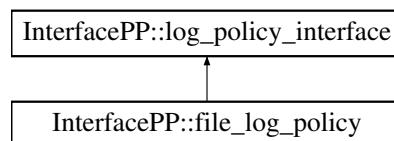
The documentation for this class was generated from the following file:

- include/EnumString.h

7.15 InterfacePP::file_log_policy Class Reference

```
#include <jlogger.h>
```

Inheritance diagram for InterfacePP::file_log_policy:



Public Member Functions

- [file_log_policy\(\)](#)
file_log_policy constructor
- `void open_ostream(const std::string &name)`
- `void close_ostream()`

- close_ostream - close the log file*
- void [write](#) (const std::string &msg)
 - [~file_log_policy \(\)](#)
- standard deconstructor*

7.15.1 Constructor & Destructor Documentation

7.15.1.1 InterfacePP::file_log_policy::file_log_policy() [inline]

[file_log_policy](#) constructor

7.15.1.2 InterfacePP::file_log_policy::~file_log_policy()

standard deconstructor

7.15.2 Member Function Documentation

7.15.2.1 void InterfacePP::file_log_policy::close_ostream() [virtual]

close_ostream - close the log file

Implements [InterfacePP::log_policy_interface](#).

7.15.2.2 void InterfacePP::file_log_policy::open_ostream(const std::string & name) [virtual]

open_ostream

Parameters

<i>name</i>	- name of the log output file
-------------	-------------------------------

Implements [InterfacePP::log_policy_interface](#).

7.15.2.3 void InterfacePP::file_log_policy::write(const std::string & msg) [virtual]

write - write msg to log file

Parameters

<i>msg</i>	- message to write to log file
------------	--------------------------------

Implements [InterfacePP::log_policy_interface](#).

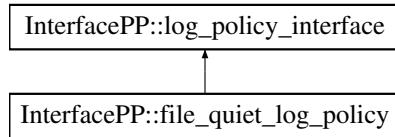
The documentation for this class was generated from the following file:

- jlogger/include/jlogger.h

7.16 InterfacePP::file_quiet_log_policy Class Reference

```
#include <jlogger.h>
```

Inheritance diagram for InterfacePP::file_quiet_log_policy:



Public Member Functions

- [file_quiet_log_policy \(\)](#)
file_log_policy constructor
- [void open_ostream \(const std::string &name\)](#)
- [void close_ostream \(\)](#)
close_ostream - close the log file
- [void write \(const std::string &msg\)](#)
- [~file_quiet_log_policy \(\)](#)
standard deconstructor

7.16.1 Constructor & Destructor Documentation

7.16.1.1 [InterfacePP::file_quiet_log_policy::file_quiet_log_policy\(\)](#) [inline]

[file_log_policy](#) constructor

7.16.1.2 [InterfacePP::file_quiet_log_policy::~file_quiet_log_policy\(\)](#)

standard deconstructor

7.16.2 Member Function Documentation

7.16.2.1 [void InterfacePP::file_quiet_log_policy::close_ostream\(\)](#) [virtual]

`close_ostream` - close the log file

Implements [InterfacePP::log_policy_interface](#).

7.16.2.2 [void InterfacePP::file_quiet_log_policy::open_ostream \(const std::string & name \)](#) [virtual]

`open_ostream`

Parameters

<code>name</code>	- name of the log output file
-------------------	-------------------------------

Implements [InterfacePP::log_policy_interface](#).

7.16.2.3 [void InterfacePP::file_quiet_log_policy::write \(const std::string & msg \)](#) [virtual]

`write` - write msg to log file

Parameters

<i>msg</i>	- message to write to log file
------------	--------------------------------

Implements [InterfacePP::log_policy_interface](#).

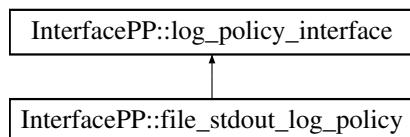
The documentation for this class was generated from the following file:

- [jlogger/include/jlogger.h](#)

7.17 InterfacePP::file_stdout_log_policy Class Reference

```
#include <jlogger.h>
```

Inheritance diagram for InterfacePP::file_stdout_log_policy:



Public Member Functions

- **file_stdout_log_policy ()**
file_log_policy constructor
- void **open_ostream** (const std::string &name)
- void **close_ostream ()**
close_ostream - close the log file
- void **write** (const std::string &msg)
- **~file_stdout_log_policy ()**
standard deconstructor

7.17.1 Constructor & Destructor Documentation

7.17.1.1 InterfacePP::file_stdout_log_policy::file_stdout_log_policy () [inline]

[file_log_policy](#) constructor

7.17.1.2 InterfacePP::file_stdout_log_policy::~file_stdout_log_policy ()

standard deconstructor

7.17.2 Member Function Documentation

7.17.2.1 void InterfacePP::file_stdout_log_policy::close_ostream () [virtual]

close_ostream - close the log file

Implements [InterfacePP::log_policy_interface](#).

7.17.2.2 void InterfacePP::file_stdout_log_policy::open_ostream (const std::string & name) [virtual]

open_ostream

Parameters

<i>name</i>	- name of the log output file
-------------	-------------------------------

Implements [InterfacePP::log_policy_interface](#).

7.17.2.3 void [InterfacePP::file_stdout_log_policy::write](#) (const std::string & *msg*) [virtual]

write - write msg to log file

Parameters

<i>msg</i>	- message to write to log file
------------	--------------------------------

Implements [InterfacePP::log_policy_interface](#).

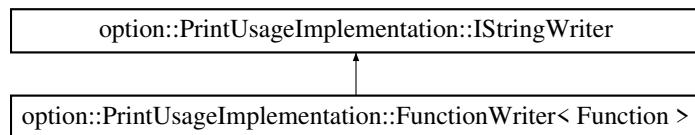
The documentation for this class was generated from the following file:

- [jlogger/include/jlogger.h](#)

7.18 option::PrintUsageImplementation::FunctionWriter< Function > Struct Template Reference

```
#include <optionparser.h>
```

Inheritance diagram for option::PrintUsageImplementation::FunctionWriter< Function >:



Public Member Functions

- virtual void [operator\(\)](#) (const char *str, int size)
Writes the given number of chars beginning at the given pointer somewhere.
- [FunctionWriter](#) (Function *w)

Public Attributes

- Function * [write](#)

7.18.1 Constructor & Destructor Documentation

7.18.1.1 template<typename Function> option::PrintUsageImplementation::FunctionWriter< Function >::[FunctionWriter](#) (Function * w) [inline]

7.18.2 Member Function Documentation

7.18.2.1 template<typename Function> virtual void [option::PrintUsageImplementation::FunctionWriter< Function >::operator\(\)](#) (const char *, int) [inline], [virtual]

Writes the given number of chars beginning at the given pointer somewhere.

Reimplemented from [option::PrintUsageImplementation::IStringWriter](#).

7.18.3 Member Data Documentation

7.18.3.1 template<typename Function> Function* option::PrintUsageImplementation::FunctionWriter< Function >::write

The documentation for this struct was generated from the following file:

- jtestbench/include/[optionparser.h](#)

7.19 ikev2 Class Reference

```
#include "schema/ikev2.xsd"
```

7.19.1 Detailed Description

7.19.2 IKEv2 Configuration Schema

For API information:

See Also

[ProtocolPP::jikev2](#)
[ProtocolPP::jikev2sa](#)
[ProtocolPP::jikeparse](#)
[ProtocolPP::ikev2](#)

For Additional Documentation:

See Also

[jikev2](#)
[jikev2sa](#)
[jikeparse](#)
[ikev2](#)

Protocol++ written by : John Peter Greninger © John Peter Greninger 2015-2019 All Rights Reserved

All copyrights and trademarks are the property of their respective owners

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer
- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution
- Any and all modifications must be returned to John Peter Greninger at GitHub.com <https://github.com/jpgreninger/protocolpp> for evaluation. Inclusion of modifications in the source code shall be determined solely by John Peter Greninger. Failure to provide modifications shall render this license NULL and VOID and revoke any rights to use of Protocol++
- Commercial use requires a fee-based license obtainable at www.protocolpp.com
- Academic use requires written and notarized permission from John Peter and Sheila Rocha Greninger

- US Copyrights at <https://www.copyright.gov/>
 - TXu002059872 (Version 1.0.0)
 - TXu002066632 (Version 1.2.7)
 - TXu002082674 (Version 1.4.0)
 - TXu002097880 (Version 2.0.0)

The names of its contributors may not be used to endorse or promote products derived from this software without specific prior written permission

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE

The documentation for this class was generated from the following file:

- schema/kev2.xsd

7.20 interfaceapp Class Reference

```
#include "include/interfaceapp.h"
```

7.20.1 Detailed Description

Protocol++ modified by : John Peter Greninger © John Peter Greninger 2015-2019 All Rights Reserved

All copyrights and trademarks are the property of their respective owners

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer
- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution
- Any and all modifications must be returned to John Peter Greninger at GitHub.com <https://github.com/jpgreninger/protocolpp> for evaluation. Inclusion of modifications in the source code shall be determined solely by John Peter Greninger. Failure to provide modifications shall render this license NULL and VOID and revoke any rights to use of Protocol++
- Commercial use requires a fee-based license obtainable at www.protocolpp.com
- Academic use requires written and notarized permission from John Peter and Sheila Rocha Greninger
- US Copyrights at <https://www.copyright.gov/>
 - TXu002059872 (Version 1.0.0)
 - TXu002066632 (Version 1.2.7)
 - TXu002082674 (Version 1.4.0)

- TXu002097880 (Version 2.0.0)

The names of its contributors may not be used to endorse or promote products derived from this software without specific prior written permission

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE

The documentation for this class was generated from the following file:

- jtestbench/include/interfaceapp.h

7.21 option::PrintUsageImplementation::IStringWriter Struct Reference

```
#include <optionparser.h>
```

Inheritance diagram for option::PrintUsageImplementation::IStringWriter:



Public Member Functions

- virtual void [operator\(\)](#) (const char *, int)

Writes the given number of chars beginning at the given pointer somewhere.

7.21.1 Member Function Documentation

7.21.1.1 virtual void option::PrintUsageImplementation::IStringWriter::operator() (const char * , int) [inline], [virtual]

Writes the given number of chars beginning at the given pointer somewhere.

Reimplemented in [option::PrintUsageImplementation::StreamWriter< Function, Stream >](#), [option::PrintUsageImplementation::SyscallWriter< Syscall >](#), [option::PrintUsageImplementation::TemporaryWriter< Temporary >](#), [option::PrintUsageImplementation::OStreamWriter< OStream >](#), and [option::PrintUsageImplementation::FunctionWriter< Function >](#).

The documentation for this struct was generated from the following file:

- jtestbench/include/optionparser.h

7.22 jarray Class Reference

```
#include "include/jarray.h"
```

7.22.1 Detailed Description

Protocol++ written by : John Peter Greninger © John Peter Greninger 2015-2019 All Rights Reserved

All copyrights and trademarks are the property of their respective owners

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer
- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution
- Any and all modifications must be returned to John Peter Greninger at GitHub.com <https://github.com/jpgreninger/protocolpp> for evaluation. Inclusion of modifications in the source code shall be determined solely by John Peter Greninger. Failure to provide modifications shall render this license NULL and VOID and revoke any rights to use of Protocol++
- Commercial use requires a fee-based license obtainable at www.protocolpp.com
- Academic use requires written and notarized permission from John Peter and Sheila Rocha Greninger
- US Copyrights at <https://www.copyright.gov/>
 - TXu002059872 (Version 1.0.0)
 - TXu002066632 (Version 1.2.7)
 - TXu002082674 (Version 1.4.0)
 - TXu002097880 (Version 2.0.0)

The names of its contributors may not be used to endorse or promote products derived from this software without specific prior written permission

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE

The documentation for this class was generated from the following file:

- [include/jarray.h](#)

7.23 ProtocolPP::jarray< T > Class Template Reference

```
#include <jarray.h>
```

Public Member Functions

- [jarray \(\)](#)
Standard constructor for jarray class.
- [jarray \(unsigned int size\)](#)

- `jarray` (`unsigned int size, T value, endian_t format=BIG`)
- `jarray` (`const jarray< T > &rhs, endian_t format=BIG`)
- `jarray` (`const std::string &rhs, endian_t format=BIG, bool encoded=false`)
- `jarray` (`const std::vector< T > &rhs, endian_t format=BIG`)
- `jarray` (`const uint8_t *rhs, unsigned int size, endian_t format=BIG`)
- template<typename TT>
 `jarray(jarray< TT > &rhs, endian_t format=BIG)`
- virtual `~jarray()`
Standard deconstructor for jarray.
- `unsigned int get_size()`
- `bool empty()`
- `jarray< T > split(unsigned int elements)`
- `jarray< T > extract(unsigned int start, unsigned int elements)`
- `void push_back(T rhs)`
- `void pop_back()`
- `void resize(unsigned int newsize)`
- `void append(jarray< T > rhs)`
- `void insert(unsigned int index, jarray< T > &rhs)`
- `void update(unsigned int index, jarray< T > &rhs)`
- `void erase(unsigned int index, unsigned int size)`
- `void reverse()`
Method to reverse the elements of the array.
- `T * get_ptr()`
- `const T * get_ptr() const`
- `std::string to_string(bool printable=false, int indent=10, bool xml=false)`
- `std::string debug(jarray< T > &expect, int indent=10)`
- `bool operator==(jarray< T > &rhs)`
- `bool operator==(const jarray< T > &rhs) const`
- `bool operator!=(jarray< T > &rhs)`
- `bool operator!=(const jarray< T > &rhs) const`
- `void operator^=(jarray< T > &rhs)`
- `void operator|=(jarray< T > &rhs)`
- `void operator&=(jarray< T > &rhs)`
- `void operator*=(jarray< T > &rhs)`
- `void operator+==(jarray< T > &rhs)`
- `T & operator[](unsigned int index)`
- `const T & operator[](unsigned int index) const`
- `endian_t get_format()`

7.23.1 Constructor & Destructor Documentation

7.23.1.1 template<typename T> jarray::jarray()

Standard constructor for jarray class.

7.23.1.2 template<typename T> jarray::jarray(unsigned int size)

Constructor for jarray initialized to size

Parameters

<i>size</i>	- size to initialize the array to
-------------	-----------------------------------

7.23.1.3 template<typename T> jarray::jarray (*unsigned int size, T value, endian_t format = BIG*)

Constructor for jarray initialized to size with value

Parameters

<i>size</i>	- size to initialize the array to
<i>value</i>	- value to initialize the array to
<i>format</i>	- endianness of the constructed array (BIG, LITTLE)

7.23.1.4 template<typename T> jarray::jarray (*const jarray< T > & rhs, endian_t format = BIG*)

Copy Constructor for jarray

Parameters

<i>rhs</i>	- array to copy
<i>format</i>	- endianness of the constructed array (BIG, LITTLE)

7.23.1.5 template<typename T> jarray::jarray (*const std::string & rhs, endian_t format = BIG, bool encoded = false*)

Constructor for jarray from a string

Parameters

<i>rhs</i>	- string to populate the array with
<i>format</i>	- endianness of the constructed array (BIG, LITTLE)
<i>encoded</i>	- Input string has already been encoded as HEX characters instead of being a simple string. For example, if the input string had been encoded as bytes each character in the string would be equal to something like '0x6C' or '0xAE'. If the input was a simple string each character would represent a single value such that the string "123ABC" is really "1", "2" "3", "A", "B", "C" and needs to be converted and grouped into HEX such that the resulting array has "0x12", "0x3A", and "0xBC". The Crypto++ library uses encoded strings. Set encoded to true to read in ASCII characters

7.23.1.6 template<typename T> jarray::jarray (*const std::vector< T > & rhs, endian_t format = BIG*)

Constructor for jarray from a standard vector

Parameters

<i>rhs</i>	- vector to initialize the array with
<i>format</i>	- endianness of the constructed array (BIG, LITTLE)

7.23.1.7 template<typename T> jarray::jarray (*const uint8_t * rhs, unsigned int size, endian_t format = BIG*)

Constructor for jarray from a standard vector

Parameters

<i>rhs</i>	- byte pointer to initialize the array with
<i>size</i>	- length of the data to convert
<i>format</i>	- endianness of the constructed array (BIG, LITTLE)

```
7.23.1.8 template<typename T> template<typename TT> jarray::jarray( jarray<TT> &rhs, endian_t format = BIG )
```

Constructor for jarray from one type to another

Parameters

<i>rhs</i>	- initial array to convert. Conversion is to type of array that is being constructed. Example: <pre>* jarray<uint32_t> oldarray("AABBCCDD"); * jarray<uint8_t> newarray(oldarray, LITTLE); * newarray.to_string(); * "DDCCBBAA" *</pre>
<i>format</i>	- endianness of the constructed array (BIG, LITTLE)

7.23.1.9 template<typename T> virtual ProtocolIPP::jarray< T >::jarray() [inline], [virtual]

Standard deconstructor for jarray.

7.23.2 Member Function Documentation

7.23.2.1 template<typename T> void jarray::append (jarray< T > rhs)

Method to add the array rhs to the current array

Parameters

rhs - array to add to this array

```
7.23.2.2 template<typename T> std::string jarray::debug( jarray< T > & expect, int indent = 10 )
```

Method to compare arrays side by side for debug as shown below prints to std::err

Parameters

expect - Expected data to print

<i>indent</i>	- default indentation for pretty printing
---------------	---

Returns

string representation of this array

7.23.2.3 template<typename T > bool jarray::empty()

Method to determine if the array is empty

Returns

True/False if array is empty

7.23.2.4 template<typename T> void jarray::erase(unsigned int index, unsigned int size)

Method to remove the elements from the array starting at index and continuing for size. The original array is reduced by the number of elements requested

Parameters

<i>index</i>	- location to start erasing elements
<i>size</i>	- number of elements to remove from this array

7.23.2.5 template<typename T > jarray< T > jarray::extract(unsigned int start, unsigned int elements)

Method to extract the number of elements from the array starting at the start value

Parameters

<i>start</i>	- index to start the extraction
<i>elements</i>	- number of elements to split from the array

Returns

new array containing the split off values

7.23.2.6 template<typename T > endian_t ProtocolPP::jarray< T >::get_format() [inline]

Method to return endianness format of this array

Returns

endianness format for this array

7.23.2.7 template<typename T > T * jarray::get_ptr()

Method to return the pointer to the underlying data object of the array

Returns

pointer to the data object

7.23.2.8 template<typename T> const T * jarray::get_ptr() const

Method to return the pointer to the underlying data object of the array

Returns

pointer to the data object

7.23.2.9 template<typename T> unsigned int jarray::get_size()

Method to retrieve size of this array

Returns

Size of the array

7.23.2.10 template<typename T> void jarray::insert(unsigned int index, jarray< T > & rhs)

Method to insert the array rhs into this array starting at index. The array is resized to accomodate this array plus the new array. No elements of the original array are overwritten.

Parameters

<i>index</i>	- location to start the insertion
<i>rhs</i>	- array to insert into this array

7.23.2.11 template<typename T> bool jarray::operator!= (jarray< T > & rhs)

Method to compare two different arrays. First compares array size then iterates through all array values and compares them for inequality

Parameters

<i>rhs</i>	- array to compare with this array
------------	------------------------------------

Returns

true/false value for array comparison

7.23.2.12 template<typename T> bool jarray::operator!= (const jarray< T > & rhs) const

Method to compare two different arrays. First compares array size then iterates through all array values and compares them for inequality

Parameters

<i>rhs</i>	- array to compare with this array
------------	------------------------------------

Returns

true/false value for array comparison

7.23.2.13 template<typename T> void jarray::operator&= (jarray< T > & rhs)

Method to AND two different arrays

Parameters

<i>rhs</i>	- array to AND with this array
------------	--------------------------------

7.23.2.14 template<typename T> void jarray::operator*=((jarray< T > & *rhs*)

Method to MULTIPLY two different arrays

Parameters

<i>rhs</i>	- array to MULTIPLY with this array
------------	-------------------------------------

7.23.2.15 template<typename T> void jarray::operator+= (jarray< T > & *rhs*)

Method to ADD two different arrays

Parameters

<i>rhs</i>	- array to ADD with this array
------------	--------------------------------

7.23.2.16 template<typename T> bool jarray::operator==((jarray< T > & *rhs*)

Method to compare two different arrays. First compares array size then iterates through all array values and compares them for equality

Parameters

<i>rhs</i>	- array to compare with this array
------------	------------------------------------

Returns

true/false value for array comparison

7.23.2.17 template<typename T> bool jarray::operator==((const jarray< T > & *rhs*) const

Method to compare two different arrays. First compares array size then iterates through all array values and compares them for equality

Parameters

<i>rhs</i>	- array to compare with this array
------------	------------------------------------

Returns

true/false value for array comparison

7.23.2.18 template<typename T > T & jarray::operator[]((unsigned int *index*)

Method to return array value at index

Parameters

<i>index</i>	- location of array element to return
--------------	---------------------------------------

Returns

array element at index

7.23.2.19 template<typename T > const T & jarray::operator[](unsigned int *index*) const

Method to return const array value at index

Parameters

<i>index</i>	- location of array element to return
--------------	---------------------------------------

Returns

const array element at index

7.23.2.20 template<typename T> void jarray::operator^=(jarray< T > & rhs)

Method to XOR two different arrays

Parameters

<i>rhs</i>	- array to XOR with this array
------------	--------------------------------

7.23.2.21 template<typename T> void jarray::operator|=(jarray< T > & rhs)

Method to OR two different arrays

Parameters

<i>rhs</i>	- array to OR with this array
------------	-------------------------------

7.23.2.22 template<typename T > void jarray::pop_back()

Method to removed the last element from the array. The element is destroyed in the process

7.23.2.23 template<typename T> void jarray::push_back(T rhs)

Method to add a single element to the end of the array

Parameters

<i>rhs</i>	- element to add to this array
------------	--------------------------------

7.23.2.24 template<typename T > void jarray::resize(unsigned int *newsize*)

Method to change the array size

Parameters

<i>newsize</i>	- either shrink or expand the array to the new size
----------------	---

7.23.2.25 template<typename T > void jarray::reverse ()

Method to reverse the elements of the array.

7.23.2.26 template<typename T > jarray< T > jarray::split (unsigned int *elements*)

Method to split the number of elements from the front of the array and return it to the caller. The original array is reduced by the number of elements split from it

Parameters

<i>elements</i>	- number of elements to split from the array
-----------------	--

Returns

new array containing the split off values

7.23.2.27 template<typename T > std::string jarray::to_string (bool *printable* = false, int *indent* = 10, bool *xml* = false)

Method to convert this array into a string representation for printing purposes. Output is formatted either as a single, long string of HEX characters if 'printable' is false or is a pretty printed 16-bytes across with underscores as follows if 'printable' is true

```
* [AAAABBBB_CCCCCDDD_EEEEEE_00001111
* 22223333_44445555_66667777_88889999
* 22223333_44445555_66667777_88889999
* 22223333_44445555_66667777_88889999
* 22223333_44445555_66667777_88889999
* 22223333_44445555_66667777_88889999
* 22223333_44445555_66667777_88889999]
```

Parameters

<i>printable</i>	- if false, output is a single string of HEX characters. If true, enables pretty_printing of the array with formatting
<i>indent</i>	- default indentation for pretty printing ignored if printable is false
<i>xml</i>	<p>- Print as an XML data object as found below</p> <pre>* 22223333444455556666777788889999 * 2222333344445555666677778888 *</pre>

Returns

string representation of this array

7.23.2.28 template<typename T> void jarray::update (unsigned int *index*, jarray< T > & *rhs*)

Method to update the array rhs into this array starting at index. The array is resized to accomodate this array plus the new array. Elements of the original array starting at index and extending for the length of rhs are overwritten with the new values

Parameters

<i>index</i>	- location to start the update
<i>rhs</i>	- array to update this array with

The documentation for this class was generated from the following file:

- include/jarray.h

7.24 jdata Class Reference

```
#include "include/jdata.h"
```

7.24.1 Detailed Description

Protocol++ modified by : John Peter Greninger © John Peter Greninger 2015-2019 All Rights Reserved

All copyrights and trademarks are the property of their respective owners

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer
- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution
- Any and all modifications must be returned to John Peter Greninger at GitHub.com <https://github.com/jpgreninger/protocolpp> for evaluation. Inclusion of modifications in the source code shall be determined solely by John Peter Greninger. Failure to provide modifications shall render this license NULL and VOID and revoke any rights to use of Protocol++
- Commercial use requires a fee-based license obtainable at www.protocolpp.com
- Academic use requires written and notarized permission from John Peter and Sheila Rocha Greninger
- US Copyrights at <https://www.copyright.gov/>
 - TXu002059872 (Version 1.0.0)
 - TXu002066632 (Version 1.2.7)
 - TXu002082674 (Version 1.4.0)
 - TXu002097880 (Version 2.0.0)

The names of its contributors may not be used to endorse or promote products derived from this software without specific prior written permission

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE

The documentation for this class was generated from the following file:

- include/jdata.h

7.25 ProtocolPP::jdata Class Reference

```
#include <jdata.h>
```

Public Member Functions

- **jdata ()**
constructor for JDATA object
- **jdata (std::map< std::string, std::shared_ptr< jstream >> &flows, std::map< std::string, std::list< std::shared_ptr< jpacket >>> &pkts)**
- **void set_flow (std::shared_ptr< jstream > &newstream)**
- **void set_packet (std::string &stream, std::shared_ptr< jpacket > &newpkt)**
- **void get_flow (std::string &name, std::shared_ptr< jstream > &flow)**
- **void get_packet (std::string &stream, std::shared_ptr< jpacket > &packet)**
- **std::map< std::string, std::shared_ptr< jstream >> get_flows ()**
- **std::map< std::string, std::list< std::shared_ptr< jpacket >>> get_packets ()**
- **unsigned int get_flowcnt ()**
- **unsigned int get_pktcnt ()**

7.25.1 Constructor & Destructor Documentation

7.25.1.1 **ProtocolPP::jdata::jdata () [inline]**

constructor for JDATA object

7.25.1.2 **ProtocolPP::jdata::jdata (std::map< std::string, std::shared_ptr< jstream >> & flows, std::map< std::string, std::list< std::shared_ptr< jpacket >>> & pkts) [inline]**

constructor for JDATA object

Parameters

<i>flows</i>	- std::map<std::string, std::shared_ptr<jstream>>
<i>pkts</i>	- std::list<std::shared_ptr<jpacket>>

7.25.2 Member Function Documentation

7.25.2.1 **void ProtocolPP::jdata::get_flow (std::string & name, std::shared_ptr< jstream > & flow) [inline]**

returns the specified flow

Parameters

<i>name</i>	- name of the flow to return
<i>flow</i>	- pointer to flow

7.25.2.2 **unsigned int ProtocolPP::jdata::get_flowcnt () [inline]**

returns the number of flows left

Returns

- number of flows in the data object

7.25.2.3 `std::map<std::string, std::shared_ptr<jstream> > ProtocolPP::jdata::get_flows() [inline]`

returns the map with all flows

Returns

- map with all flows

7.25.2.4 `void ProtocolPP::jdata::get_packet(std::string & stream, std::shared_ptr<jpacket> & packet) [inline]`

returns the next packet

Returns

- pointer to packet

7.25.2.5 `std::map<std::string, std::list<std::shared_ptr<jpacket> > > ProtocolPP::jdata::get_packets() [inline]`

returns the map with all packets

Returns

- map with all packets

7.25.2.6 `unsigned int ProtocolPP::jdata::get_pktnum() [inline]`

returns the number of packets left

Returns

- number of packets in the data object

7.25.2.7 `void ProtocolPP::jdata::set_flow(std::shared_ptr<jstream> & newstream) [inline]`

adds the flow

Parameters

<code>newstream</code>	- adds a flow to the jdata object
------------------------	-----------------------------------

7.25.2.8 `void ProtocolPP::jdata::set_packet(std::string & stream, std::shared_ptr<jpacket> & newpkt) [inline]`

adds the packet

Parameters

<i>stream</i>	- name of the stream the packet belongs to
<i>newpkt</i>	- adds a packet to the jdata object

The documentation for this class was generated from the following file:

- [include/jdata.h](#)

7.26 jdirectdrive Class Reference

```
#include "include/jdirectdrive.h"
```

7.26.1 Detailed Description

7.26.2 Driver for Protocol++

See Also

[jdata](#)
[jpacket](#)
[jstream](#)
[jprotocolpp](#)

Protocol++ written by : John Peter Greninger © John Peter Greninger 2015-2019 All Rights Reserved

All copyrights and trademarks are the property of their respective owners

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer
- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution
- Any and all modifications must be returned to John Peter Greninger at GitHub.com for evaluation. Inclusion of modifications in the source code shall be determined solely by John Peter Greninger. Failure to provide modifications shall render this license NULL and VOID and revoke any rights to use of Protocol++
- Commercial use requires a fee-based license obtainable at www.protocolpp.com
- US Copyrights at <https://www.copyright.gov/>
 - TXU002059872 (Version 1.0.0)
 - TXU002066632 (Version 1.2.7)
 - TXU002082674 (Version 1.4.0)
 - TXU002097880 (Version 2.0.0)

The names of its contributors may not be used to endorse or promote products derived from this software without specific prior written permission

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL

THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE

The documentation for this class was generated from the following file:

- drivers/include/jdirectdrive.h

7.27 DriverPP::jdirectdrive Class Reference

```
#include <jdirectdrive.h>
```

Public Member Functions

- **jdirectdrive** (bool *listen*, int *port*, [ProtocolPP::protocol_t](#) *prot*, [ProtocolPP::iana_t](#) *lvl3*, [ProtocolPP::protocol_t](#) *lvl2*, bool *lvl3_sec*, std::string &*host*, unsigned long *iqlsize*, unsigned long *oqlsize*, unsigned long *seed*, [ProtocolPP::endian_t](#) *endianess*, std::shared_ptr<[InterfacePP::jmmu](#)> &*mmu*, std::shared_ptr<[InterfacePP::jlogger](#)> &*logger*)
 • virtual ~**jdirectdrive** ()
 standard deconstructor
 • uint64_t **write_mem** (uint8_t **data*, unsigned int *length*)
 • unsigned int **avail** ()
 • uint32_t **push** (uint8_t **packet*, unsigned int *size*)
 • uint32_t **push** (std::shared_ptr<[ProtocolPP::jarray](#)< uint8_t >> &*input*)
 • void **runsnd** ()
 • uint32_t **pop** (uint8_t **output*, unsigned int *size*)
 • uint32_t **pop** (std::shared_ptr<[ProtocolPP::jarray](#)< uint8_t >> &*output*)
 • void **runrcv** ()
 • unsigned int **get_size** ()
 • void **teardown** ()
 terminate the driver

7.27.1 Constructor & Destructor Documentation

7.27.1.1 **DriverPP::jdirectdrive::jdirectdrive** (bool *listen*, int *port*, [ProtocolPP::protocol_t](#) *prot*, [ProtocolPP::iana_t](#) *lvl3*, [ProtocolPP::protocol_t](#) *lvl2*, bool *lvl3_sec*, std::string & *host*, unsigned long *iqlsize*, unsigned long *oqlsize*, unsigned long *seed*, [ProtocolPP::endian_t](#) *endianess*, std::shared_ptr<[InterfacePP::jmmu](#)> &*mmu*, std::shared_ptr<[InterfacePP::jlogger](#)> &*logger*)

jdirectdrive uses protocol++ to drive packets into a socket which can then be read by any device. Application layer protocols supported are TLS, SRTP, TCP, and UDP

Parameters

<i>listen</i>	- if set to true, configures to listen on the port specified
<i>port</i>	- port number
<i>prot</i>	- transport or application level protocol (TCP, UDP, TLS, SRTP)

<i>lvl3</i>	- network layer protocol (IPV4, IPV6)
<i>lvl2</i>	- data-link layer protocol (MACSEC, WIFI, WIMAX, LTE)
<i>lvl3_sec</i>	- Enable ESP for network level
<i>host</i>	- string of the host name (or IP Address in standard byte format XX.XX.XX...)
<i>iqlsize</i>	- Size of the send queue
<i>oqlsize</i>	- Size of the receive queue
<i>seed</i>	- seed for the randomizer
<i>endianess</i>	- Endianess of the platform to support
<i>mmu</i>	- tracks dynamic memory
<i>logger</i>	- object for writing

7.27.1.2 virtual DriverPP::jdirectdrive::~jdirectdrive() [inline], [virtual]

standard deconstructor

7.27.2 Member Function Documentation

7.27.2.1 unsigned int DriverPP::jdirectdrive::avail()

Returns

Number of slots available in input queue

7.27.2.2 unsigned int DriverPP::jdirectdrive::get_size()

Returns the size of the next output packet

Returns

- size of the next output packet

7.27.2.3 uint32_t DriverPP::jdirectdrive::pop(uint8_t * *output*, unsigned int *size*)

Returns the next packet in the receive queue

Parameters

<i>output</i>	- payload from decapsulated packet
<i>size</i>	- size of the output pointer

7.27.2.4 uint32_t DriverPP::jdirectdrive::pop(std::shared_ptr< ProtocolPP::jarray< uint8_t >> & *output*)

Returns the next packet in the receive queue

Parameters

<i>output</i>	- payload from decapsulated packet
---------------	------------------------------------

7.27.2.5 uint32_t DriverPP::jdirectdrive::push(uint8_t * *packet*, unsigned int *size*)

Issues the packet into the send queue

Parameters

<i>packet</i>	- byte array for the input payload
<i>size</i>	- length of the input data

Returns

- status of push command

7.27.2.6 uint32_t DriverPP::jdirectdrive::push (std::shared_ptr< ProtocolPP::jarray< uint8_t >> & *input*)

Issues the packet into the send queue

Parameters

<i>input</i>	- byte array for the input payload
--------------	------------------------------------

Returns

- status of push command

7.27.2.7 void DriverPP::jdirectdrive::runrcv ()

should be called in a separate thread, constantly checks the receive socket for new packets, processes them, and places the processed packet into the receive queue

7.27.2.8 void DriverPP::jdirectdrive::runsnd ()

should be called in a separate thread, constantly send any packets found in the input queue through the send socket

7.27.2.9 void DriverPP::jdirectdrive::teardown ()

terminate the driver

7.27.2.10 uint64_t DriverPP::jdirectdrive::write_mem (uint8_t * *data*, unsigned int *length*)

Write data to memory

Parameters

<i>data</i>	- Data to write to memory
<i>length</i>	- length of data to write

Returns

- address - Address data written to

The documentation for this class was generated from the following file:

- drivers/include/jdirectdrive.h

7.28 jdrive Class Reference

```
#include "include/jdrive.h"
```

7.28.1 Detailed Description

7.28.2 Driver for Protocol++

See Also

[jdata](#)
[jpacket](#)
[jstream](#)
[jprotocolpp](#)

Protocol++ written by : John Peter Greninger © John Peter Greninger 2015-2019 All Rights Reserved

All copyrights and trademarks are the property of their respective owners

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer
- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution
- Any and all modifications must be returned to John Peter Greninger at GitHub.com for evaluation. Inclusion of modifications in the source code shall be determined solely by John Peter Greninger. Failure to provide modifications shall render this license NULL and VOID and revoke any rights to use of Protocol++
- Commercial use requires a fee-based license obtainable at www.protocolpp.com
- US Copyrights at <https://www.copyright.gov/>
 - TXU002059872 (Version 1.0.0)
 - TXU002066632 (Version 1.2.7)
 - TXU002082674 (Version 1.4.0)
 - TXU002097880 (Version 2.0.0)

The names of its contributors may not be used to endorse or promote products derived from this software without specific prior written permission

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE

The documentation for this class was generated from the following file:

- drivers/include/jdrive.h

7.29 DriverPP::jdrive Class Reference

```
#include <jdrive.h>
```

Public Member Functions

- `jdrive` (bool *listen*, int *port*, `ProtocolPP::protocol_t` *prot*, `ProtocolPP::iana_t` *lvl3*, `ProtocolPP::protocol_t` *lvl2*, bool *lvl3_sec*, std::string &*host*, unsigned long *iqlsize*, unsigned long *oqlsize*, unsigned long *seed*, `ProtocolPP::endian_t` *endianess*, std::shared_ptr<`InterfacePP::jmmu`> &*mmu*, std::shared_ptr<`InterfacePP::jlogger`> &*logger*)
- virtual `~jdrive` ()

standard deconstructor

 - `uint64_t write_mem` (`uint8_t *data`, unsigned int *length*)
 - `uint32_t send` (`uint8_t *packet`, unsigned int *size*)
 - `uint32_t send` (std::shared_ptr<`ProtocolPP::jarray`<`uint8_t`>> &*input*)
 - `bool rcv` ()
 - `uint32_t pop` (`uint8_t *output`, unsigned int *size*)
 - `uint32_t pop` (std::shared_ptr<`ProtocolPP::jarray`<`uint8_t`>> &*output*)
 - unsigned int `get_size` ()
 - void `teardown` ()

terminate the driver

7.29.1 Constructor & Destructor Documentation

7.29.1.1 `DriverPP::jdrive::jdrive (bool listen, int port, ProtocolPP::protocol_t prot, ProtocolPP::iana_t lvl3, ProtocolPP::protocol_t lvl2, bool lvl3_sec, std::string & host, unsigned long iqlsize, unsigned long oqlsize, unsigned long seed, ProtocolPP::endian_t endianess, std::shared_ptr<InterfacePP::jmmu> & mmu, std::shared_ptr<InterfacePP::jlogger> & logger)`

`jdrive` uses protocol++ to drive packets into a socket which can then be read by any device. Application layer protocols supported are TLS, SRTP, TCP, and UDP

Parameters

<i>listen</i>	- if set to true, configures to listen on the port specified
<i>port</i>	- port number
<i>prot</i>	- transport or application level protocol (TCP, UDP, TLS, SRTP)
<i>lvl3</i>	- network layer protocol (IPV4, IPV6)
<i>lvl2</i>	- data-link layer protocol (MACSEC, WIFI, WIMAX, LTE)
<i>lvl3_sec</i>	- Enable ESP for network level
<i>host</i>	- string of the host name (or IP Address in standard byte format XX.XX.XX...)
<i>iqlsize</i>	- Size of the send queue
<i>oqlsize</i>	- Size of the receive queue
<i>seed</i>	- seed for the randomizer
<i>endianess</i>	- Endianess of the platform to support
<i>mmu</i>	- tracks dynamic memory
<i>logger</i>	- object for writing

7.29.1.2 virtual `DriverPP::jdrive::~jdrive ()` [inline], [virtual]

standard deconstructor

7.29.2 Member Function Documentation

7.29.2.1 unsigned int `DriverPP::jdrive::get_size ()`

Returns

- size of the next output packet

7.29.2.2 `uint32_t DriverPP::jdrive::pop (uint8_t * output, unsigned int size)`

Returns the next packet in the receive queue

Parameters

<code>output</code>	- payload from decapsulated packet
<code>size</code>	- size of the byte pointer

7.29.2.3 `uint32_t DriverPP::jdrive::pop (std::shared_ptr<ProtocolPP::jarray< uint8_t >> & output)`

Returns the next packet in the receive queue

Parameters

<code>output</code>	- payload from decapsulated packet
---------------------	------------------------------------

7.29.2.4 `bool DriverPP::jdrive::rcv ()`

Returns

True if packet received. Poll this in your program

7.29.2.5 `uint32_t DriverPP::jdrive::send (uint8_t * packet, unsigned int size)`

Issues the packet into the socket

Parameters

<code>packet</code>	- byte array for the input payload
<code>size</code>	- length of the input data

Returns

- status of push command

7.29.2.6 `uint32_t DriverPP::jdrive::send (std::shared_ptr<ProtocolPP::jarray< uint8_t >> & input)`

Issues the packet into the socket

Parameters

<code>input</code>	- byte array for the input payload
--------------------	------------------------------------

Returns

- status of push command

7.29.2.7 `void DriverPP::jdrive::teardown ()`

terminate the driver

7.29.2.8 `uint64_t DriverPP::jdrive::write_mem (uint8_t * data, unsigned int length)`

Write data to memory

Parameters

<i>data</i>	- Data to write to memory
<i>length</i>	- length of data to write

Returns

address - Address data written to

The documentation for this class was generated from the following file:

- drivers/include/jdrive.h

7.30 jdriver Class Reference

```
#include "include/jdriver.h"
```

7.30.1 Detailed Description

7.30.2 Driver for Protocol++

See Also

[jring](#)
[jdata](#)
[jpacket](#)
[jstream](#)
[protocolpp](#)

Protocol++ written by : John Peter Greninger © John Peter Greninger 2015-2019 All Rights Reserved

All copyrights and trademarks are the property of their respective owners

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer
- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution
- Any and all modifications must be returned to John Peter Greninger at GitHub.com for evaluation. Inclusion of modifications in the source code shall be determined solely by John Peter Greninger. Failure to provide modifications shall render this license NULL and VOID and revoke any rights to use of Protocol++
- Commercial use requires a fee-based license obtainable at www.protocolpp.com
- US Copyrights at <https://www.copyright.gov/>
 - TXu002059872 (Version 1.0.0)
 - TXu002066632 (Version 1.2.7)
 - TXu002082674 (Version 1.4.0)
 - TXu002097880 (Version 2.0.0)

The names of its contributors may not be used to endorse or promote products derived from this software without specific prior written permission

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE

The documentation for this class was generated from the following file:

- drivers/include/jdriver.h

7.31 DriverPP::jdriver< Q > Class Template Reference

```
#include <jdriver.h>
```

Public Member Functions

- `jdriver` (int port, std::string &host, unsigned long seed, ProtocolPP::endian_t endianess, std::shared_ptr< InterfacePP::jmmu > &mmu, std::shared_ptr< InterfacePP::jlogger > &logger, std::shared_ptr< InterfacePP::jring< InterfacePP::ringin >> &iring, std::shared_ptr< InterfacePP::jring< InterfacePP::ringout >> &oring, std::shared_ptr< Q > &secass_snd, std::shared_ptr< Q > &secass_rcv)
- virtual `~jdriver()`
standard deconstructor
- `uint64_t write_mem` (uint8_t *data, unsigned int length)
- `void send()`
- `void receive()`
- `uint32_t get_status` (std::string &packet)
- `void teardown` (std::shared_ptr< Q > &secass_snd, std::shared_ptr< Q > &secass_rcv)
- `void teardown()`
terminate the driver

7.31.1 Constructor & Destructor Documentation

```
7.31.1.1 template<typename Q> DriverPP::jdriver< Q >::jdriver( int port, std::string & host, unsigned long seed,
ProtocolPP::endian_t endianess, std::shared_ptr< InterfacePP::jmmu > & mmu, std::shared_ptr<
InterfacePP::jlogger > & logger, std::shared_ptr< InterfacePP::jring< InterfacePP::ringin >> & iring,
std::shared_ptr< InterfacePP::jring< InterfacePP::ringout >> & oring, std::shared_ptr< Q > & secass_snd,
std::shared_ptr< Q > & secass_rcv )
```

jdriver uses protocol++ to drive packets into a ring which can then be read by any device that supports a ring

Parameters

<code>port</code>	- port number
<code>host</code>	- string of the host name

<i>seed</i>	- seed for the randomizer
<i>mmu</i>	- tracks dynamic memory
<i>logger</i>	- object to print output
<i>iring</i>	- Software ring for the input packets
<i>oring</i>	- Software ring for the output packets
<i>secass_snd</i>	- Security association for send
<i>secass_rcv</i>	- Security association for receive
<i>endianess</i>	- Endianess of the platform to support

7.31.1.2 template<typename Q > virtual DriverPP::jdriver< Q >::~jdriver() [inline], [virtual]

standard deconstructor

7.31.2 Member Function Documentation

7.31.2.1 template<typename Q > uint32_t jdriver::get_status (std::string & packet)

retrieve the status word

Returns

- status of the indicated packet

7.31.2.2 template<typename Q > void jdriver::receive ()

Issues the packet into the testbench or system must be overloaded in user's code

See Also

[jpacket](#)

7.31.2.3 template<typename Q > void jdriver::send ()

Issues the packet into the testbench or system must be overloaded in user's code

See Also

[jpacket](#)

7.31.2.4 template<typename Q > void jdriver::teardown (std::shared_ptr< Q > & secass_snd, std::shared_ptr< Q > & secass_rcv)

terminate the driver and return the security associations

Parameters

<i>secass_snd</i>	- Security Association for sending packets
<i>secass_rcv</i>	- Security Association for receiving packets

7.31.2.5 template<typename Q > void jdriver::teardown ()

terminate the driver

7.31.2.6 template<typename Q> uint64_t jdriver::write_mem(uint8_t * *data*, unsigned int *length*)

Write function to overload in your testbench

Parameters

<i>data</i>	- Data to write to memory
<i>length</i>	- length of data to write

Returns

address - Address data written to

The documentation for this class was generated from the following file:

- drivers/include/jdriver.h

7.32 jdsa Class Reference

```
#include "include/jdsa.h"
```

7.32.1 Detailed Description

7.32.2 Digital Signature Algorithm (DSA)

see https://en.wikipedia.org/wiki/Digital_Signature_Algorithm

The Digital Signature Algorithm (DSA) is a Federal Information Processing Standard for digital signatures, based on the mathematical concept of modular exponentiations and the discrete logarithm problem.

In August 1991 the National Institute of Standards and Technology (NIST) proposed DSA for use in their Digital Signature Standard (DSS) and adopted it as FIPS 186 in 1994. Four revisions to the initial specification have been released: FIPS 186-1 in 1996, FIPS 186-2 in 2000, FIPS 186-3 in 2009, and FIPS 186-4 in 2013

DSA: How does it work?

The DSA algorithm works in the framework of public-key cryptosystems and is based on the algebraic properties of the modular exponentiations, together with the discrete logarithm problem (which is considered to be computationally intractable). Messages are signed by the signer's private key and the signatures are verified by the signer's corresponding public key. The digital signature provides message authentication, integrity and non-repudiation

Key generation

Key generation has two phases. The first phase is a choice of algorithm parameters which may be shared between different users of the system, while the second phase computes public and private keys for a single user

Parameter generation

Choose an approved cryptographic hash function H. In the original DSS, H was always SHA-1, but the stronger SHA-2 hash functions are approved for use in the current DSS. The hash output may be truncated to the size of a key pair

Decide on a key length L and N (the cryptographic strength of the L bit long key). The original DSS constrained L to be a multiple of 64 between 512 and 1,024 (inclusive). NIST 800-57 recommends lengths of 2,048 (or 3,072) for keys with security lifetimes extending beyond 2010 (or 2030), using correspondingly longer N. FIPS 186-3 specifies L and N length pairs of (1,024, 160), (2,048, 224), (2,048, 256), and (3,072, 256). N must be less than or equal to the output length of the hash H

Choose an N-bit prime q

Choose an L-bit prime p such that $p - 1$ is a multiple of q

Choose g , a number whose multiplicative order modulo p is q . This means that q is the smallest positive integer such that $gq \equiv 1 \pmod{p}$. This may be done by setting $g = h^{(p-1)/q} \pmod{p}$ for some arbitrary h ($1 < h < p - 1$), and trying again with a different h if the result comes out as 1. Most choices of h will lead to a usable g ; commonly $h = 2$ is used

The algorithm parameters (p, q, g) may be shared between different users of the system

Per-user keys

Given a set of parameters, the second phase computes private and public keys for a single user:

- Choose a secret private key x by some random method, where $0 < x < q$
- Calculate the public key $y = g^x \bmod p$

There exist efficient algorithms for computing the modular exponentiations $h^{\frac{p-1}{q}} \bmod p$ and $g^x \bmod p$, such as exponentiation by squaring

Signing

- Let H be the hashing function and m the message:
- Generate a random per-message value k where $1 < k < q$
- Calculate $r = (g^k \bmod p) \bmod q$
- In the unlikely case that $r = 0$, start again with a different random k
- Calculate $s = k^{-1} (H(m) + xr) \bmod q$
- In the unlikely case that $s = 0$, start again with a different random k
- The signature is (r, s)

The first two steps amount to creating a new per-message key. The modular exponentiation here is the most computationally expensive part of the signing operation, and it may be computed before the message hash is known. The modular inverse

$k^{-1} \bmod q$

is the second most expensive part, and it may also be computed before the message hash is known. It may be computed using the extended Euclidean algorithm or using Fermat's little theorem as

$k^{q-2} \bmod q$

Verifying

- Reject the signature if $0 < r < q$ or $0 < s < q$ is not satisfied
- Calculate $w = s^{-1} \bmod q$
- Calculate $u_1 = H(m) \cdot w \bmod q$
- Calculate $u_2 = r \cdot w \bmod q$
- Calculate $v = (g^{u_1} y^{u_2} \bmod p) \bmod q$
- The signature is valid if and only if $v = r$

For API information:

See Also

[ProtocolPP::jdsa](#)
[ProtocolPP::jprotocol](#)
[ProtocolPP::jprotocolpp](#)
[ProtocolPP::jenum](#)

For Additional Documentation:

See Also

[jdsa](#)
[jprotocol](#)
[jprotocolpp](#)
[jenum](#)

Protocol++ written by : John Peter Greninger © John Peter Greninger 2015-2019 All Rights Reserved

All copyrights and trademarks are the property of their respective owners

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer
- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution
- Any and all modifications must be returned to John Peter Greninger at GitHub.com <https://github.com/jpgreninger/protocolpp> for evaluation. Inclusion of modifications in the source code shall be determined solely by John Peter Greninger. Failure to provide modifications shall render this license NULL and VOID and revoke any rights to use of Protocol++
- Commercial use requires a fee-based license obtainable at www.protocolpp.com
- Academic use requires written and notarized permission from John Peter and Sheila Rocha Greninger
- US Copyrights at <https://www.copyright.gov/>
 - TXU002059872 (Version 1.0.0)
 - TXU002066632 (Version 1.2.7)
 - TXU002082674 (Version 1.4.0)
 - TXU002097880 (Version 2.0.0)

The names of its contributors may not be used to endorse or promote products derived from this software without specific prior written permission

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE

The documentation for this class was generated from the following file:

- [include/jdsa.h](#)

7.33 ProtocolPP::jdsa Class Reference

```
#include <jdsa.h>
```

Public Member Functions

- `jdsa (int bitsize, std::shared_ptr< InterfacePP::jlogger > &logger)`
Diffie-Hellman Constructor for named type (see /ref jenum)
- `jdsa (int bitsize, CryptoPP::Integer p, CryptoPP::Integer q, CryptoPP::Integer g, std::shared_ptr< InterfacePP::jlogger > &logger)`
Diffie-Hellman MODP Constructor.
- `~jdsa ()`
Standard deconstructor.
- template<typename T >
`void set_field (field_t field, T value)`
- template<typename T >
`T get_field (field_t field)`
- `bool gen_keypair ()`
Generate a public key pair.
- `bool sign (std::shared_ptr< jarray< uint8_t >> &data, std::shared_ptr< jarray< uint8_t >> &signature)`
- `bool verify (std::shared_ptr< jarray< uint8_t >> &signature)`

7.33.1 Constructor & Destructor Documentation

7.33.1.1 ProtocolPP::jdsa (int bitsize, std::shared_ptr< InterfacePP::jlogger > & logger)

Diffie-Hellman Constructor for named type (see /ref jenum)

7.33.1.2 ProtocolPP::jdsa (int bitsize, CryptoPP::Integer p, CryptoPP::Integer q, CryptoPP::Integer g, std::shared_ptr< InterfacePP::jlogger > & logger)

Diffie-Hellman MODP Constructor.

7.33.1.3 ProtocolPP::jdsa::~jdsa () [inline]

Standard deconstructor.

7.33.2 Member Function Documentation

7.33.2.1 bool ProtocolPP::jdsa::gen_keypair ()

Generate a public key pair.

7.33.2.2 template<typename T > T ProtocolPP::jdsa::get_field (field_t field)

Returns the version field of the IP security association

Parameters

<code>field</code>	- field to retrieve from the IP security association
--------------------	--

Returns

field of the IP security association

7.33.2.3 template<typename T > void ProtocolPP::jdsa::set_field (field_t field, T value)

Allows the user to update the field Diffie-Hellman object

field type	field name	Example
dh_mode_t	DHMODE	dh_mode_t dhmode = get_field<ProtocolPP::dh_mode_t>(ProtocolPP::field_t::DHMODE)
CryptoPP::Integer	GENERATOR	CryptoPP::Integer generator = get_field<CryptoPP::Integer>(-ProtocolPP::GENERATOR)
CryptoPP::Integer	QUOTIENT	CryptoPP::Integer quotient = get_field<CryptoPP::Integer>(-ProtocolPP::QUOTIENT)
CryptoPP::Integer	PRODUCT	CryptoPP::Integer product = get_field<CryptoPP::Integer>(-ProtocolPP::PRODUCT)
CryptoPP::SecByteBlock	PUBKEY	CryptoPP::SecByteBlock pubkey = get_field<CryptoPP::SecByteBlock>(ProtocolPP::PUBKEY)
CryptoPP::SecByteBlock	PRVKEY	CryptoPP::SecByteBlock prvkey = get_field<CryptoPP::SecByteBlock>(ProtocolPP::PRVKEY)
std::pair<CryptoPP::SecByteBlock, CryptoPP::SecByteBlock>	KEYPAIR	shared_ptr<std::pair<CryptoPP::SecByteBlock, CryptoPP::SecByteBlock>> keypair = get_field<std::pair<CryptoPP::SecByteBlock, CryptoPP::SecByteBlock>>(ProtocolPP::KEYPAIR)

Table 7.1: Diffie-Hellman Set Fields

Parameters

<i>field</i>	- field to update the IP security association
<i>value</i>	- value to update the IP security association

7.33.2.4 bool ProtocolPP::jdsa::sign (std::shared_ptr<jarray< uint8_t >> & *data*, std::shared_ptr<jarray< uint8_t >> & *signature*)

Generate a public key pair

Parameters

<i>data</i>	- data to sign
<i>signature</i>	- Signature generated from private key and data

7.33.2.5 bool ProtocolPP::jdsa::verify (std::shared_ptr<jarray< uint8_t >> & *signature*)

Verify the signature using the received public key

Parameters

<i>signature</i>	- signature to verify
------------------	-----------------------

Returns

Signature verification

The documentation for this class was generated from the following file:

- include/jdsa.h

field type	field name	Example
dh_mode_t	DHMODE	set_field<ProtocolPP::dh_mode_t>(ProtocolPP::field_t::DHMODE, ProtocolPP::MODP)
CryptoPP::Integer	GENERATOR	set_field<CryptoPP::Integer>(-ProtocolPP::GENERATOR, g)
CryptoPP::Integer	QUOTIENT	set_field<CryptoPP::Integer>(-ProtocolPP::QUOTIENT, q)
CryptoPP::Integer	PRODUCT	set_field<CryptoPP::Integer>(-ProtocolPP::PRODUCT, p)
CryptoPP::SecByteBlock	PUBKEY	set_field<CryptoPP::SecByteBlock>(ProtocolPP::PUBKEY, pubkey)
CryptoPP::SecByteBlock	PRVKEY	set_field<CryptoPP::SecByteBlock>(ProtocolPP::PRVKEY, prvkey)
std::pair<CryptoPP::SecByteBlock, CryptoPP::SecByteBlock>	KEYPAIR	set_field<std::pair<CryptoPP::SecByteBlock, CryptoPP::SecByteBlock>>(ProtocolPP::KEYPAIR, keypair)

Table 7.2: Diffie-Hellman Set Fields

7.34 jecdsa Class Reference

```
#include "include/jecdsa.h"
```

7.34.1 Detailed Description

7.34.2 Elliptic Curve Digital Signature Algorithm (ECDSA)

see https://en.wikipedia.org/wiki/Elliptic_Curve_Digital_Signature_Algorithm

In cryptography, the Elliptic Curve Digital Signature Algorithm (ECDSA) offers a variant of the Digital Signature Algorithm (DSA) which uses elliptic curve cryptography

Key and signature-size comparison to DSA

As with elliptic-curve cryptography in general, the bit size of the public key believed to be needed for ECDSA is about twice the size of the security level, in bits. For example, at a security level of 80 bits (meaning an attacker requires a maximum of about 2^{80} operations to find the private key) the size of an ECDSA public key would be 160 bits, whereas the size of a DSA public key is at least 1024 bits. On the other hand, the signature size is the same for both DSA and ECDSA: approximately $4t$ bits, where t is the security level measured in bits, that is, about 320 bits for a security level of 80 bits

Signature generation algorithm

Suppose Alice wants to send a signed message to Bob. Initially, they must agree on the curve parameters (**CURVE**, **G**, **n**). In addition to the field and equation of the curve, we need **G**, a base point of prime order on the curve; **n** is the multiplicative order of the point **G**

The order **n** of the base point **G** **must be prime**. Indeed, we assume that every nonzero element of the ring $\frac{Z}{nZ}$ are invertible, so that $\frac{Z}{nZ}$ must be a field. It implies that **n** must be prime (cf. Bézout's identity)

Alice creates a key pair, consisting of a private key integer d_A , randomly selected in the interval $[1, n - 1]$; and a public key curve point $Q_A = d_A \times G$

We use \times to denote elliptic curve point multiplication by a scalar

Parameter	Description
CURVE	the elliptic curve field and equation used
G	elliptic curve base point, such as a pt (x_0, y_0) on $y^2 = x^3 + 7$, a generator of the elliptic curve with large prime order n
n	integer order of G, means that $n \times G = O$, where O is the identity element

Table 7.3: ECDSA Parameters

For Alice to sign a message m, she follows these steps:

- Calculate $e = \text{HASH}(m)$, where HASH is a cryptographic hash function, such as **SHA-2**
- Let z be the L_n leftmost bits of e , where L_n is the bit length of the group order n
- Select a cryptographically secure random integer k from $[1, n - 1]$
- Calculate the curve point $(x_1, y_1) = k \times G$
- Calculate $r = x_1 \bmod n$. If $r = 0$, go back to step 3
- Calculate $s = k^{-1}(z + rd_A) \bmod n$
- If $s = 0$, go back to step 3
- The signature is the pair (r, s)

When computing s , the string z resulting from $\text{HASH}(m)$ shall be converted to an integer. Note that z can be greater than n but not longer.

As the standard notes, it is not only required for k to be secret, but it is also crucial to select different k for different signatures, otherwise the equation in step 6 can be solved for d_A , the private key: Given two signatures (r, s) and (r, s') , employing the same unknown k for different known messages m and m' , an attacker can calculate z and z' , and since $s - s' = k^{-1}(z - z')$ (all operations in this paragraph are done modulo n) the attacker can find $k = \frac{z - z'}{s - s'}$. Since $s = k^{-1}(z + rd_A)$, the attacker can now calculate the private key $d_A = \frac{sk - z}{r}$.

Another way ECDSA signature may leak private keys is when k is generated by a faulty random number generator. To ensure that k is unique for each message one may bypass random number generation completely and generate deterministic signatures by deriving k from both the message and the private key.

Signature verification algorithm

For Bob to authenticate Alice's signature, he must have a copy of her public-key curve point Q_A

Bob can verify Q_A is a valid curve point as follows:

- Check that Q_A is not equal to the identity element O , and its coordinates are otherwise valid
- Check that Q_A lies on the curve
- Check that $n \times Q_A = O$

After that, Bob follows these steps:

- Verify that r and s are integers in $[1, n - 1]$. If not, the signature is invalid.
- Calculate $e = \text{HASH}(m)$, where HASH is the same function used in the signature generation
- Let z be the L_n leftmost bits of e
- Calculate $w = s^{-1} \bmod n$

- Calculate $u_1 = z \text{wmodn}$ and $u_2 = r \text{wmodn}$
- Calculate the curve point $(x_1, y_1) = u_1 \times G + u_2 \times Q_A$
- If $(x_1, y_1) = O$ then the signature is invalid
- The signature is valid if $r \equiv x_1 \pmod{n}$, invalid otherwise

Note that using Shamir's trick, a sum of two scalar multiplications $u_1 \times G + u_2 \times Q_A$ can be calculated faster than two scalar multiplications done independently

Concerns

There exist two sorts of concerns with ECDSA:

- Political concerns: the trustworthiness of NIST-produced curves being questioned after revelations that the NSA willingly inserts backdoors into software, hardware components and published standards were made; well-known cryptographers have expressed doubts about how the NIST curves were designed, and voluntary tainting has already been proved in the past
- Technical concerns: the difficulty of properly implementing the standard, its slowness, and design flaws which reduce security in insufficiently defensive implementations of the Dual EC DRBG random number generator

For API information:

See Also

[ProtocolPP::jecdsa](#)
[ProtocolPP::jprotocol](#)
[ProtocolPP::jprotocolpp](#)
[ProtocolPP::jenum](#)

For Additional Documentation:

See Also

[jecdsa](#)
[jprotocol](#)
[jprotocolpp](#)
[jenum](#)

Protocol++ written by : John Peter Greninger © John Peter Greninger 2015-2019 All Rights Reserved

All copyrights and trademarks are the property of their respective owners

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer
- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution
- Any and all modifications must be returned to John Peter Greninger at GitHub.com <https://github.com/jpgreninger/protocolpp> for evaluation. Inclusion of modifications in the source code shall be determined solely by John Peter Greninger. Failure to provide modifications shall render this license NULL and VOID and revoke any rights to use of Protocol++
- Commercial use requires a fee-based license obtainable at www.protocolpp.com
- Academic use requires written and notarized permission from John Peter and Sheila Rocha Greninger

- US Copyrights at <https://www.copyright.gov/>
 - TXu002059872 (Version 1.0.0)
 - TXu002066632 (Version 1.2.7)
 - TXu002082674 (Version 1.4.0)
 - TXu002097880 (Version 2.0.0)

The names of its contributors may not be used to endorse or promote products derived from this software without specific prior written permission

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE

The documentation for this class was generated from the following file:

- [include/jecdsa.h](#)

7.35 ProtocolPP::jecdsa Class Reference

```
#include <jecdsa.h>
```

Public Member Functions

- [jecdsa \(ike_hash_t hash, CryptoPP::OID curve, std::shared_ptr< InterfacePP::jlogger > &logger\)](#)
- [~jecdsa \(\)](#)
Standard deconstructor.
- template<typename T >
[void set_field \(field_t field, T value\)](#)
- template<typename T >
[T get_field \(field_t field\)](#)
- [bool gen_keypair \(\)](#)
Generate a public key pair.
- [bool sign \(std::shared_ptr< jarray< uint8_t >> &data, std::shared_ptr< jarray< uint8_t >> &signature\)](#)
- [bool verify \(std::shared_ptr< jarray< uint8_t >> &signature\)](#)

7.35.1 Constructor & Destructor Documentation

7.35.1.1 [ProtocolPP::jecdsa::jecdsa \(ike_hash_t hash, CryptoPP::OID curve, std::shared_ptr< InterfacePP::jlogger > & logger \)](#)

ECDSA Constructor for named type (see /ref jenum)

Parameters

<i>hash</i>	- hash type for curve
<i>curve</i>	- Name of the OID curve
<i>logger</i>	- logging object

7.35.1.2 `ProtocolPP::jecdsa::~jecdsa() [inline]`

Standard deconstructor.

7.35.2 Member Function Documentation

7.35.2.1 `bool ProtocolPP::jecdsa::gen_keypair()`

Generate a public key pair.

7.35.2.2 `template<typename T> T ProtocolPP::jecdsa::get_field(field_t field)`

Returns the version field of the IP security association

field type	field name	Example
<code>dh_mode_t</code>	<code>DHMODE</code>	<code>dh_mode_t dhmode = get_field<ProtocolPP::dh_mode_t>(ProtocolPP::field_t::DHMODE)</code>
<code>CryptoPP::Integer</code>	<code>GENERATOR</code>	<code>CryptoPP::Integer generator = get_field<CryptoPP::Integer>(-ProtocolPP::GENERATOR)</code>
<code>CryptoPP::Integer</code>	<code>QUOTIENT</code>	<code>CryptoPP::Integer quotient = get_field<CryptoPP::Integer>(-ProtocolPP::QUOTIENT)</code>
<code>CryptoPP::Integer</code>	<code>PRODUCT</code>	<code>CryptoPP::Integer product = get_field<CryptoPP::Integer>(-ProtocolPP::PRODUCT)</code>
<code>CryptoPP::SecByteBlock</code>	<code>PUBKEY</code>	<code>CryptoPP::SecByteBlock pubkey = get_field<CryptoPP::SecByteBlock>(ProtocolPP::PUBKEY)</code>
<code>CryptoPP::SecByteBlock</code>	<code>PRVKEY</code>	<code>CryptoPP::SecByteBlock prvkey = get_field<CryptoPP::SecByteBlock>(ProtocolPP::PRVKEY)</code>
<code>std::pair<CryptoPP::SecByteBlock, CryptoPP::SecByteBlock></code>	<code>KEYPAIR</code>	<code>shared_ptr<std::pair<CryptoPP::SecByteBlock, CryptoPP::SecByteBlock>> keypair = get_field<std::pair<CryptoPP::SecByteBlock, CryptoPP::SecByteBlock>>(ProtocolPP::KEYPAIR)</code>

Table 7.4: Diffie-Hellman Set Fields

Parameters

<code>field</code>	- field to retrieve from the IP security association
--------------------	--

Returns

field of the IP security association

7.35.2.3 `template<typename T> void ProtocolPP::jecdsa::set_field(field_t field, T value)`

Allows the user to update the field Diffie-Hellman object

field type	field name	Example
dh_mode_t	DHMODE	set_field<ProtocolPP::dh_mode_t>(ProtocolPP::field_t::DHMODE, ProtocolPP::MODP)
CryptoPP::Integer	GENERATOR	set_field<CryptoPP::Integer>(-ProtocolPP::GENERATOR, g)
CryptoPP::Integer	QUOTIENT	set_field<CryptoPP::Integer>(-ProtocolPP::QUOTIENT, q)
CryptoPP::Integer	PRODUCT	set_field<CryptoPP::Integer>(-ProtocolPP::PRODUCT, p)
CryptoPP::SecByteBlock	PUBKEY	set_field<CryptoPP::SecByteBlock>(ProtocolPP::PUBKEY, pubkey)
CryptoPP::SecByteBlock	PRVKEY	set_field<CryptoPP::SecByteBlock>(ProtocolPP::PRVKEY, prvkey)
std::pair<CryptoPP::SecByteBlock, CryptoPP::SecByteBlock>	KEYPAIR	set_field<std::pair<CryptoPP::SecByteBlock, CryptoPP::SecByteBlock>>(ProtocolPP::KEYPAIR, keypair)

Table 7.5: Diffie-Hellman Set Fields

Parameters

<i>field</i>	- field to update the IP security association
<i>value</i>	- value to update the IP security association

7.35.2.4 **bool ProtocolPP::jecdsa::sign (std::shared_ptr<jarray< uint8_t >> & *data*, std::shared_ptr<jarray< uint8_t >> & *signature*)**

Generate a public key pair

Parameters

<i>data</i>	- data to sign
<i>signature</i>	- Signature generated from private key and data

7.35.2.5 **bool ProtocolPP::jecdsa::verify (std::shared_ptr<jarray< uint8_t >> & *signature*)**

Verify the signature using the received public key

Parameters

<i>signature</i>	- signature to verify
------------------	-----------------------

Returns

Signature verification

The documentation for this class was generated from the following file:

- include/jecdsa.h

7.36 jenum Class Reference

```
#include "include/jenum.h"
```

7.36.1 Detailed Description

Protocol++ written by : John Peter Greninger © John Peter Greninger 2015-2018 All Rights Reserved

All copyrights and trademarks are the property of their respective owners

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer
- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution
- Any and all modifications must be returned to John Peter Greninger at GitHub.com <https://github.com/jpgreninger/protocolpp> for evaluation. Inclusion of modifications in the source code shall be determined solely by John Peter Greninger. Failure to provide modifications shall render this license NULL and VOID and revoke any rights to use of Protocol++
- Commercial use requires a fee-based license obtainable at www.protocolpp.com
- Academic use requires written and notarized permission from John Peter and Sheila Rocha Greninger
- US Copyrights at [https://www.copyright.gov/](http://www.copyright.gov/)
 - TXu002059872 (Version 1.0.0)
 - TXu002066632 (Version 1.2.7)
 - TXu002082674 (Version 1.4.0)
 - TXu002097880 (Version 2.0.0)

The names of its contributors may not be used to endorse or promote products derived from this software without specific prior written permission

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE

The documentation for this class was generated from the following file:

- [include/jenum.h](#)

7.37 jexec Class Reference

```
#include "include/jexec.h"
```

7.37.1 Detailed Description

7.37.2 Execution unit for use in testbench for Protocol++

Protocol++ written by : John Peter Greninger © John Peter Greninger 2015-2019 All Rights Reserved

All copyrights and trademarks are the property of their respective owners

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer
- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution
- Any and all modifications must be returned to John Peter Greninger at GitHub.com <https://github.com/jpgreninger/protocolpp> for evaluation. Inclusion of modifications in the source code shall be determined solely by John Peter Greninger. Failure to provide modifications shall render this license NULL and VOID and revoke any rights to use of Protocol++
- Commercial use requires a fee-based license obtainable at www.protocolpp.com
- Academic use requires written and notarized permission from John Peter and Sheila Rocha Greninger
- US Copyrights at <https://www.copyright.gov/>
 - TXu002059872 (Version 1.0.0)
 - TXu002066632 (Version 1.2.7)
 - TXu002082674 (Version 1.4.0)
 - TXu002097880 (Version 2.0.0)

The names of its contributors may not be used to endorse or promote products derived from this software without specific prior written permission

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE

The documentation for this class was generated from the following file:

- jresponder/include/jexec.h

7.38 InterfacePP::jexec Class Reference

```
#include <jexec.h>
```

Public Member Functions

- **jexec** (unsigned long seed, std::shared_ptr< [jlogger](#) > &logger, std::string read_latency=std::string("80..120"), std::string compute_latency=std::string("2000..5000"), std::string write_latency=std::string("60..100"))
 - virtual ~**jexec** ()
standard deconstructor
 - void **set_readlat** (const char *latency)
 - void **set_computlat** (const char *latency)
 - void **set_writelat** (const char *latency)
 - std::string **get_readlat** ()
 - std::string **get_complat** ()
 - std::string **get_writelat** ()
 - bool **get_busy** ()
 - uint32_t **get_status** ()
 - uint32_t **get_outlen** ()
 - void **exec** (std::shared_ptr< [ProtocolPP](#)::[jarray](#)< uint8_t >> &input, uint64_t inaddr, uint64_t outaddr, unsigned int outlen, std::map< std::string, std::shared_ptr< [ProtocolPP](#)::[jstream](#) >> &flows, bool protect=false, std::string stream=std::string(""))

7.38.1 Constructor & Destructor Documentation

```
7.38.1.1 InterfacePP::jexec::jexec( unsigned long seed, std::shared_ptr<jlogger> & logger, std::string read_latency = std::string("80..120"), std::string compute_latency = std::string("2000..5000"), std::string write_latency = std::string("60..100") )
```

Execution units for use in a responder equivalent for a superscale design with multiple threads

Parameters

<i>seed</i>	- seed for random number generator for randomizing latencies
<i>logger</i>	- object for logging output
<i>read_latency</i>	- latency of reads, can be a range (i.e., "4..30")
<i>compute_latency</i>	- latency of computation, can be a range (i.e., "100..200")
<i>write_latency</i>	- latency of writes, can be a range (i.e., "50..80")

7.38.1.2 virtual InterfacePP::jexec::~jexec() [inline], [virtual]

standard deconstructor

7.38.2 Member Function Documentation

```
7.38.2.1 void InterfacePP::jexec::exec ( std::shared_ptr< ProtocolPP::jarray< uint8_t >> & input, uint64_t inaddr,  
    uint64_t outaddr, unsigned int outlen, std::map< std::string, std::shared_ptr< ProtocolPP::jstream >> & flows,  
    bool protect = false, std::string stream = std::string( "" ) )
```

Executes the packet with the latencies requested

Parameters

<i>input</i>	- packet to process
<i>inaddr</i>	- address to read input from
<i>outaddr</i>	- address to write output to
<i>outlen</i>	- Length of the desired output
<i>flows</i>	- map to all available flows
<i>protect</i>	- apply protection to the packet
<i>stream</i>	- stream for encapsulation

7.38.2.2 bool InterfacePP::jexec::get_busy()

Checks to see if the execution unit is busy

Returns

- state of execution unit

7.38.2.3 std::string InterfacePP::jexec::get_complat()

Retrieves the compute latency as a string

Returns

- compute latency

7.38.2.4 uint32_t InterfacePP::jexec::get_outlen()

Retrieve the output packet length

Returns

- length of the output packet

7.38.2.5 std::string InterfacePP::jexec::get_readlat()

Retrieves the read latency as a string

Returns

- read latency

7.38.2.6 uint32_t InterfacePP::jexec::get_status()

Checks to see if the execution unit is busy

Returns

- status of processing

7.38.2.7 std::string InterfacePP::jexec::get_writelat()

Retrieves the write latency as a string

Returns

- write latency

7.38.2.8 void InterfacePP::jexec::set_complat(const char * latency)

Sets the compute latency in the execution unit

Parameters

<i>latency</i>	- latency (i.e., "5..20")
----------------	---------------------------

7.38.2.9 void InterfacePP::jexec::set_readlat (const char * *latency*)

Sets the read latency in the execution unit

Parameters

<i>latency</i>	- latency (i.e., "5..20")
----------------	---------------------------

7.38.2.10 void InterfacePP::jexec::set_writelat (const char * *latency*)

Sets the write latency in the execution unit

Parameters

<i>latency</i>	- latency (i.e., "5..20")
----------------	---------------------------

The documentation for this class was generated from the following file:

- jresponder/include/jexec.h

7.39 jicmp Class Reference

```
#include "include/jicmp.h"
```

7.39.1 Detailed Description

7.39.2 Internet Control Message Protocol (ICMP)

See https://en.wikipedia.org/wiki/Internet_Control_Message_Protocol

The Internet Control Message Protocol (ICMP) is one of the main protocols of the internet protocol suite. It is used by network devices, like routers, to send error messages indicating, for example, that a requested service is not available or that a host or router could not be reached. ICMP can also be used to relay query messages.[1] It is assigned protocol number 1.[2] ICMP[3] differs from transport protocols such as TCP and UDP in that it is not typically used to exchange data between systems, nor is it regularly employed by end-user network applications (with the exception of some diagnostic tools like ping and traceroute)

ICMP_STRUCT ICMP datagram structure

Header

The ICMP header starts after the IPv4 header and is identified by IP protocol number '1'. All ICMP packets have an 8-byte header and variable-sized data section. The first 4 bytes of the header have fixed format, while the last 4 bytes depend on the type/code of that ICMP packet

ICMP Header Format																																			
Offsets		Octet		0								1								2								3							
Octet	Bit	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31		
0	0	Type								Code								Checksum																	
4	32	Rest of Header																																	

Figure 7.1: ICMP Version 4 Header [1]

Type

ICMP type, see Control messages

Code

ICMP subtype, see Control messages

ICMP Version 4 Message Types

Summary of ICMP Version 4 Message Types	
Message Value	Message Name
0	Echo Reply
3	Destination Unreachable
4	Source Quench
5	Redirect
8	Echo
11	Time Exceeded
12	Parameter Problem
13	Timestamp
14	Timestamp Reply
15	Information Request
16	Information Reply

Table 7.6: Summary of Message Types

Checksum

Error checking data, calculated from the ICMP header and data, with value 0 substituted for this field. The Internet Checksum is used, specified in RFC 1071

Rest of Header

Four-bytes field, contents vary based on the ICMP type and code

Data

ICMP error messages contain a data section that includes the entire IPv4 header, plus the first eight bytes of data from the IPv4 packet that caused the error message. The ICMP packet is then encapsulated in a new IPv4 packet.[1] This data is used by the host to match the message to the appropriate process. If a higher level protocol uses port numbers, they are assumed to be in the first 64 data bits of the original datagram's data

The variable size of the ICMP packet data section has been exploited. In the "Ping of death", large or fragmented ping packets are used for denial-of-service attacks. ICMP can also be used to create covert channels for communication. These channels are known as ICMP tunnels

Control Messages

Type	Code	Status	Description
0 – Echo Reply ^{[3]:¹⁴}	0		Echo reply (used to ping)
1 and 2		unassigned	<i>Reserved</i>
	0		Destination network unreachable
	1		Destination host unreachable
	2		Destination protocol unreachable
	3		Destination port unreachable
	4		Fragmentation required, and DF flag set
	5		Source route failed
	6		Destination network unknown
	7		Destination host unknown
3 – Destination Unreachable ^{[3]:⁴}	8		Source host isolated
	9		Network administratively prohibited
	10		Host administratively prohibited
	11		Network unreachable for ToS
	12		Host unreachable for ToS
	13		Communication administratively prohibited
	14		Host Precedence Violation
	15		Precedence cutoff in effect
4 – Source Quench	0	deprecated	Source quench (congestion control)
	0		Redirect Datagram for the Network
5 – Redirect Message	1		Redirect Datagram for the Host
	2		Redirect Datagram for the ToS & network
	3		Redirect Datagram for the ToS & host
6		deprecated	Alternate Host Address
7		unassigned	<i>Reserved</i>
8 – Echo Request	0		Echo request (used to ping)
9 – Router Advertisement	0		Router Advertisement

10 – Router Solicitation	0		Router discovery/selection/solicitation
11 – Time Exceeded ^{[3][6]}	0		TTL expired in transit
	1		Fragment reassembly time exceeded
	0		Pointer indicates the error
12 – Parameter Problem: Bad IP header	1		Missing a required option
	2		Bad length
13 – Timestamp	0		Timestamp
14 – Timestamp Reply	0		Timestamp reply
15 – Information Request	0	deprecated	Information Request
16 – Information Reply	0	deprecated	Information Reply
17 – Address Mask Request	0	deprecated	Address Mask Request
18 – Address Mask Reply	0	deprecated	Address Mask Reply
19		reserved	Reserved for security
20 through 29		reserved	Reserved for robustness experiment
30 – Traceroute	0	deprecated	Information Request
31		deprecated	Datagram Conversion Error
32		deprecated	Mobile Host Redirect
33		deprecated	Where-Are-You (originally meant for IPv6)
34		deprecated	Here-I-Am (originally meant for IPv6)
35		deprecated	Mobile Registration Request
36		deprecated	Mobile Registration Reply
37		deprecated	Domain Name Request
38		deprecated	Domain Name Reply
39		deprecated	SKIP Algorithm Discovery Protocol, Simple Key-Management for Internet Protocol
40			Photuris, Security failures
41		experimental	ICMP for experimental mobility protocols such as Seamoby [RFC4065]
42 through 252		unassigned	Reserved
253		experimental	RFC3692-style Experiment 1 (RFC 4727)
254		experimental	RFC3692-style Experiment 2 (RFC 4727)
255		reserved	Reserved

Figure 7.2: ICMP Control Message Values [2]

7.39.2.1 Internet Control Message Protocol version 6

See https://en.wikipedia.org/wiki/Internet_Control_Message_Protocol_version_6

Internet Control Message Protocol version 6 (ICMPv6) is the implementation of the Internet Control Message Protocol (ICMP) for Internet Protocol version 6 (IPv6). ICMPv6 is defined in RFC 4443.[1] ICMPv6 is an integral part of IPv6 and performs error reporting and diagnostic functions (e.g., ping), and has a framework for extensions to implement future changes

Several extensions have been published, defining new ICMPv6 message types as well as new options for existing ICMPv6 message types. Neighbor Discovery Protocol (NDP) is a node discovery protocol in IPv6 which replaces and enhances functions of ARP.[2] Secure Neighbor Discovery (SEND) is an extension of NDP with extra security. Multicast Router Discovery (MRD) allows discovery of multicast routers

Technical details

ICMPv6 messages may be classified into two categories: error messages and information messages. ICMPv6 messages are transported by IPv6 packets in which the IPv6 Next Header value for ICMPv6 is set to 58

Packet format

The ICMPv6 packet consists of a header and the protocol payload. The header contains only three fields:

- type (8 bits) - Type specifies the type of the message. Values in the range from 0 to 127 (high-order bit is 0) indicate an error message, while values in the range from 128 to 255 (high-order bit is 1) indicate an information message
- code (8 bits) - The code field value depends on the message type and provides an additional level of message granularity
- checksum (16 bits) - The checksum field provides a minimal level of integrity verification for the ICMP message

ICMPv6 packet			
Bit offset	0–7	8–15	16–31
0	Type	Code	Checksum
32	Message body		

Figure 7.3: ICMP Version 6 Header [3]

Message checksum

ICMPv6 provides a minimal level of message integrity verification by the inclusion of a 16-bit checksum in its header. The checksum is calculated starting with a pseudo-header of IPv6 header fields according to the IPv6 standard,[3] which consists of the source and destination addresses, the packet length and the next header field, the latter of which is set to the value 58. Following this pseudo header, the checksum is continued with the ICMPv6 message in which the checksum is initially set to zero. The checksum computation is performed according to Internet protocol standards using 16-bit ones' complement summation, followed by complementing the checksum itself and inserting it into the checksum field.[4] Note that this differs from the way it is calculated for IPv4 in ICMP, but is similar to the calculation done in TCP

ICMPv6 pseudo-header				
Bit offset	0 - 7	8–15	16–23	24–31
0				
32				Source address
64				
96				
128				
160				Destination address
192				
224				
256			ICMPv6 length	
288		Zeros		Next header

Figure 7.4: ICMP Version 6 Psuedo Header for Checksum Calculation [4]

Message processing

When an ICMPv6 node receives a packet, it must undertake actions that depend on the type of message. The ICMPv6 protocol must limit the number of error messages sent to the same destination to avoid network overloading. For example, if a node continues to forward erroneous packets, ICMP will signal the error to the first packet and then do so periodically, with a fixed minimum period or with a fixed network maximum load. An ICMP error message must never be sent in response to another ICMP error message

1. https://en.wikipedia.org/wiki/Internet_Control_Message_Protocol
2. https://en.wikipedia.org/wiki/Internet_Control_Message_Protocol
3. https://en.wikipedia.org/wiki/Internet_Control_Message_Protocol_version_6
4. https://en.wikipedia.org/wiki/Internet_Control_Message_Protocol_version_6

For API Documentation:

See Also

[ProtocolPP::jprotocol](#)
[ProtocolPP::jenum](#)
[ProtocolPP::jarray](#)

For Additional Documentation:

See Also

[jprotocol](#)
[jenum](#)
[jarray](#)

Protocol++ modified by : John Peter Greninger © John Peter Greninger 2015-2019 All Rights Reserved

All copyrights and trademarks are the property of their respective owners

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer
- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution
- Any and all modifications must be returned to John Peter Greninger at GitHub.com <https://github.com/jpgreninger/protocolpp> for evaluation. Inclusion of modifications in the source code shall be determined solely by John Peter Greninger. Failure to provide modifications shall render this license NULL and VOID and revoke any rights to use of Protocol++
- Commercial use requires a fee-based license obtainable at www.protocolpp.com
- Academic use requires written and notarized permission from John Peter and Sheila Rocha Greninger
- US Copyrights at [https://www.copyright.gov/](http://www.copyright.gov/)
 - TXu002059872 (Version 1.0.0)
 - TXu002066632 (Version 1.2.7)
 - TXu002082674 (Version 1.4.0)
 - TXu002097880 (Version 2.0.0)

The names of its contributors may not be used to endorse or promote products derived from this software without specific prior written permission

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE

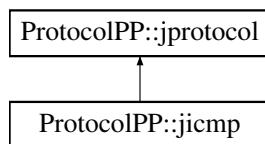
The documentation for this class was generated from the following file:

- `include/jicmp.h`

7.40 ProtocolPP::jicmp Class Reference

```
#include <jicmp.h>
```

Inheritance diagram for ProtocolPP::jicmp:



Public Member Functions

- `jicmp (std::shared_ptr< jicmpsa > &security)`
- virtual `~jicmp ()`
Standard deconstructor.
- void `encap_packet (std::shared_ptr< jarray< uint8_t >> &input, std::shared_ptr< jarray< uint8_t >> &output)`
- void `decap_packet (std::shared_ptr< jarray< uint8_t >> &input, std::shared_ptr< jarray< uint8_t >> &output)`
- void `set_hdr (jarray< uint8_t > &hdr)`
- void `set_field (field_t field, uint64_t value)`
- `jarray< uint8_t > get_hdr ()`
- uint64_t `get_field (field_t field, jarray< uint8_t > &header)`
- void `to_xml (tinyxml2::XMLPrinter &myxml, direction_t direction)`
- uint8_t `convmsg (icmppmsg_t msg)`
- uint8_t `convcode (icmpcode_t code)`

Additional Inherited Members

7.40.1 Constructor & Destructor Documentation

7.40.1.1 ProtocolPP::jicmp::jicmp (std::shared_ptr< [jicmpsa](#) > & security)

constructor for ICMP and ICMPv6

Parameters

<i>security</i>	- Parameters necessary to setup ICMP flow such as DIRECTION, VERSION, MSG, CODE, SRC, DST
-----------------	---

7.40.1.2 `virtual ProtocolPP::jicmp::~jicmp() [inline], [virtual]`

Standard deconstructor.

7.40.2 Member Function Documentation

7.40.2.1 `uint8_t ProtocolPP::jicmp::convcode (icmpcode_t code)`

Converts ICMP code to unsigned int

Parameters

<i>code</i>	- ICMP code to convert
-------------	------------------------

Returns

- Unsigned int for ICMP code

7.40.2.2 `uint8_t ProtocolPP::jicmp::convmsg (icmpmsg_t msg)`

Converts ICMP message code to unsigned int

Parameters

<i>msg</i>	- ICMP message code to convert
------------	--------------------------------

Returns

- Unsigned int for ICMP message

7.40.2.3 `void ProtocolPP::jicmp::decap_packet (std::shared_ptr<jarray< uint8_t >> & input, std::shared_ptr<jarray< uint8_t >> & output) [virtual]`

This function is for use with the constructor without a file handle. Decap will produce a payload from the packet passed in.

Parameters

<i>input</i>	- IP encapsulated packet to decapsulate
<i>output</i>	- decapsulated payload

Implements [ProtocolPP::jprotocol](#).

7.40.2.4 `void ProtocolPP::jicmp::encap_packet (std::shared_ptr<jarray< uint8_t >> & input, std::shared_ptr<jarray< uint8_t >> & output) [virtual]`

This function is for use with the constructor without a file handle. Encap will produce a packet with the payload passed in

Parameters

<i>input</i>	- payload to encapsulate with IP
<i>output</i>	- IP encapsulated packet

Implements [ProtocolPP::jprotocol](#).

7.40.2.5 uint64_t ProtocolPP::jicmp::get_field (field_t *field*, jarray<uint8_t> & *header*) [virtual]

Returns the version field of the IP header

Parameters

<i>field</i>	- field to retrieve from the IP header
<i>header</i>	- ICMP header to retrieve field from

Returns

version field of the IP header

Implements [ProtocolPP::jprotocol](#).

7.40.2.6 jarray<uint8_t> ProtocolPP::jicmp::get_hdr () [virtual]

Allows the user to retrieve the header

Returns

full ICMP header

Implements [ProtocolPP::jprotocol](#).

7.40.2.7 void ProtocolPP::jicmp::set_field (field_t *field*, uint64_t *value*) [virtual]

Allows the user to update the field of the IP header

Parameters

<i>field</i>	- field to update the IP header with
<i>value</i>	- value to update the IP header with

Implements [ProtocolPP::jprotocol](#).

7.40.2.8 void ProtocolPP::jicmp::set_hdr (jarray<uint8_t> & *hdr*) [virtual]

Allows the user to update the header

Parameters

<i>hdr</i>	- new ICMP header
------------	-------------------

Implements [ProtocolPP::jprotocol](#).

7.40.2.9 void ProtocolPP::jicmp::to_xml (tinyxml2::XMLPrinter & *myxml*, direction_t *direction*) [virtual]

Prints the protocol as an XML object

Parameters

<i>myxml</i>	- XMLPrinter object to print with
<i>direction</i>	- randomization

Implements [ProtocolPP::jprotocol](#).

The documentation for this class was generated from the following file:

- [include/jicmp.h](#)

7.41 jicmpsa Class Reference

```
#include "include/jicmpsa.h"
```

7.41.1 Detailed Description

7.41.2 Internet Control Message Protocol Security Association (ICMPSA)

See https://en.wikipedia.org/wiki/Internet_Control_Message_Protocol

The Internet Control Message Protocol (ICMP) is one of the main protocols of the internet protocol suite. It is used by network devices, like routers, to send error messages indicating, for example, that a requested service is not available or that a host or router could not be reached. ICMP can also be used to relay query messages.[1] It is assigned protocol number 1.[2] ICMP[3] differs from transport protocols such as TCP and UDP in that it is not typically used to exchange data between systems, nor is it regularly employed by end-user network applications (with the exception of some diagnostic tools like ping and traceroute)

ICMP_STRUCT ICMP datagram structure

Header

The ICMP header starts after the IPv4 header and is identified by IP protocol number '1'. All ICMP packets have an 8-byte header and variable-sized data section. The first 4 bytes of the header have fixed format, while the last 4 bytes depend on the type/code of that ICMP packet

ICMP Header Format																																	
Offsets	Octet	0							1							2							3										
Octet	Bit	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0	0	Type							Code							Checksum																	
4	32								Rest of Header																								

Figure 7.5: ICMP Version 4 Header [1]

Type

ICMP type, see Control messages

Code

ICMP subtype, see Control messages

Checksum

Error checking data, calculated from the ICMP header and data, with value 0 substituted for this field. The Internet Checksum is used, specified in RFC 1071

Rest of Header

Four-bytes field, contents vary based on the ICMP type and code

Data

ICMP error messages contain a data section that includes the entire IPv4 header, plus the first eight bytes of data from the IPv4 packet that caused the error message. The ICMP packet is then encapsulated in a new IPv4 packet.^[1] This data is used by the host to match the message to the appropriate process. If a higher level protocol uses port numbers, they are assumed to be in the first 64 data bits of the original datagram's data

The variable size of the ICMP packet data section has been exploited. In the "Ping of death", large or fragmented ping packets are used for denial-of-service attacks. ICMP can also be used to create covert channels for communication. These channels are known as ICMP tunnels

Control Messages

Type	Code	Status	Description
0 – Echo Reply ^{[3]:14}	0		Echo reply (used to ping)
1 and 2		unassigned	<i>Reserved</i>
	0		Destination network unreachable
	1		Destination host unreachable
	2		Destination protocol unreachable
	3		Destination port unreachable
	4		Fragmentation required, and DF flag set
	5		Source route failed
	6		Destination network unknown
	7		Destination host unknown
3 – Destination Unreachable ^{[3]:4}	8		Source host isolated
	9		Network administratively prohibited
	10		Host administratively prohibited
	11		Network unreachable for ToS
	12		Host unreachable for ToS
	13		Communication administratively prohibited
	14		Host Precedence Violation
	15		Precedence cutoff in effect
4 – Source Quench	0	deprecated	Source quench (congestion control)
	0		Redirect Datagram for the Network
5 – Redirect Message	1		Redirect Datagram for the Host
	2		Redirect Datagram for the ToS & network
	3		Redirect Datagram for the ToS & host
6		deprecated	Alternate Host Address
7		unassigned	<i>Reserved</i>
8 – Echo Request	0		Echo request (used to ping)
9 – Router Advertisement	0		Router Advertisement

10 – Router Solicitation	0		Router discovery/selection/solicitation
11 – Time Exceeded ^{[3][6]}	0		TTL expired in transit
	1		Fragment reassembly time exceeded
	0		Pointer indicates the error
12 – Parameter Problem: Bad IP header	1		Missing a required option
	2		Bad length
13 – Timestamp	0		Timestamp
14 – Timestamp Reply	0		Timestamp reply
15 – Information Request	0	deprecated	Information Request
16 – Information Reply	0	deprecated	Information Reply
17 – Address Mask Request	0	deprecated	Address Mask Request
18 – Address Mask Reply	0	deprecated	Address Mask Reply
19		reserved	<i>Reserved for security</i>
20 through 29		reserved	<i>Reserved for robustness experiment</i>
30 – Traceroute	0	deprecated	Information Request
31		deprecated	Datagram Conversion Error
32		deprecated	Mobile Host Redirect
33		deprecated	Where-Are-You (originally meant for IPv6)
34		deprecated	Here-I-Am (originally meant for IPv6)
35		deprecated	Mobile Registration Request
36		deprecated	Mobile Registration Reply
37		deprecated	Domain Name Request
38		deprecated	Domain Name Reply
39		deprecated	SKIP Algorithm Discovery Protocol, Simple Key-Management for Internet Protocol
40			Photuris , Security failures
41		experimental	ICMP for experimental mobility protocols such as Seamoby [RFC4065]
42 through 252		unassigned	<i>Reserved</i>
253		experimental	RFC3692-style Experiment 1 (RFC 4727)
254		experimental	RFC3692-style Experiment 2 (RFC 4727)
255		reserved	Reserved

Figure 7.6: ICMP Control Message Values [2]

7.41.2.1 Internet Control Message Protocol version 6

See https://en.wikipedia.org/wiki/Internet_Control_Message_Protocol_version-6

Internet Control Message Protocol version 6 (ICMPv6) is the implementation of the Internet Control Message Protocol (ICMP) for Internet Protocol version 6 (IPv6). ICMPv6 is defined in RFC 4443.[1] ICMPv6 is an integral part of IPv6 and performs error reporting and diagnostic functions (e.g., ping), and has a framework for extensions to implement future changes

Several extensions have been published, defining new ICMPv6 message types as well as new options for existing ICMPv6 message types. Neighbor Discovery Protocol (NDP) is a node discovery protocol in IPv6 which replaces and enhances functions of ARP.[2] Secure Neighbor Discovery (SEND) is an extension of NDP with extra security. Multicast Router Discovery (MRD) allows discovery of multicast routers

For API Documentation:

See Also

[ProtocolPP::jprotocol](#)
[ProtocolPP::jenum](#)
[ProtocolPP::jarray](#)

For Additional Documentation:**See Also**

[jprotocol](#)
[jenum](#)
[jarray](#)

Protocol++ modified by : John Peter Greninger © John Peter Greninger 2015-2019 All Rights Reserved

All copyrights and trademarks are the property of their respective owners

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer
- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution
- Any and all modifications must be returned to John Peter Greninger at GitHub.com <https://github.com/jpgreninger/protocolpp> for evaluation. Inclusion of modifications in the source code shall be determined solely by John Peter Greninger. Failure to provide modifications shall render this license NULL and VOID and revoke any rights to use of Protocol++
- Commercial use requires a fee-based license obtainable at www.protocolpp.com
- Academic use requires written and notarized permission from John Peter and Sheila Rocha Greninger
- US Copyrights at [https://www.copyright.gov/](http://www.copyright.gov/)
 - TXu002059872 (Version 1.0.0)
 - TXu002066632 (Version 1.2.7)
 - TXu002082674 (Version 1.4.0)
 - TXu002097880 (Version 2.0.0)

The names of its contributors may not be used to endorse or promote products derived from this software without specific prior written permission

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE

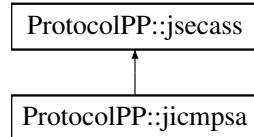
The documentation for this class was generated from the following file:

- [include/jicmpsa.h](#)

7.42 ProtocolPP::jicmpsa Class Reference

```
#include <jicmpsa.h>
```

Inheritance diagram for ProtocolPP::jicmpsa:



Public Member Functions

- [jicmpsa \(\)](#)
 - [jicmpsa \(direction_t dir, iana_t ver, icmpmsg_t msg, icmpcode_t code, uint8_t dsecn, uint8_t ttl, uint8_t flags, uint16_t fragoff, uint16_t id, uint32_t label, jarray< uint8_t > src, jarray< uint8_t > dst\)](#)
 - [jicmpsa \(jicmpsa &security\)](#)
 - [jicmpsa \(std::shared_ptr< jicmpsa > &security\)](#)
 - virtual [~jicmpsa \(\)](#)
- Standard deconstructor.*
- template<typename T>
void [set_field \(field_t field, T fieldval\)](#)
 - template<typename T>
T [get_field \(field_t field\)](#)
 - void [to_xml \(tinyxml2::XMLPrinter &myxml, direction_t direction\)](#)

7.42.1 Constructor & Destructor Documentation

7.42.1.1 ProtocolPP::jicmpsa::jicmpsa ()

Standard Constructor with defaults

field	Default Value
dir	ProtocolPP::DECAP
ver	ProtocolPP::iana_t::ICMP
msg	ProtocolPP::icmpmsg_t::TIMEEXCEED
code	ProtocolPP::icmpcode_t::TTL_EXPIRE
dsecn	0
ttl	0xFF
flags	0
fragoff	0
id	0xFFFF
label	0xFFFFFFFF
src	0xFFFFFFFF
dst	0xFFFFFFFF

Table 7.7: ICMP SA Defaults

Generate the security association for ENCAP and new source and destination addresses do the following:

```

*   jicmpsa snd;
*
*   snd.set_field<direction_t>(field_t::DIRECTION,
*                               direction_t::ENCAP);
*   snd.set_field<jarray<uint8_t>>(field_t::SOURCE,
*                                   jarray<uint8_t>("00112233"));
*   snd.set_field<jarray<uint8_t>>(field_t::DESTINATION,
*                                   jarray<uint8_t>("33221100"));
*

```

7.42.1.2 ProtocolPP::jicmpsa::jicmpsa (*direction_t dir*, *iana_t ver*, *icmppmsg_t msg*, *icmpcode_t code*, *uint8_t dsecn*,
uint8_t ttl, *uint8_t flags*, *uint16_t fragoff*, *uint16_t id*, *uint32_t label*, *jarray<uint8_t> src*, *jarray<uint8_t> dst*)

ICMP security association

Parameters

<i>dir</i>	- Direction of data flow (usually opposite the IP/IPsec flow)
<i>ver</i>	- Version of ICMP (ICMP or ICMPv6)
<i>msg</i>	- Message to be sent to SENDER
<i>code</i>	- Code to be sent to SENDER
<i>dsecn</i>	- DS/ECN from sender
<i>ttl</i>	- Time-to-Live/HOP Limit
<i>flags</i>	- IPv4 flags from sender
<i>fragoff</i>	- IPv4 fragmentation offset from sender
<i>id</i>	- IPv4 identification field from sender
<i>label</i>	- IPv6 label from the sender
<i>src</i>	- Address of RECEIVER
<i>dst</i>	- Address of SENDER

7.42.1.3 ProtocolPP::jicmpsa::jicmpsa (*jicmpsa & security*)

constructor for ICMP and ICMPv6

Parameters

<i>security</i>	- Parameters necessary to setup ICMP flow such as DIRECTION, VERSION, MSG, CODE, SRC, DST
-----------------	---

7.42.1.4 ProtocolPP::jicmpsa::jicmpsa (std::shared_ptr<*jicmpsa* > & *security*)

constructor for ICMP and ICMPv6

Parameters

<i>security</i>	- Parameters necessary to setup ICMP flow such as DIRECTION, VERSION, MSG, CODE, SRC, DST
-----------------	---

7.42.1.5 virtual ProtocolPP::jicmpsa::~jicmpsa () [inline], [virtual]

Standard deconstructor.

7.42.2 Member Function Documentation**7.42.2.1 template<typename T > T ProtocolPP::jicmpsa::get_field (*field_t field*)**

Returns the version field of the IP header

Some fields may only be available in the jicmp protocol due to their dynamic nature as fields in the ICMP header or packet and are not stored in the security association. For ICMP they include

- CHECKSUM
- POINTER
- TYPE
- SEQNUM
- GATEWAY
- ORIGTIMESTAMP

field type	field name	Example
direction_t	DIRECTION	ProtocolPP::direction_t mydir = get_field<ProtocolPP::direction_t>(ProtocolPP::field_t::DIRECTION)
iana_t	VERSION	ProtocolPP::iana_t myver = get_field<ProtocolPP::iana_t>(-ProtocolPP::field_t::VERSION)
icmpmsg_t	MESSAGE	ProtocolPP::icmpmsg_t mymsg = get_field<ProtocolPP::icmpmsg_t>(ProtocolPP::field_t::MESSAGE)
icmpcode_t	CODE	ProtocolPP::icmpcode_t mycode = get_field<ProtocolPP::icmpcode_t>(ProtocolPP::field_t::CODE)
uint8_t	DSECN	uint8_t mydsecn = get_field<uint8_t>(ProtocolPP::field_t::DSECN)
	TTLHOP	uint8_t myttl = get_field<uint8_t>(ProtocolPP::field_t::TTLHOP)
	FLAGS	uint8_t myflags = get_field<uint8_t>(ProtocolPP::field_t::FLAGS)
uint16_t	FRAGOFFSET	uint16_t myfragoff = get_field<uint16_t>(ProtocolPP::field_t::FRAGOFFSET)
	ID	uint16_t myid = get_field<uint16_t>(ProtocolPP::field_t::ID)
	LABEL	uint32_t mylabel = get_field<uint32_t>(ProtocolPP::field_t::LABEL)
jarray<uint8_t>	SOURCE	jarray<uint8_t> mysrc = get_field<jarray<uint8_t>>(ProtocolPP::field_t::SOURCE)
	DESTINATION	jarray<uint8_t> mydst = get_field<jarray<uint8_t>>(ProtocolPP::field_t::DESTINATION)

Table 7.8: ICMP SA Get Fields

- RXTIMESTAMP
- TXTIMESTAMP

Parameters

<i>field</i>	- field to retrieve from the IP header
--------------	--

Returns

value of the field in the security association

7.42.2.2 template<typename T > void ProtocolPP::jicmpsa::set_field (field_t *field*, T *fieldval*)

Allows the user to update the field of the IP header

Some fields may only be available in the jicmp protocol due to their dynamic nature as fields in the ICMP header or packet and are not stored in the security association. For ICMP they include

- CHECKSUM
- POINTER
- TYPE
- SEQNUM
- GATEWAY
- ORIGTIMESTAMP
- RXTIMESTAMP
- TXTIMESTAMP

Parameters

<i>field</i>	- field to update the IP header with
<i>fieldval</i>	- value to update the IP header with

7.42.2.3 void ProtocolPP::jicmpsa::to_xml (tinyxml2::XMLPrinter & *myxml*, direction_t *direction*) [virtual]

Prints the protocol as an XML object

Parameters

<i>myxml</i>	- XMLPrinter object to print with
<i>direction</i>	- randomization

Implements [ProtocolPP::jsecass](#).

The documentation for this class was generated from the following file:

- [include/jicmpsa.h](#)

7.43 ProtocolPP::jikeparse::jikecfg Struct Reference

```
#include <jikeparse.h>
```

Public Member Functions

- [jikecfg \(\)](#)

Public Attributes

- std::string [connection](#)
- std::string [format](#)
- unsigned int [max_ike_negotiation](#)
- unsigned int [cookies_lifetime](#)
- unsigned int [cookies_threshold](#)
- std::string [vendor_id](#)
- unsigned long [seed](#)
- bool [show_extra_info](#)
- std::string [show_mask](#)
- std::string [hide_mask](#)
- std::string [debugclr](#)

- std::string `infoclr`
- std::string `warnclr`
- std::string `errclr`
- std::string `fatalclr`
- std::string `passclr`
- bool `flush_before`
- bool `generate_allow_policies`
- std::string `role`
- std::string `peer_addr`
- unsigned int `retransmition_time`
- unsigned int `retransmition_factor`
- unsigned int `max_retries`
- unsigned int `rekey_time`
- unsigned int `max_idle_time`
- unsigned int `reauth_time`
- bool `use_uname`
- std::string `auth_generator`
- std::string `auth_verifiers`
- std::string `preshared_key`
- std::string `peer_preshared_key`
- std::string `certificates`
- std::string `ca_certificates`
- std::string `send_cert_payload`
- std::string `peer_ca_certificates`
- std::string `cert_white_list`
- std::string `cert_black_list`
- bool `hash_url_support`
- bool `send_cert_req_payload`
- std::string `eap_clients`
- std::string `eap_md5_password`
- std::string `eap_tls_client_cert`
- std::string `eap_tls_private_key`
- std::string `eap_tls_private_key_password`
- std::string `eap_tls_ca_cert`
- std::string `eap_server`
- std::string `eap_md5_user_db`
- std::string `radius_server`
- unsigned int `radius_port`
- std::string `radius_secret`
- std::string `local_type`
- std::string `local_id`
- std::string `peerid_type`
- std::string `peerid_id`
- std::string `ike_encr`
- std::string `ike_integ`
- std::string `ike_prf`
- std::string `ike_dh`
- std::string `remote_port`
- std::string `remote_host`
- std::string `method`
- std::string `espemode`
- `jarray< uint8_t > protected_ipv4_subnet`
- `jarray< uint8_t > protected_ipv4_subnet_mask`
- `jarray< uint8_t > protected_ipv6_subnet`
- `jarray< uint8_t > protected_ipv6_subnet_mask`

- `jarray< uint8_t > fixed_ipv4_prefix`
- `jarray< uint8_t > fixed_ipv4_prefix_mask`
- `jarray< uint8_t > fixed_ipv6_prefix`
- `jarray< uint8_t > fixed_ipv6_prefix_mask`
- `std::string dhcp_interface`
- `std::string dhcp_server_ip`
- `unsigned int dhcp_timeout`
- `unsigned int dhcp_retries`
- `bool request_configuration`
- `std::string request_ipv6_suffix`
- `unsigned int lifetime_soft`
- `unsigned int lifetime_hard`
- `unsigned int max_bytes_soft`
- `unsigned int max_bytes_hard`
- `std::string esp_encr`
- `std::string esp_integ`
- `std::string esp_dh`
- `bool use_esn`
- `bool bypassdf`
- `bool bypassdscp`
- `bool randiv`
- `unsigned int tfclen`
- `unsigned int arwindow`
- `std::string ah_integ`
- `jarray< uint8_t > noncelInitiator`
- `jarray< uint8_t > nonceResponder`
- `std::unordered_map< std::string, jikepolicy > ikepolicies`

7.43.1 Constructor & Destructor Documentation

7.43.1.1 `ProtocolPP::jikeparse::jikecfg::jikecfg() [inline]`

7.43.2 Member Data Documentation

7.43.2.1 `std::string ProtocolPP::jikeparse::jikecfg::ah_integ`

7.43.2.2 `unsigned int ProtocolPP::jikeparse::jikecfg::arwindow`

7.43.2.3 `std::string ProtocolPP::jikeparse::jikecfg::auth_generator`

7.43.2.4 `std::string ProtocolPP::jikeparse::jikecfg::auth_verifiers`

7.43.2.5 `bool ProtocolPP::jikeparse::jikecfg::bypassdf`

7.43.2.6 `bool ProtocolPP::jikeparse::jikecfg::bypassdscp`

7.43.2.7 `std::string ProtocolPP::jikeparse::jikecfg::ca_certificates`

7.43.2.8 `std::string ProtocolPP::jikeparse::jikecfg::cert_black_list`

7.43.2.9 `std::string ProtocolPP::jikeparse::jikecfg::cert_white_list`

7.43.2.10 `std::string ProtocolPP::jikeparse::jikecfg::certificates`

```
7.43.2.11 std::string ProtocolPP::jikeparse::jikecfg::connection
7.43.2.12 unsigned int ProtocolPP::jikeparse::jikecfg::cookies_lifetime
7.43.2.13 unsigned int ProtocolPP::jikeparse::jikecfg::cookies_threshold
7.43.2.14 std::string ProtocolPP::jikeparse::jikecfg::debugclr
7.43.2.15 std::string ProtocolPP::jikeparse::jikecfg::dhcp_interface
7.43.2.16 unsigned int ProtocolPP::jikeparse::jikecfg::dhcp_retries
7.43.2.17 std::string ProtocolPP::jikeparse::jikecfg::dhcp_server_ip
7.43.2.18 unsigned int ProtocolPP::jikeparse::jikecfg::dhcp_timeout
7.43.2.19 std::string ProtocolPP::jikeparse::jikecfg::eap_clients
7.43.2.20 std::string ProtocolPP::jikeparse::jikecfg::eap_md5_password
7.43.2.21 std::string ProtocolPP::jikeparse::jikecfg::eap_md5_user_db
7.43.2.22 std::string ProtocolPP::jikeparse::jikecfg::eap_server
7.43.2.23 std::string ProtocolPP::jikeparse::jikecfg::eap_tls_ca_cert
7.43.2.24 std::string ProtocolPP::jikeparse::jikecfg::eap_tls_client_cert
7.43.2.25 std::string ProtocolPP::jikeparse::jikecfg::eap_tls_private_key
7.43.2.26 std::string ProtocolPP::jikeparse::jikecfg::eap_tls_private_key_password
7.43.2.27 std::string ProtocolPP::jikeparse::jikecfg::errclr
7.43.2.28 std::string ProtocolPP::jikeparse::jikecfg::esp_dh
7.43.2.29 std::string ProtocolPP::jikeparse::jikecfg::esp_encr
7.43.2.30 std::string ProtocolPP::jikeparse::jikecfg::esp_integ
7.43.2.31 std::string ProtocolPP::jikeparse::jikecfg::espmode
7.43.2.32 std::string ProtocolPP::jikeparse::jikecfg::fatalclr
7.43.2.33 jarray<uint8_t> ProtocolPP::jikeparse::jikecfg::fixed_ipv4_prefix
7.43.2.34 jarray<uint8_t> ProtocolPP::jikeparse::jikecfg::fixed_ipv4_prefix_mask
7.43.2.35 jarray<uint8_t> ProtocolPP::jikeparse::jikecfg::fixed_ipv6_prefix
7.43.2.36 jarray<uint8_t> ProtocolPP::jikeparse::jikecfg::fixed_ipv6_prefix_mask
7.43.2.37 bool ProtocolPP::jikeparse::jikecfg::flush_before
7.43.2.38 std::string ProtocolPP::jikeparse::jikecfg::format
```

7.43.2.39 bool ProtocolPP::jikeparse::jikecfg::generate_allow_policies
7.43.2.40 bool ProtocolPP::jikeparse::jikecfg::hash_url_support
7.43.2.41 std::string ProtocolPP::jikeparse::jikecfg::hide_mask
7.43.2.42 std::string ProtocolPP::jikeparse::jikecfg::ike_dh
7.43.2.43 std::string ProtocolPP::jikeparse::jikecfg::ike_encr
7.43.2.44 std::string ProtocolPP::jikeparse::jikecfg::ike_integ
7.43.2.45 std::string ProtocolPP::jikeparse::jikecfg::ike_prf
7.43.2.46 std::unordered_map<std::string,jikepolicy> ProtocolPP::jikeparse::jikecfg::ikepolicies
7.43.2.47 std::string ProtocolPP::jikeparse::jikecfg::infoclr
7.43.2.48 unsigned int ProtocolPP::jikeparse::jikecfg::lifetime_hard
7.43.2.49 unsigned int ProtocolPP::jikeparse::jikecfg::lifetime_soft
7.43.2.50 std::string ProtocolPP::jikeparse::jikecfg::local_id
7.43.2.51 std::string ProtocolPP::jikeparse::jikecfg::local_type
7.43.2.52 unsigned int ProtocolPP::jikeparse::jikecfg::max_bytes_hard
7.43.2.53 unsigned int ProtocolPP::jikeparse::jikecfg::max_bytes_soft
7.43.2.54 unsigned int ProtocolPP::jikeparse::jikecfg::max_idle_time
7.43.2.55 unsigned int ProtocolPP::jikeparse::jikecfg::max_ike_negotiation
7.43.2.56 unsigned int ProtocolPP::jikeparse::jikecfg::max_retries
7.43.2.57 std::string ProtocolPP::jikeparse::jikecfg::method
7.43.2.58 jarray<uint8_t> ProtocolPP::jikeparse::jikecfg::noncelInitiator
7.43.2.59 jarray<uint8_t> ProtocolPP::jikeparse::jikecfg::nonceResponder
7.43.2.60 std::string ProtocolPP::jikeparse::jikecfg::passclr
7.43.2.61 std::string ProtocolPP::jikeparse::jikecfg::peer_addr
7.43.2.62 std::string ProtocolPP::jikeparse::jikecfg::peer_ca_certificates
7.43.2.63 std::string ProtocolPP::jikeparse::jikecfg::peer_preshared_key
7.43.2.64 std::string ProtocolPP::jikeparse::jikecfg::peerid_id
7.43.2.65 std::string ProtocolPP::jikeparse::jikecfg::peerid_type
7.43.2.66 std::string ProtocolPP::jikeparse::jikecfg::preshared_key

7.43.2.67 `jarray<uint8_t> ProtocolPP::jikeparse::jikecfg::protected_ipv4_subnet`

7.43.2.68 `jarray<uint8_t> ProtocolPP::jikeparse::jikecfg::protected_ipv4_subnet_mask`

7.43.2.69 `jarray<uint8_t> ProtocolPP::jikeparse::jikecfg::protected_ipv6_subnet`

7.43.2.70 `jarray<uint8_t> ProtocolPP::jikeparse::jikecfg::protected_ipv6_subnet_mask`

7.43.2.71 `unsigned int ProtocolPP::jikeparse::jikecfg::radius_port`

7.43.2.72 `std::string ProtocolPP::jikeparse::jikecfg::radius_secret`

7.43.2.73 `std::string ProtocolPP::jikeparse::jikecfg::radius_server`

7.43.2.74 `bool ProtocolPP::jikeparse::jikecfg::randiv`

7.43.2.75 `unsigned int ProtocolPP::jikeparse::jikecfg::reauth_time`

7.43.2.76 `unsigned int ProtocolPP::jikeparse::jikecfg::rekey_time`

7.43.2.77 `std::string ProtocolPP::jikeparse::jikecfg::remote_host`

7.43.2.78 `std::string ProtocolPP::jikeparse::jikecfg::remote_port`

7.43.2.79 `bool ProtocolPP::jikeparse::jikecfg::request_configuration`

7.43.2.80 `std::string ProtocolPP::jikeparse::jikecfg::request_ipv6_suffix`

7.43.2.81 `unsigned int ProtocolPP::jikeparse::jikecfg::retransmition_factor`

7.43.2.82 `unsigned int ProtocolPP::jikeparse::jikecfg::retransmition_time`

7.43.2.83 `std::string ProtocolPP::jikeparse::jikecfg::role`

7.43.2.84 `unsigned long ProtocolPP::jikeparse::jikecfg::seed`

7.43.2.85 `std::string ProtocolPP::jikeparse::jikecfg::send_cert_payload`

7.43.2.86 `bool ProtocolPP::jikeparse::jikecfg::send_cert_req_payload`

7.43.2.87 `bool ProtocolPP::jikeparse::jikecfg::show_extra_info`

7.43.2.88 `std::string ProtocolPP::jikeparse::jikecfg::show_mask`

7.43.2.89 `unsigned int ProtocolPP::jikeparse::jikecfg::tfclen`

7.43.2.90 `bool ProtocolPP::jikeparse::jikecfg::use_esn`

7.43.2.91 `bool ProtocolPP::jikeparse::jikecfg::use_uname`

7.43.2.92 `std::string ProtocolPP::jikeparse::jikecfg::vendor_id`

7.43.2.93 `std::string ProtocolPP::jikeparse::jikecfg::warnclr`

The documentation for this struct was generated from the following file:

- `jikev2/include/jikeparse.h`

7.44 ProtocolPP::jikencrypt Class Reference

```
#include <jikencrypt.h>
```

Public Member Functions

- `jikencrypt (std::shared_ptr< InterfacePP::jlogger > &logger, std::shared_ptr< jrand > &rand)`
- `~jikencrypt ()`
Standard deconstructor.
- int `get_random (uint8_t *buf, unsigned int len)`
- bool `encrypt (direction_t dir, std::shared_ptr< jikev2sa > &sa, jarray< uint8_t > &input, std::shared_ptr< jarray< uint8_t >> &output, std::shared_ptr< jarray< uint8_t >> &iv)`
- bool `integrity (direction_t dir, std::shared_ptr< jikev2sa > &sa, jarray< uint8_t > &input, std::shared_ptr< jarray< uint8_t >> &icv)`

7.44.1 Constructor & Destructor Documentation

7.44.1.1 `ProtocolPP::jikencrypt::jikencrypt (std::shared_ptr< InterfacePP::jlogger > & logger, std::shared_ptr< jrand > & rand)`

Standard constructor

Parameters

<code>logger</code>	- Logger object to write to
<code>rand</code>	- randomizer

7.44.1.2 `ProtocolPP::jikencrypt::~jikencrypt () [inline]`

Standard deconstructor.

7.44.2 Member Function Documentation

7.44.2.1 `bool ProtocolPP::jikencrypt::encrypt (direction_t dir, std::shared_ptr< jikev2sa > & sa, jarray< uint8_t > & input, std::shared_ptr< jarray< uint8_t >> & output, std::shared_ptr< jarray< uint8_t >> & iv)`

Encrypt the IKE transactions

Parameters

<code>dir</code>	- Direction of processing
<code>sa</code>	- IKE security association
<code>input</code>	- packet to encrypt or decrypt
<code>output</code>	- encrypted or decrypted packet
<code>iv</code>	- IV generated for encryption/decryption

7.44.2.2 `int ProtocolPP::jikencrypt::get_random (uint8_t * buf, unsigned int len)`

Get random numbers from the kernel

Parameters

<i>buf</i>	- buffer to hold random bytes
<i>len</i>	- number of bytes to retrieve

7.44.2.3 `bool ProtocolPP::jikencrypt::integrity(direction_t dir, std::shared_ptr<jikev2sa> &sa, jarray<uint8_t> &input, std::shared_ptr<jarray<uint8_t>> &icv)`

Authenticate the IKE transactions

Parameters

<i>dir</i>	- Direction of processing
<i>sa</i>	- IKE security association
<i>input</i>	- packet to authenticate
<i>icv</i>	- Integrity check value (zero if direction is out)

Returns

Whether the ICV is correct or not

The documentation for this class was generated from the following file:

- `jikev2/include/jikencrypt.h`

7.45 jikencrypt Class Reference

```
#include "include/jikencrypt.h"
```

7.45.1 Detailed Description

7.45.2 IKE Encrypt and Integrity Functions

For API information:

See Also

[ProtocolPP::jikev2](#)
[ProtocolPP::jrand](#)
[ProtocolPP::jenum](#)
[ProtocolPP::jsecass](#)
[ProtocolPP::jikev2sa](#)
[ProtocolPP::jipsecrsa](#)
[InterfacePP::ringflow](#)

For Additional Documentation:

See Also

[jikev2](#)
[jrand](#)
[jenum](#)
[jsecass](#)
[jikev2sa](#)
[jipsecrsa](#)
[ringflow](#)

Protocol++ written by : John Peter Greninger © John Peter Greninger 2015-2019 All Rights Reserved

All copyrights and trademarks are the property of their respective owners

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer
- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution
- Any and all modifications must be returned to John Peter Greninger at GitHub.com <https://github.com/jpgreninger/protocolpp> for evaluation. Inclusion of modifications in the source code shall be determined solely by John Peter Greninger. Failure to provide modifications shall render this license NULL and VOID and revoke any rights to use of Protocol++
- Commercial use requires a fee-based license obtainable at www.protocolpp.com
- Academic use requires written and notarized permission from John Peter and Sheila Rocha Greninger
- US Copyrights at <https://www.copyright.gov/>
 - TXU002059872 (Version 1.0.0)
 - TXU002066632 (Version 1.2.7)
 - TXU002082674 (Version 1.4.0)
 - TXU002097880 (Version 2.0.0)

The names of its contributors may not be used to endorse or promote products derived from this software without specific prior written permission

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE

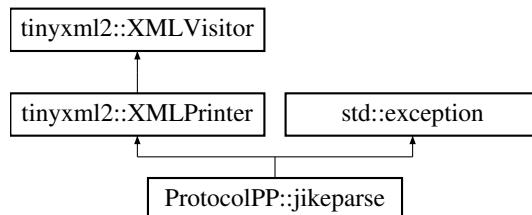
The documentation for this class was generated from the following file:

- jikev2/include/jikencrypt.h

7.46 ProtocolPP::jikeparse Class Reference

```
#include <jikeparse.h>
```

Inheritance diagram for ProtocolPP::jikeparse:



Classes

- struct [jikecfg](#)
- struct [jikepolicy](#)

Public Member Functions

- [jikeparse](#) (const char *input, const char *out=0)
- [jikeparse](#) (std::string &input, const char *out=0)
- [jikeparse](#) (std::shared_ptr<[tinyxml2::XMLDocument](#)> &input, const char *out=0)
- virtual [~jikeparse](#) ()
JIKEPARSE standard deconstructor.
- void [get](#) (std::shared_ptr<[jikecfg](#)> &cfg)

Additional Inherited Members

7.46.1 Constructor & Destructor Documentation

7.46.1.1 ProtocolPP::jikeparse::jikeparse (const char * *input*, const char * *out* = 0)

JIKEPARSE - Parse IKEv2 configuration files

Parameters

<i>input</i>	- XML data represented in a character string
<i>out</i>	- output file to write to (if desired)

7.46.1.2 ProtocolPP::jikeparse::jikeparse (std::string & *input*, const char * *out* = 0)

JIKEPARSE - Parse IKEv2 configuration files

Parameters

<i>input</i>	- Filename of an XML data file to load
<i>out</i>	- output file to write to (if desired)

7.46.1.3 ProtocolPP::jikeparse::jikeparse (std::shared_ptr<[tinyxml2::XMLDocument](#)> & *input*, const char * *out* = 0)

JIKEPARSE - Parse IKEv2 configuration files

Parameters

<i>input</i>	- XMLDocument to load
<i>out</i>	- output file to write to (if desired)

7.46.1.4 virtual ProtocolPP::jikeparse::~jikeparse () [inline], [virtual]

JIKEPARSE standard deconstructor.

7.46.2 Member Function Documentation

7.46.2.1 void ProtocolPP::jikeparse::get (std::shared_ptr<[jikecfg](#)> & *cfg*)

Returns

- shared pointer to JIKECFG object

The documentation for this class was generated from the following file:

- jikev2/include/jikeparse.h

7.47 jikeparse Class Reference

```
#include "include/jikeparse.h"
```

7.47.1 Detailed Description

7.47.2 IKEPARSE

For API Documentation:

See Also

- [tinyxml2::XMLPrinter](#)
- [tinyxml2::XMLVisitor](#)

For Additional Documentation:

See Also

- [tinyxml2::XMLPrinter](#)
- [tinyxml2::XMLVisitor](#)

Protocol++ written by : John Peter Greninger © John Peter Greninger 2015-2019 All Rights Reserved

All copyrights and trademarks are the property of their respective owners

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer
- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution
- Any and all modifications must be returned to John Peter Greninger at GitHub.com <https://github.com/jpgreninger/protocolpp> for evaluation. Inclusion of modifications in the source code shall be determined solely by John Peter Greninger. Failure to provide modifications shall render this license NULL and VOID and revoke any rights to use of Protocol++
- Commercial use requires a fee-based license obtainable at www.protocolpp.com
- Academic use requires written and notarized permission from John Peter and Sheila Rocha Greninger
- US Copyrights at <https://www.copyright.gov/>
 - TXu002059872 (Version 1.0.0)
 - TXu002066632 (Version 1.2.7)
 - TXu002082674 (Version 1.4.0)

- TXu002097880 (Version 2.0.0)

The names of its contributors may not be used to endorse or promote products derived from this software without specific prior written permission

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE

The documentation for this class was generated from the following file:

- [jikev2/include/jikeparse.h](#)

7.48 ProtocolPP::jikeparse::jikepolicy Struct Reference

```
#include <jikeparse.h>
```

Public Member Functions

- [jikepolicy \(\)](#)

Public Attributes

- [jarray< uint8_t > src_selector](#)
- [jarray< uint8_t > src_selector_mask](#)
- [jarray< uint8_t > dst_selector](#)
- [jarray< uint8_t > dst_selector_mask](#)
- [uint16_t src_port](#)
- [uint16_t dst_port](#)
- [iana_t ip_proto](#)
- [direction_t dir](#)
- [iana_t ipsec_proto](#)
- [ipmode_t mode](#)
- [jarray< uint8_t > src_tunnel](#)
- [jarray< uint8_t > dst_tunnel](#)
- [unsigned int priority](#)

7.48.1 Constructor & Destructor Documentation

7.48.1.1 [ProtocolPP::jikeparse::jikepolicy\(\) \[inline\]](#)

7.48.2 Member Data Documentation

7.48.2.1 [direction_t ProtocolPP::jikeparse::jikepolicy::dir](#)

7.48.2.2 [uint16_t ProtocolPP::jikeparse::jikepolicy::dst_port](#)

- 7.48.2.3 `jarray<uint8_t> ProtocolPP::jikeparse::jikepolicy::dst_selector`
- 7.48.2.4 `jarray<uint8_t> ProtocolPP::jikeparse::jikepolicy::dst_selector_mask`
- 7.48.2.5 `jarray<uint8_t> ProtocolPP::jikeparse::jikepolicy::dst_tunnel`
- 7.48.2.6 `iana_t ProtocolPP::jikeparse::jikepolicy::ip_proto`
- 7.48.2.7 `iana_t ProtocolPP::jikeparse::jikepolicy::ipsec_proto`
- 7.48.2.8 `ipmode_t ProtocolPP::jikeparse::jikepolicy::mode`
- 7.48.2.9 `unsigned int ProtocolPP::jikeparse::jikepolicy::priority`
- 7.48.2.10 `uint16_t ProtocolPP::jikeparse::jikepolicy::src_port`
- 7.48.2.11 `jarray<uint8_t> ProtocolPP::jikeparse::jikepolicy::src_selector`
- 7.48.2.12 `jarray<uint8_t> ProtocolPP::jikeparse::jikepolicy::src_selector_mask`
- 7.48.2.13 `jarray<uint8_t> ProtocolPP::jikeparse::jikepolicy::src_tunnel`

The documentation for this struct was generated from the following file:

- `jikev2/include/jikeparse.h`

7.49 ProtocolPP::jikeprf Class Reference

```
#include <jikeprf.h>
```

Public Member Functions

- `jikeprf()`

Standard constructor.
- `~jikeprf()`

Standard deconstructor.
- `void get_prf (prf_id_t prf, jarray< uint8_t > &key, jarray< uint8_t > &data, std::shared_ptr< jarray< uint8_t >> &output)`
- `void get_prfplus (prf_id_t prf, jarray< uint8_t > &key, jarray< uint8_t > &data, std::shared_ptr< jarray< uint8_t >> &output, unsigned int length)`

7.49.1 Constructor & Destructor Documentation

7.49.1.1 ProtocolPP::jikeprf::jikeprf()

Standard constructor.

7.49.1.2 ProtocolPP::jikeprf::~jikeprf() [inline]

Standard deconstructor.

7.49.2 Member Function Documentation

7.49.2.1 void ProtocolIPP::jikeprf::get_prf (*prf_id_t prf*, *jarray< uint8_t > & key*, *jarray< uint8_t > & data*,
std::shared_ptr< jarray< uint8_t >> & output)

Computes the hash result for PRF for IKEPRFv1 and IKEPRFv2

Parameters

<i>prf</i>	- PRF function to use (MD5, SHA, SHA256, SHA384, SHA512, AES_XCBC_MAC, AES_C-MAC depending on IKEPRF version)
<i>key</i>	- key for the PRF function
<i>data</i>	- data to hash
<i>output</i>	- pointer to hold SKEYSEED the size of the HASH output

7.49.2.2 void ProtocolPP::jikeprf::get_prfplus (*prf_id_t prf*, *jarray< uint8_t > & key*, *jarray< uint8_t > & data*,
std::shared_ptr< jarray< uint8_t >> & output, *unsigned int length*)

Computes the keying material for IKEPRFv1 and IKEPRFv2

Parameters

<i>prf</i>	- PRF function to use (MD5, SHA, SHA256, SHA384, SHA512, AES_XCBC_MAC, AES_C-MAC depending on IKEPRF version)
<i>key</i>	- Child SA key generated from the SKEYSEED (SKd) or the generated SKEYSEED if generating IKEV2 key material
<i>data</i>	- Security Parameter Indexes (SPI) SPIi and SPIr (may include seconDGRAMd ephemeral secret SECRET2 SPIi SPIr) or for IKEV2 key material, NONCES and SPIs (Ni Nr SPIi SPIr) to generate IKE connection keys
<i>output</i>	- pointer to hold generated data
<i>length</i>	- length of the desired output data

The documentation for this class was generated from the following file:

- [jikev2/include/jikeprf.h](#)

7.50 jikeprf Class Reference

```
#include "include/jikeprf.h"
```

7.50.1 Detailed Description

7.50.2 IKE PRF and PRF+ functions

For API information:

See Also

[ProtocolPP::jikev2](#)
[ProtocolPP::jrand](#)
[ProtocolPP::jenum](#)
[ProtocolPP::jsecass](#)
[ProtocolPP::jikev2sa](#)
[ProtocolPP::jipsecsa](#)

For Additional Documentation:

See Also

[jikev2](#)
[jrand](#)
[jenum](#)
[jsecass](#)

jikev2sa
jipsecsa

Protocol++ written by : John Peter Greninger © John Peter Greninger 2015-2019 All Rights Reserved

All copyrights and trademarks are the property of their respective owners

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer
- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution
- Any and all modifications must be returned to John Peter Greninger at GitHub.com <https://github.com/jpgreninger/protocolpp> for evaluation. Inclusion of modifications in the source code shall be determined solely by John Peter Greninger. Failure to provide modifications shall render this license NULL and VOID and revoke any rights to use of Protocol++
- Commercial use requires a fee-based license obtainable at www.protocolpp.com
- Academic use requires written and notarized permission from John Peter and Sheila Rocha Greninger
- US Copyrights at [https://www.copyright.gov/](http://www.copyright.gov/)
 - TXu002059872 (Version 1.0.0)
 - TXu002066632 (Version 1.2.7)
 - TXu002082674 (Version 1.4.0)
 - TXu002097880 (Version 2.0.0)

The names of its contributors may not be used to endorse or promote products derived from this software without specific prior written permission

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE

The documentation for this class was generated from the following file:

- jikev2/include/jikeprf.h

7.51 ProtocolPP::jikev2 Class Reference

```
#include <jikev2.h>
```

Public Member Functions

- `jikev2 (std::shared_ptr< InterfacePP::jlogger > &logger)`
- `jikev2 (const char *config, std::shared_ptr< InterfacePP::jlogger > &logger)`
- `~jikev2 ()`
Standard deconstructor.
- `void run ()`
Run IKEv2.
- `void teardown ()`
Teardown IKEv2.

Protected Member Functions

- `bool reload (const char *config)`
- `bool propose (std::shared_ptr< jikeparse::jikepolicy > &policy, bool connect)`
- `uint32_t add_sa (std::shared_ptr< jipsecsa > &sa)`
- `uint32_t delete_sa (uint64_t spi, unsigned int spisize)`
- `uint32_t findsa (uint64_t spi, std::shared_ptr< jikev2sa > &sa)`
- `bool comms (std::shared_ptr< jarray< uint8_t >> &input, std::shared_ptr< jarray< uint8_t >> &output, bool wait=true)`
- `bool get_fields (uint8_t *input, std::shared_ptr< jikev2sa > &sa1)`
- `bool get_fields (uint8_t *input, std::shared_ptr< jipsecsa > &sa1, std::shared_ptr< jipsecsa > &sa2)`

7.51.1 Constructor & Destructor Documentation

7.51.1.1 ProtocolPP::jikev2::jikev2 (std::shared_ptr< InterfacePP::jlogger > & logger)

Standard constructor

Parameters

<code>logger</code>	- Logger object to write to
---------------------	-----------------------------

7.51.1.2 ProtocolPP::jikev2::jikev2 (const char * config, std::shared_ptr< InterfacePP::jlogger > & logger)

Constructor with configuration file

Parameters

<code>config</code>	- Configuration(s) to initiate connection with
<code>logger</code>	- Logger object to write to

7.51.1.3 ProtocolPP::jikev2::~jikev2 () [inline]

Standard deconstructor.

7.51.2 Member Function Documentation

7.51.2.1 uint32_t ProtocolPP::jikev2::add_sa (std::shared_ptr< jipsecsa > & sa) [protected]

Add a new security association to the key ring, Updates the security association if it already exists

Parameters

<i>sa</i>	- Security Association to add to the ring
-----------	---

Returns

status for adding the key

7.51.2.2 bool ProtocolPP::jikev2::comms (std::shared_ptr<jarray< uint8_t >> &*input*, std::shared_ptr<jarray< uint8_t >>& *output*, bool *wait* = true) [protected]

Create a Security Association with the negotiated parameters

Parameters

<i>input</i>	- input packet
<i>output</i>	- output packet
<i>wait</i>	- use blocking socket

7.51.2.3 uint32_t ProtocolPP::jikev2::delete_sa (uint64_t *spi*, unsigned int *spisize*) [protected]

Delete a security association in the key ring

Parameters

<i>spi</i>	- security protocol index for the IKE connection
<i>spisize</i>	- size of the SPI in bytes (4 or 8)

Returns

- status of the lookup

7.51.2.4 uint32_t ProtocolPP::jikev2::findsa (uint64_t *spi*, std::shared_ptr<jikev2sa >& *sa*) [protected]

Find a security association in the key ring

Parameters

<i>spi</i>	- security protocol index for the IKE connection
<i>sa</i>	- Security Association to return from the ring

Returns

- status of the lookup

7.51.2.5 bool ProtocolPP::jikev2::get_fields (uint8_t * *input*, std::shared_ptr<jikev2sa >& *sa1*) [protected]

Parse the input and fill the Security Association(s)

Parameters

<i>input</i>	- input packet
--------------	----------------

<i>sa1</i>	- Security Association (IKE)
------------	------------------------------

7.51.2.6 `bool ProtocolPP::jikev2::get_fields(uint8_t * input, std::shared_ptr<jipsecsa> & sa1, std::shared_ptr<jipsecsa> & sa2) [protected]`

Parse the input and fill the Security Association(s)

Parameters

<i>input</i>	- input packet
<i>sa1</i>	- Security Association (ESP_IN)
<i>sa2</i>	- Security Association (ESP_OUT)

7.51.2.7 `bool ProtocolPP::jikev2::propose(std::shared_ptr<jikeparse::jikepolicy> & policy, bool connect) [protected]`

Propose a new connection

Parameters

<i>policy</i>	- New policy to add to database
<i>connect</i>	- Create the connection as initiator

Returns

True for success, False for connection failure

7.51.2.8 `bool ProtocolPP::jikev2::reload(const char * config) [protected]`

reload the configuration for IKEv2 with new configuration

Parameters

<i>config</i>	- New configuration file
---------------	--------------------------

Returns

True for success, False for connection failure

7.51.2.9 `void ProtocolPP::jikev2::run()`

Run IKEv2.

7.51.2.10 `void ProtocolPP::jikev2::teardown()`

Teardown IKEv2.

The documentation for this class was generated from the following file:

- jikev2/include/jikev2.h

7.52 jikev2 Class Reference

```
#include "include/jikev2.h"
```

7.52.1 Detailed Description

7.52.2 Internet Key Exchange Version 2

(See RFC7296 for complete details)

IKE performs mutual authentication between two parties and establishes an IKE security association (SA) that includes shared secret information that can be used to efficiently establish SAs for Encapsulating Security Payload (ESP) [RFC4303] and/or Authentication Header (AH) [RFC4302] and a set of cryptographic algorithms to be used by the SAs to protect the traffic that they carry. In this document, the term "suite" or "cryptographic suite" refers to a complete set of algorithms used to protect an SA. An initiator proposes one or more suites by listing supported algorithms that can be combined into suites in a mix-and-match fashion. IKE can also negotiate use of IP Compression (IPComp) [IPCOMP] in connection with an ESP and/or AH SA. We call the IKE SA an "IKE_SA". The SAs for ESP and/or AH that get set up through that IKE_SA we call "CHILD_SAs"

All IKE communications consist of pairs of messages: a request and a response. The pair is called an "exchange". We call the first messages establishing an IKE_SA IKE_SA_INIT and IKE_AUTH exchanges and subsequent IKE exchanges CREATE_CHILD_SA or INFORMATIONAL exchanges. In the common case, there is a single IKE_SA_INIT exchange and a single IKE_AUTH exchange (a total of four messages) to establish the IKE_SA and the first CHILD_SA. In exceptional cases, there may be more than one of each of these exchanges. In all cases, all IKE_SA_INIT exchanges MUST complete before any other exchange type, then all IKE_AUTH exchanges MUST complete, and following that any number of CREATE_CHILD_SA and INFORMATIONAL exchanges may occur in any order. In some scenarios, only a single CHILD_SA is needed between the IPsec endpoints, and therefore there would be no additional exchanges. Subsequent exchanges MAY be used to establish additional CHILD_SAs between the same authenticated pair of endpoints and to perform housekeeping functions

IKE message flow always consists of a request followed by a response. It is the responsibility of the requester to ensure reliability. If the response is not received within a timeout interval, the requester needs to retransmit the request (or abandon the connection)

The first request/response of an IKE session (IKE_SA_INIT) negotiates security parameters for the IKE_SA, sends nonces, and sends Diffie-Hellman values. The second request/response (IKE_AUTH) transmits identities, proves knowledge of the secrets corresponding to the two identities, and sets up an SA for the first (and often only) AH and/or ESP CHILD_SA. The types of subsequent exchanges are CREATE_CHILD_SA (which creates a CHILD_SA) and INFORMATIONAL (which deletes an SA, reports error conditions, or does other housekeeping). Every request requires a response. An INFORMATIONAL request with no payloads (other than the empty Encrypted payload required by the syntax) is commonly used as a check for liveness. These subsequent exchanges cannot be used until the initial exchanges have completed

7.52.3 Scenarios

Security Gateway to Security Gateway Tunnel

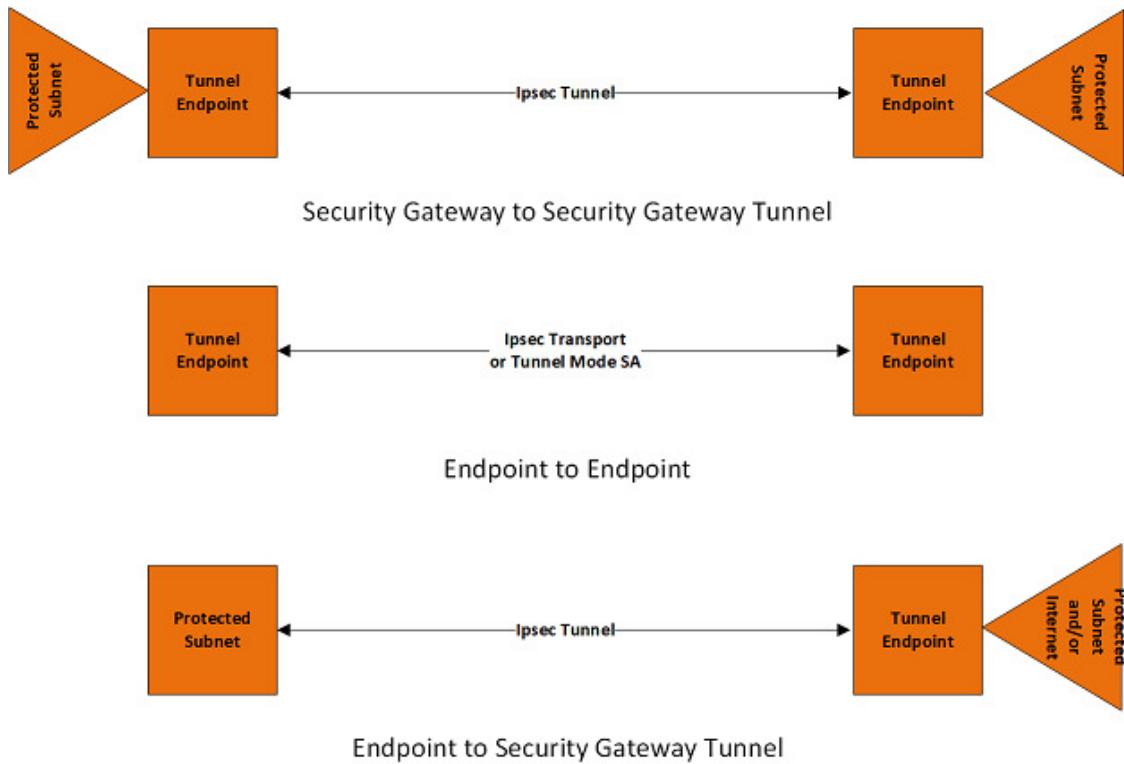
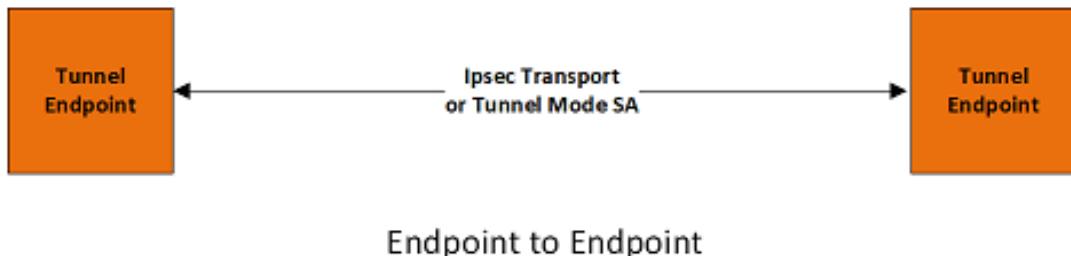


Figure 7.7: Security Gateway to Security Gateway Tunnel

In this scenario, neither endpoint of the IP connection implements IPsec, but network nodes between them protect traffic for part of the way. Protection is transparent to the endpoints, and depends on ordinary routing to send packets through the tunnel endpoints for processing. Each endpoint would announce the set of addresses "behind" it, and packets would be sent in tunnel mode where the inner IP header would contain the IP addresses of the actual endpoints

Endpoint-to-Endpoint Transport



Endpoint to Endpoint

Figure 7.8: Endpoint to Endpoint

In this scenario, both endpoints of the IP connection implement IPsec, as required of hosts in [RFC4301]. Transport mode will commonly be used with no inner IP header. If there is an inner IP header, the inner addresses will be the same as the outer addresses. A single pair of addresses will be negotiated for packets to be protected by this SA. These endpoints MAY implement application layer access controls based on the IPsec authenticated identities of the participants. This scenario enables the end-to-end security that has been a guiding principle for the Internet since [RFC1958], [RFC2775], and a method of limiting the inherent problems with complexity in networks noted by [RFC3439]. Although this scenario may not be fully applicable to the IPv4 Internet, it has been deployed successfully in specific scenarios within intranets using IKEv1. It should be more broadly enabled during the transition to IPv6 and with the adoption of IKEv2

It is possible in this scenario that one or both of the protected endpoints will be behind a network address translation (NAT) node, in which case the tunneled packets will have to be UDP encapsulated so that port numbers in the UDP headers can be used to identify individual endpoints "behind" the NAT (see section 2.23)

Endpoint to Security Gateway Tunnel

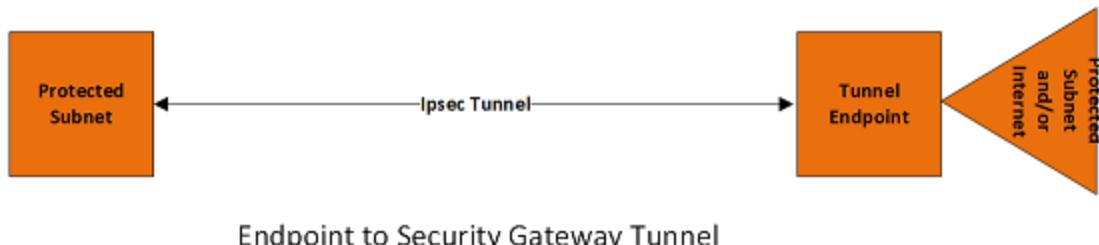


Figure 7.9: Endpoint to Security Gateway Tunnel

In this scenario, a protected endpoint (typically a portable roaming computer) connects back to its corporate network through an IPsec-protected tunnel. It might use this tunnel only to access information on the corporate network, or it might tunnel all of its traffic back through the corporate network in order to take advantage of protection provided by a corporate firewall against Internet-based attacks. In either case, the protected endpoint will want an IP address associated with the security gateway so that packets returned to it will go to the security gateway and be tunneled back. This IP address may be static or may be dynamically allocated by the security gateway. In support of the latter case, IKEv2 includes a mechanism for the initiator to request an IP address owned by the security gateway for use for the duration of its SA. In this scenario, packets will use tunnel mode. On each packet from the protected endpoint, the outer IP header will contain the source IP address associated with its current location (i.e., the address that will get traffic routed to the endpoint directly), while the inner IP header will contain the source IP address assigned by the security gateway (i.e., the address that will get traffic routed to the security gateway for forwarding to the endpoint). The outer destination address will always be that of the security gateway, while the inner destination address will be the ultimate destination for the packet.

Generating Keying Material

In the context of the IKE_SA, four cryptographic algorithms are negotiated: an encryption algorithm, an integrity protection algorithm, a Diffie-Hellman group, and a pseudo-random function (prf). The pseudo-random function is used for the construction of keying material for all of the cryptographic algorithms used in both the IKE_SA and the CHILD_SAs.

We assume that each encryption algorithm and integrity protection algorithm uses a fixed-size key and that any randomly chosen value of that fixed size can serve as an appropriate key. For algorithms that accept a variable length key, a fixed key size MUST be specified as part of the cryptographic transform negotiated. For algorithms for which not all values are valid keys (such as DES or 3DES with key parity), the algorithm by which keys are derived from arbitrary values MUST be specified by the cryptographic transform. For integrity protection functions based on Hashed Message Authentication Code (HMAC), the fixed key size is the size of the output of the underlying hash function. When the prf function takes a variable length key, variable length data, and produces a fixed-length output (e.g., when using HMAC), the formulas in this document apply. When the key for the prf function has fixed length, the data provided as a key is truncated or padded with zeros as necessary unless exceptional processing is explained following the formula

Keying material will always be derived as the output of the negotiated prf algorithm. Since the amount of keying material needed may be greater than the size of the output of the prf algorithm, we will use the prf iteratively. We will use the terminology prf^+ to describe the function that outputs a pseudo-random stream based on the inputs to a prf as follows: (where $|$ indicates concatenation)

$$prf + (K, S) = T1 \mid T2 \mid T3 \mid T4 \mid \dots$$

where:

- $T1 = prf(K, S \mid 0x01)$
 - $T2 = prf(K, T1 \mid S \mid 0x01)$

- $T3 = \text{prf}(K, T2 | S | 0x01)$
- $T4 = \text{prf}(K, T3 | S | 0x01)$

continuing as needed to compute all required keys. The keys are taken from the output string without regard to boundaries (e.g., if the required keys are a 256-bit Advanced Encryption Standard (AES) key and a 160-bit HMAC key, and the prf function generates 160 bits, the AES key will come from T1 and the beginning of T2, while the HMAC key will come from the rest of T2 and the beginning of T3). The constant concatenated to the end of each string feeding the prf is a single octet. prf+ in this document is not defined beyond 255 times the size of the prf output

Generating Keying Material for CHILD_SAs

The shared keys are computed as follows. A quantity called *SKEYSEED* is calculated from the nonces exchanged during the IKE_SA_INIT exchange and the Diffie-Hellman shared secret established during that exchange. *SKEYSEED* is used to calculate seven other secrets:

- SK_d - used for deriving new keys for the CHILD_SAs established with this IKE_SA
- SK_{ai} and SK_{ar} - used as a key to the integrity protection algorithm for authenticating the component messages of subsequent exchanges
- SK_{ei} and SK_{er} - used for encrypting (and of course decrypting) all subsequent exchanges
- SK_{pi} and SK_{pr} - which are used when generating an AUTH payload

SKEYSEED and its derivatives are computed as follows:

$$SKEYSEED = \text{prf}(Ni | Nr; g^ir)$$

$$SK_d | SK_{ai} | SK_{ar} | SK_{ei} | SK_{er} | SK_{pi} | SK_{pr} = \text{prf} + (SKEYSEED, Ni | Nr | SPIi | SPIr)$$

(indicating that the quantities SK_d , SK_{ai} , SK_{ar} , SK_{ei} , SK_{er} , SK_{pi} , and SK_{pr} are taken in order from the generated bits of the prf+). g^ir is the shared secret from the ephemeral Diffie-Hellman exchange. g^ir is represented as a string of octets in big endian order padded with zeros if necessary to make it the length of the modulus. Ni and Nr are the nonces, stripped of any headers. If the negotiated prf takes a fixed-length key and the lengths of Ni and Nr do not add up to that length, half the bits must come from Ni and half from Nr , taking the first bits of each

The two directions of traffic flow use different keys. The keys used to protect messages from the original initiator are SK_{ai} and SK_{ei} . The keys used to protect messages in the other direction are SK_{ar} and SK_{er} . Each algorithm takes a fixed number of bits of keying material, which is specified as part of the algorithm. For integrity algorithms based on a keyed hash, the key size is always equal to the length of the output of the underlying hash function

A single CHILD_SA is created by the IKE_AUTH exchange, and additional CHILD_SAs can optionally be created in CREATE_CHILD_SA exchanges. Keying material for them is generated as follows:

$$KEYMAT = \text{prf} + (SK_d, Ni | Nr)$$

Where Ni and Nr are the nonces from the IKE_SA_INIT exchange if this request is the first CHILD_SA created or the fresh Ni and Nr from the CREATE_CHILD_SA exchange if this is a subsequent creation.

For CREATE_CHILD_SA exchanges including an optional Diffie-Hellman exchange, the keying material is defined as:

$$KEYMAT = \text{prf} + (SK_d, g^ir (\text{new}) | Ni | Nr)$$

where g^ir (new) is the shared secret from the ephemeral Diffie-Hellman exchange of this CREATE_CHILD_SA exchange (represented as an octet string in big endian order padded with zeros in the high-order bits if necessary to make it the length of the modulus). A single CHILD_SA negotiation may result in multiple security associations. ESP and AH SAs exist in pairs (one in each direction), and four SAs could be created in a single CHILD_SA negotiation if a combination of ESP and AH is being negotiated. Keying material MUST be taken from the expanded KEYMAT in the following order:

- All keys for SAs carrying data from the initiator to the responder are taken before SAs going in the reverse direction

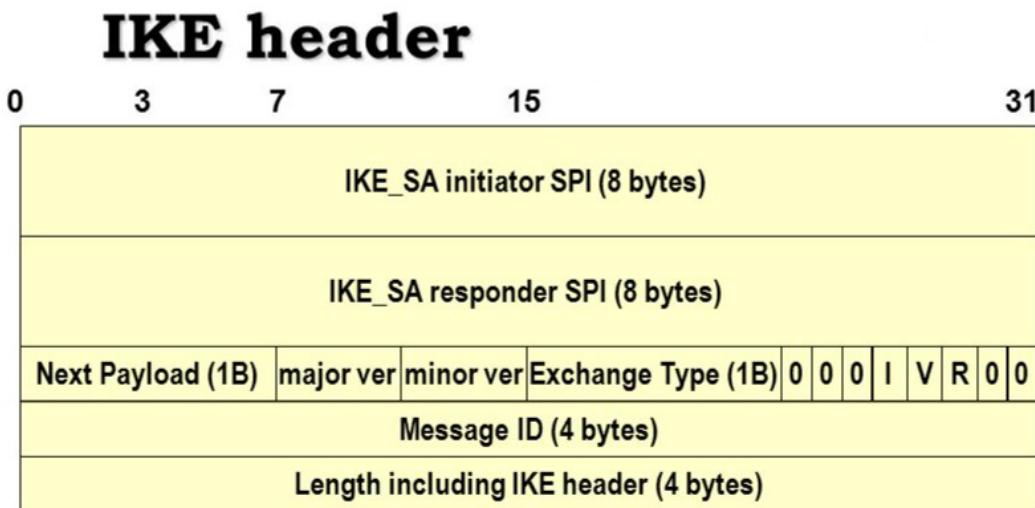
- If multiple IPsec protocols are negotiated, keying material is taken in the order in which the protocol headers will appear in the encapsulated packet
- If a single protocol has both encryption and authentication keys, the encryption key is taken from the first octets of KEYMAT and the authentication key is taken from the next octets

Each cryptographic algorithm takes a fixed number of bits of keying material specified as part of the algorithm

7.52.4 Header and Payload Formats

IKE Header

The format of the IKE header is shown in Figure 4



- **Initiator SPI, responder SPI:**
 - ⇒ SA id for the IKE protocol. Initially, responder SPI set to 0; responder fills it in
 - don't confuse with the SPI for the CHILD AH/ESP SA
- **Versions: major = 2, minor = 0**
 - ⇒ Bit V=1 says that implementation can "speak" higher version than what written in the hdr
- **Exchange type: one of 4 messages: IKE_SA_INIT, IKE_AUTH, CREATE_CHILD_SA, INFORMATIONAL**
 - ⇒ Direction: bit R (response=1, request=0); bit I (original initiator=1, original responder=0; needed to properly match SPIs)
- **Message ID: Sequence number counter, to match requests with responses, to manage retransmission, to combat replay attacks**

— Giuseppe Bianchi —

Figure 7.10: Figure 4: IKE Header Format [1]

- Initiator's SPI (8 octets) - A value chosen by the initiator to identify a unique IKE security association. This value MUST NOT be zero
- Responder's SPI (8 octets) - A value chosen by the responder to identify a unique IKE security association. This value MUST be zero in the first message of an IKE Initial Exchange (including repeats of that message including a cookie) and MUST NOT be zero in any other message
- Next Payload (1 octet) - Indicates the type of payload that immediately follows the header. The format and value of each payload are defined below
- Major Version (4 bits) - Indicates the major version of the IKE protocol in use. Implementations based on this version of IKE MUST set the Major Version to 2. Implementations based on previous versions of IKE and

ISAKMP MUST set the Major Version to 1. Implementations based on this version of IKE MUST reject or ignore messages containing a version number greater than 2

- Minor Version (4 bits) - Indicates the minor version of the IKE protocol in use. Implementations based on this version of IKE MUST set the Minor Version to 0. They MUST ignore the minor version number of received messages
- Exchange Type (1 octet) - Indicates the type of exchange being used. This constrains the payloads sent in each message and orderings of messages in an exchange

- Flags (1 octet) - Indicates specific options that are set for the message. Presence of options are indicated by the appropriate bit in the flags field being set. The bits are defined LSB first, so bit 0 would be the least significant bit of the Flags octet. In the description below, a bit being 'set' means its value is '1', while 'cleared' means its value is '0'
 - X(reserved) (bits 0-2) - These bits MUST be cleared when sending and MUST be ignored on receipt
 - I(initiator) (bit 3 of Flags) - This bit MUST be set in messages sent by the original initiator of the IKE-SA and MUST be cleared in messages sent by the original responder. It is used by the recipient to determine which eight octets of the SPI were generated by the recipient
 - V(ersion) (bit 4 of Flags) - This bit indicates that the transmitter is capable of speaking a higher major version number of the protocol than the one indicated in the major version number field. Implementations of IKEv2 must clear this bit when sending and MUST ignore it in incoming messages
 - R(esponse) (bit 5 of Flags) - This bit indicates that this message is a response to a message containing the same message ID. This bit MUST be cleared in all request messages and MUST be set in all responses. An IKE endpoint MUST NOT generate a response to a message that is marked as being a response
 - X(reserved) (bits 6-7 of Flags) - These bits MUST be cleared when sending and MUST be ignored on receipt
- Message ID (4 octets) - Message identifier used to control retransmission of lost packets and matching of requests and responses. It is essential to the security of the protocol because it is used to prevent message replay attacks
- Length (4 octets) - Length of total message (header + payloads) in octets

7.52.5 Generic Payload Header

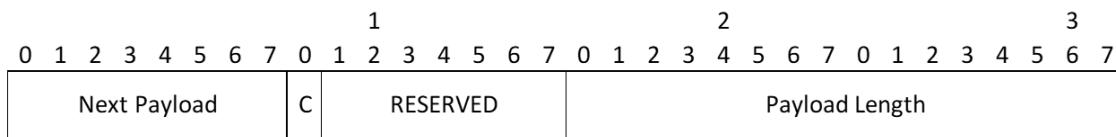


Figure 7.11: Generic Payload Header

- Next Payload - Identifier for the payload type of the next payload in the message. If the current payload is the last in the message, then this field will be 0. This field provides a "chaining" capability whereby additional payloads can be added to a message by appending each one to the end of the message and setting the Next Payload field of the preceding payload to indicate the new payload's type. An Encrypted payload, which must always be the last payload of a message, is an exception. It contains data structures in the format of additional payloads. In the header of an Encrypted payload, the Next Payload field is set to the payload type of the first contained payload (instead of 0); conversely, the Next Payload field of the last contained payload is set to zero. The payload type values are listed here. The values in the following table are only current as of the publication date of RFC 4306. Other values may have been added since then or will be added after the publication of this document. Readers should refer to IANA.org for the latest values

- Critical - MUST be set to zero if the sender wants the recipient to skip this payload if it does not understand the payload type code in the Next Payload field of the previous payload. MUST be set to one if the sender wants the recipient to reject this entire message if it does not understand the payload type. MUST be ignored by the recipient if the recipient understands the payload type code. MUST be set to zero for payload types defined in this document. Note that the critical bit applies to the current payload rather than the "next" payload whose type code appears in the first octet. The reasoning behind not setting the critical bit for payloads defined in this document is that all implementations MUST understand all payload types defined in this document and therefore must ignore the critical bit's value. Skipped payloads are expected to have valid Next Payload and Payload Length fields. See Section 2.5 for more information on this bit
- RESERVED - MUST be set to zero; MUST be ignored on receipt
- Payload Length - Length in octets of the current payload, including the generic payload header

7.52.6 Security Association Payload

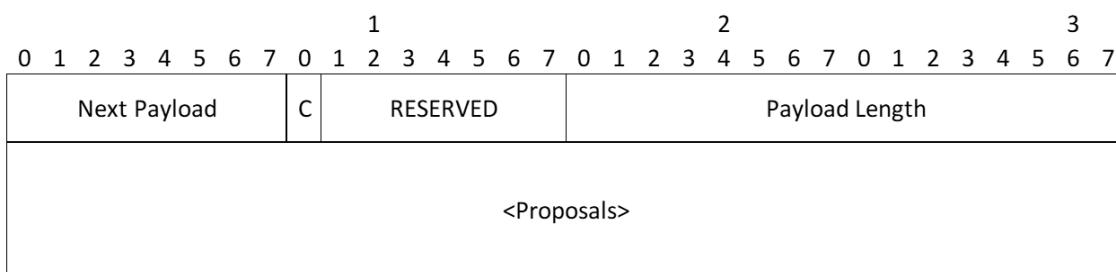


Figure 7.12: Security Association Payload

- Proposals - One or more proposal substructures

7.52.7 Proposal Substructure

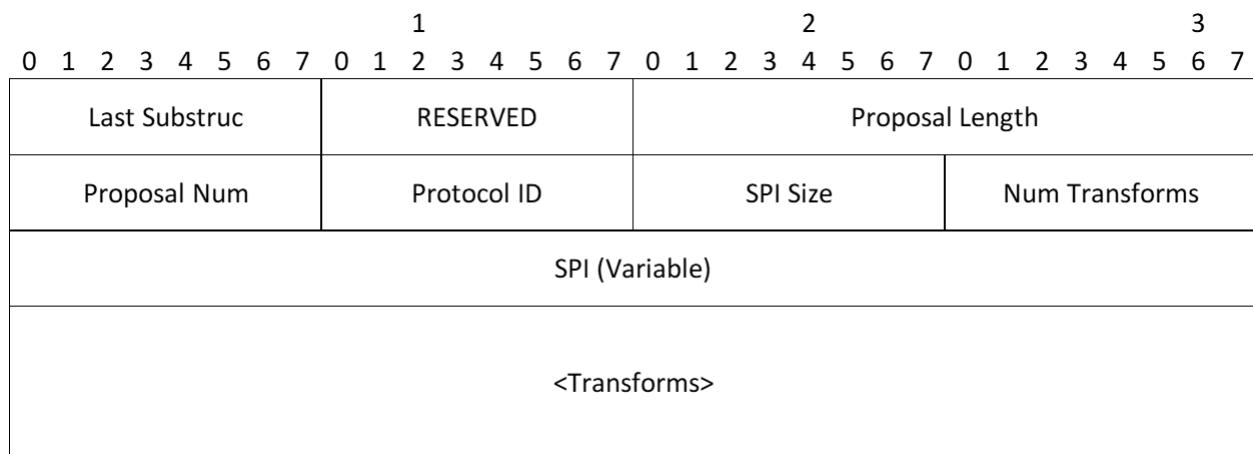


Figure 7.13: Proposal Substructure

- Last Substruc - Specifies whether or not this is the last Proposal Substructure in the SA. This field has a value of 0 if this was the last Proposal Substructure, and a value of 2 if there are more Proposal Substructures. This syntax is inherited from ISAKMP, but is unnecessary because the last Proposal could be identified from the length of the SA. The value (2) corresponds to a payload type of Proposal in IKEv1, and the first four octets of the Proposal structure are designed to look somewhat like the header of a payload

- RESERVED - MUST be sent as zero; MUST be ignored on receipt
 - Proposal Length - Length of this proposal, including all transforms and attributes that follow
 - Proposal Num - When a proposal is made, the first proposal in an SA payload MUST be 1, and subsequent proposals MUST be one more than the previous proposal (indicating an OR of the two proposals). When a proposal is accepted, the proposal number in the SA payload MUST match the number on the proposal sent that was accepted
 - Protocol ID - Specifies the IPsec protocol identifier for the current negotiation. The values in the following table are only current as of the publication date of RFC 4306. Other values may have been added since then or will be added after the publication of this document. Readers should refer to IANA.org for the latest values
 - SPI Size - For an initial IKE SA negotiation, this field MUST be zero; the SPI is obtained from the outer header. During subsequent negotiations, it is equal to the size, in octets, of the SPI of the corresponding protocol (8 for IKE, 4 for ESP and AH)
 - Num Transforms - Specifies the number of transforms in this proposal
 - SPI (variable) - The sending entity's SPI. Even if the SPI Size is not a multiple of 4 octets, there is no padding applied to the payload. When the SPI Size field is zero, this field is not present in the Security Association payload
 - Transforms (variable) - One or more transform substructures

7.52.8 Transform Substructure

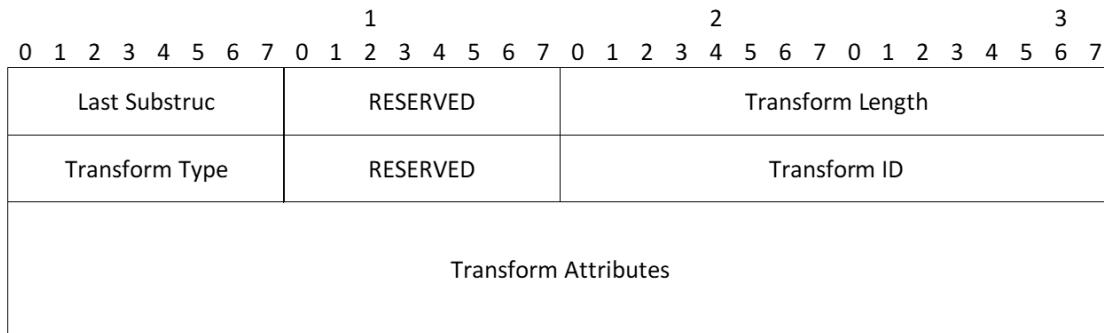


Figure 7.14: Transform Substructure

- Last Substruc - Specifies whether or not this is the last Transform Substructure in the Proposal. This field has a value of 0 if this was the last Transform Substructure, and a value of 3 if there are more Transform Substructures. This syntax is inherited from ISAKMP, but is unnecessary because the last transform could be identified from the length of the proposal. The value (3) corresponds to a payload type of Transform in IKEv1, and the first four octets of the Transform structure are designed to look somewhat like the header of a payload
 - RESERVED - MUST be sent as zero; MUST be ignored on receipt
 - Transform Length - The length (in octets) of the Transform Substructure including Header and Attributes
 - Transform Type - The type of transform being specified in this transform. Different protocols support different Transform Types. For some protocols, some of the transforms may be optional. If a transform is optional and the initiator wishes to propose that the transform be omitted, no transform of the given type is included in the proposal. If the initiator wishes to make use of the transform optional to the responder, it includes a transform substructure with Transform ID = 0 as one of the options
 - Transform ID - The specific instance of the Transform Type being proposed see IANA.org for all values

7.52.9 Transform Attributes

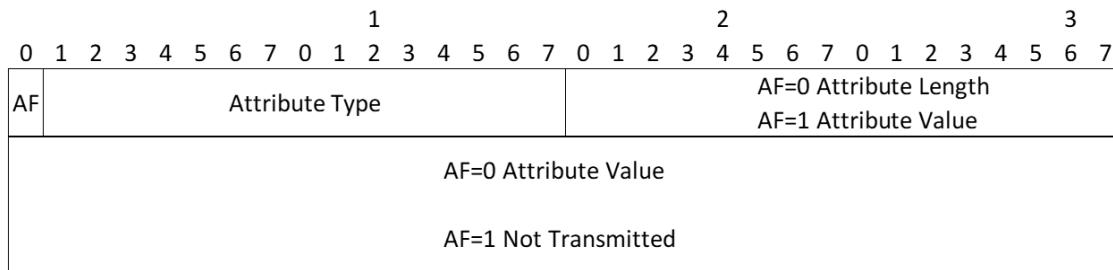


Figure 7.15: Transform Attributes

- Attribute Format (AF) - Indicates whether the data attribute follows the Type/Length/Value (TLV) format or a shortened Type/Value (TV) format. If the AF bit is zero (0), then the attribute uses TLV format; if the AF bit is one (1), the TV format (with two-byte value) is used.
- Attribute Type - Unique identifier for each type of attribute
- Attribute Value (variable length) - Value of the attribute associated with the attribute type. If the AF bit is a zero (0), this field has a variable length defined by the Attribute Length field. If the AF bit is a one (1), the Attribute Value has a length of 2 octets

The only currently defined attribute type (Key Length) is fixed length; the variable-length encoding specification is included only for future extensions. Attributes described as fixed length MUST NOT be encoded using the variable-length encoding unless that length exceeds two bytes. Variable-length attributes MUST NOT be encoded as fixed-length even if their value can fit into two octets. Note: This is a change from IKEv1, where increased flexibility may have simplified the composer of messages but certainly complicated the parser

The Key Length attribute specifies the key length in bits (MUST use network byte order) for certain transforms as follows:

- The Key Length attribute MUST NOT be used with transforms that use a fixed-length key. For example, this includes ENCR_DES, ENCR_IDEA, and all the Type 2 (Pseudorandom Function) and Type 3 (Integrity Algorithm) transforms specified in this document. It is recommended that future Type 2 or 3 transforms do not use this attribute
- Some transforms specify that the Key Length attribute MUST be always included (omitting the attribute is not allowed, and proposals not containing it MUST be rejected). For example, this includes ENCR_AES_CBC and ENCR_AES_CTR
- Some transforms allow variable-length keys, but also specify a default key length if the attribute is not included. For example, these transforms include ENCR_RC5 and ENCR_BLOWFISH

7.52.10 Key Exchange Payload

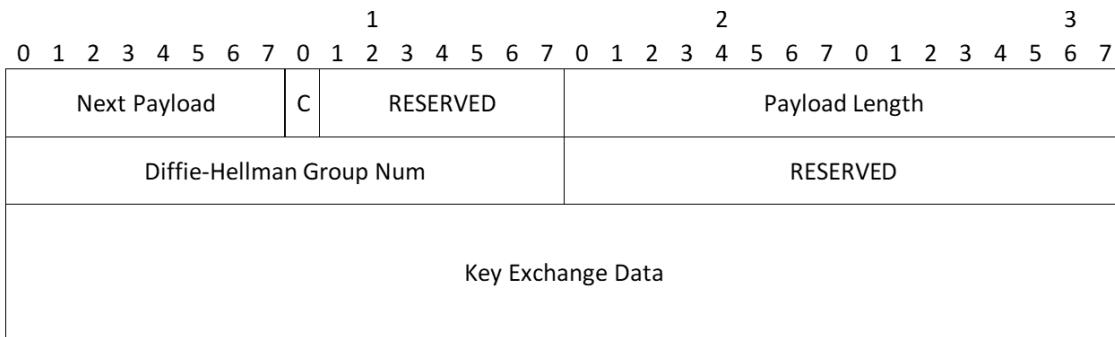


Figure 7.16: Key Exchange

A Key Exchange payload is constructed by copying one's Diffie-Hellman public value into the "Key Exchange Data" portion of the payload. The length of the Diffie-Hellman public value for MODP groups MUST be equal to the length of the prime modulus over which the exponentiation was performed, prepending zero bits to the value if necessary

7.52.11 Identification Payload

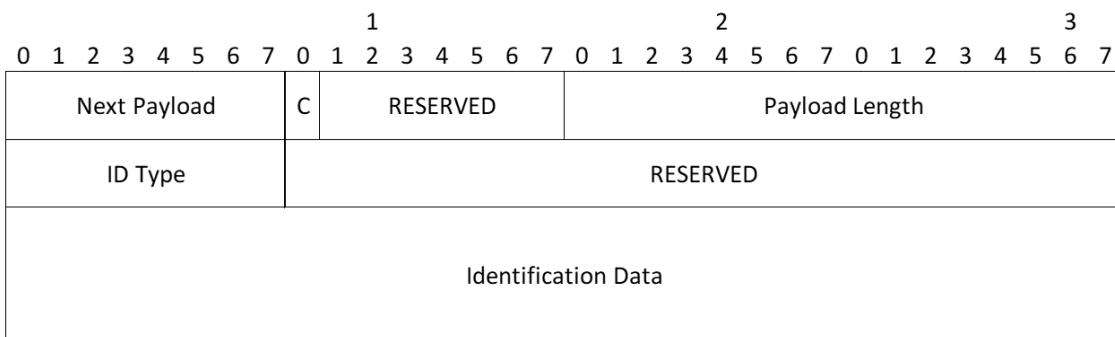


Figure 7.17: Identification Payload

- ID Type - Specifies the type of Identification being used
- RESERVED - MUST be sent as zero; MUST be ignored on receipt
- Identification Data (variable length) - Value, as indicated by the Identification Type. The length of the Identification Data is computed from the size in the ID payload header

7.52.12 Certification Payload

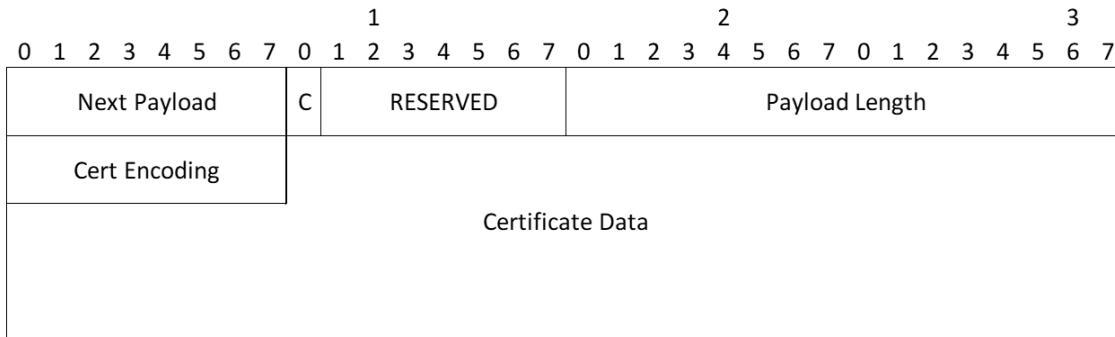


Figure 7.18: Certification Payload

- Certificate Encoding - This field indicates the type of certificate or certificate-related information contained in the Certificate Data field. The values in the following table are only current as of the publication date of RFC 4306. Other values may have been added since then or will be added after the publication of this document. Readers should refer to IANA.org for the latest values
- Certificate Data (variable length) - Actual encoding of certificate data. The type of certificate is indicated by the Certificate Encoding field

7.52.13 Certification Request Payload



Figure 7.19: Certification Request Payload

- Certificate Encoding - Contains an encoding of the type or format of certificate requested. Values are listed in Section 3.6
- Certification Authority (variable length) - Contains an encoding of an acceptable certification authority for the type of certificate requested

7.52.14 Authentication Payload

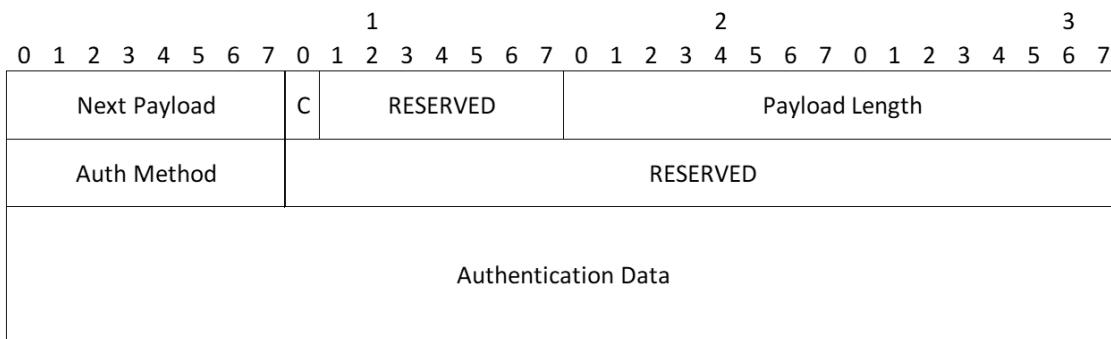


Figure 7.20: Authentication Payload

- Auth Method - Specifies the method of authentication used. The types of signatures are listed here. The values in the following table are only current as of the publication date of RFC 4306. Other values may have been added since then or will be added after the publication of this document. Readers should refer to IANA.org for the latest values
 - RSA Digital Signature - Computed as specified in Section 2.15 using an RSA private key with RSASSA-PKCS1-v1_5 signature scheme specified in [PKCS1] (implementers should note that IKEv1 used a different method for RSA signatures). To promote interoperability, implementations that support this type SHOULD support signatures that use SHA-1 as the hash function and SHOULD use SHA-1 as the default hash function when generating signatures. Implementations can use the certificates received from a given peer as a hint for selecting a mutually understood hash function for the AUTH payload signature. Note, however, that the hash algorithm used in the AUTH payload signature doesn't have to be the same as any hash algorithm(s) used in the certificate(s)
 - Shared Key Message Integrity Code - Computed as specified in Section 2.15 using the shared key associated with the identity in the ID payload and the negotiated PRF
 - DSS Digital Signature - Computed as specified in Section 2.15 using a DSS private key (see [DSS]) over a SHA-1 hash
 - Elliptic Curve DSA (ECDSA) - Computed as specified in Section 2.15 encoding of the computed signature consisting of the concatenation of a pair of integers r and s. The definitions of r and s are given in Section 8 RFC4754
 - Digital Signature - Computer as specified in Section 2.15 with authentication method to support signature methods in a more general way. The current signature-based authentication methods in IKEv2 are per algorithm, i.e., there is one for RSA digital signatures, one for DSS digital signatures (using SHA-1), and three for different ECDSA curves, each tied to exactly one hash algorithm. This design is cumbersome when more signature algorithms, hash algorithms, and elliptic curves need to be supported:
 - In IKEv2, authentication using RSA digital signatures calls for padding based on RSASSA-PKCS1-v1_5, although the newer RSASSA-PSS padding method is now recommended. (See Section 5 of "Additional Algorithms and Identifiers for RSA Cryptography for use in PKIX Profile" [RFC4055].)
 - With ECDSA and the Digital Signature Standard (DSS), there is no way to extract the hash algorithm from the signature. Thus, for each new hash function to be supported with ECDSA or DSA, new authentication methods would be needed. Support for new hash functions is particularly needed for DSS, because the current restriction to SHA-1 limits its security, meaning there is no point of using long keys with SHA-1
 - The tying of ECDSA authentication methods to particular elliptic curve groups requires definition of additional methods for each new group. The combination of new ECDSA groups and hash functions will cause the number of required authentication methods to become unmanageable. Furthermore, the restriction of ECDSA authentication to a specific group is inconsistent with the approach taken with DSS
 - RESERVED - MUST be sent as zero; MUST be ignored on receipt

- Authentication Data (variable length) - see Section 2.15

7.52.15 Nonce Payload

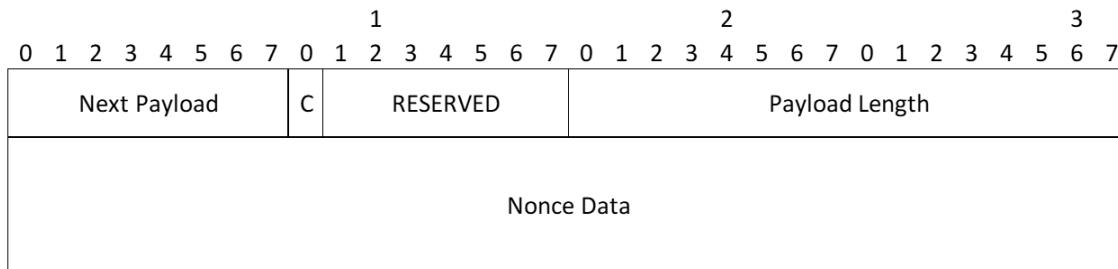


Figure 7.21: Nonce Payload

- Nonce Data (variable length) - Contains the random data generated by the transmitting entity

7.52.16 Notify Payload

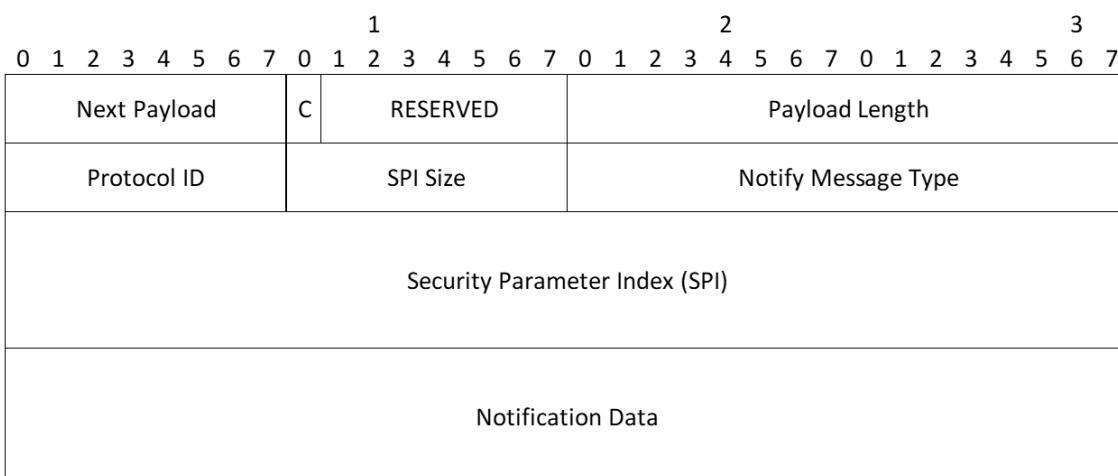


Figure 7.22: Notify Payload

- Protocol ID - If this notification concerns an existing SA whose SPI is given in the SPI field, this field indicates the type of that SA. For notifications concerning Child SAs, this field MUST contain either (2) to indicate AH or (3) to indicate ESP. Of the notifications defined in this document, the SPI is included only with INVALID_SELECTORS, REKEY_SA, and CHILD_SA_NOT_FOUND. If the SPI field is empty, this field MUST be sent as zero and MUST be ignored on receipt
- SPI Size - Length in octets of the SPI as defined by the IPsec protocol ID or zero if no SPI is applicable. For a notification concerning the IKE SA, the SPI Size MUST be zero and the field must be empty
- Notify Message Type - Specifies the type of notification message
- SPI (variable length) - Security Parameter Index
- Notification Data (variable length) - Status or error data transmitted in addition to the Notify Message Type. Values for this field are type specific (see below)

7.52.17 Delete Payload

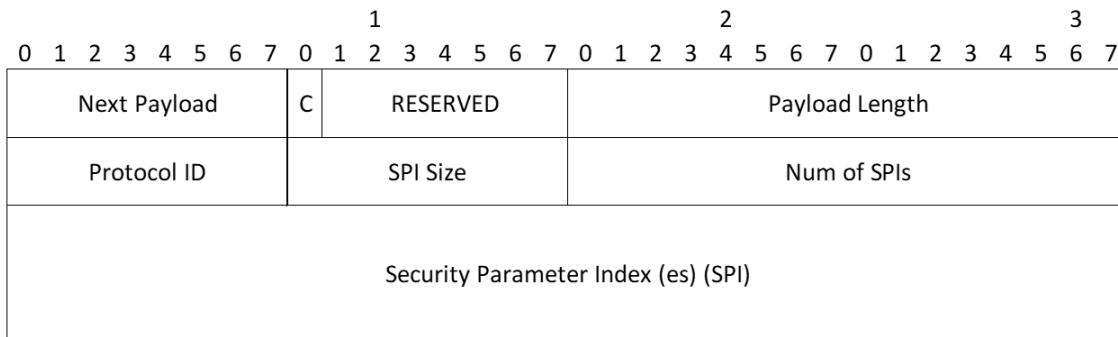


Figure 7.23: Delete Payload

- Protocol ID - Must be 1 for an IKE SA, 2 for AH, or 3 for ESP
- SPI Size - Length in octets of the SPI as defined by the protocol ID. It MUST be zero for IKE (SPI is in message header) or four for AH and ESP
- Num of SPIs (unsigned integer) - The number of SPIs contained in the Delete payload. The size of each SPI is defined by the SPI Size field
- Security Parameter Index(es) (variable length) - Identifies the specific Security Association(s) to delete. The length of this field is determined by the SPI Size and Num of SPIs fields

7.52.18 Vendor ID Payload

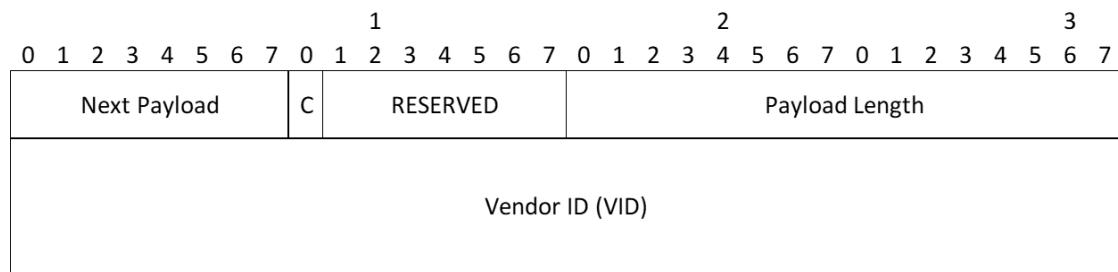


Figure 7.24: Vendor ID Payload

- Vendor ID (variable length) - It is the responsibility of the person choosing the Vendor ID to assure its uniqueness in spite of the absence of any central registry for IDs. Good practice is to include a company name, a person name, or some such information. If you want to show off, you might include the latitude and longitude and time where you were when you chose the ID and some random input. A message digest of a long unique string is preferable to the long unique string itself

7.52.19 Traffic Selectors Payload



Figure 7.25: Traffic Selectors Payload

- Number of TSs - Number of Traffic Selectors being provided
- RESERVED - This field MUST be sent as zero and MUST be ignored on receipt
- Traffic Selectors (variable length) - One or more individual Traffic Selectors

7.52.20 Traffic Selector

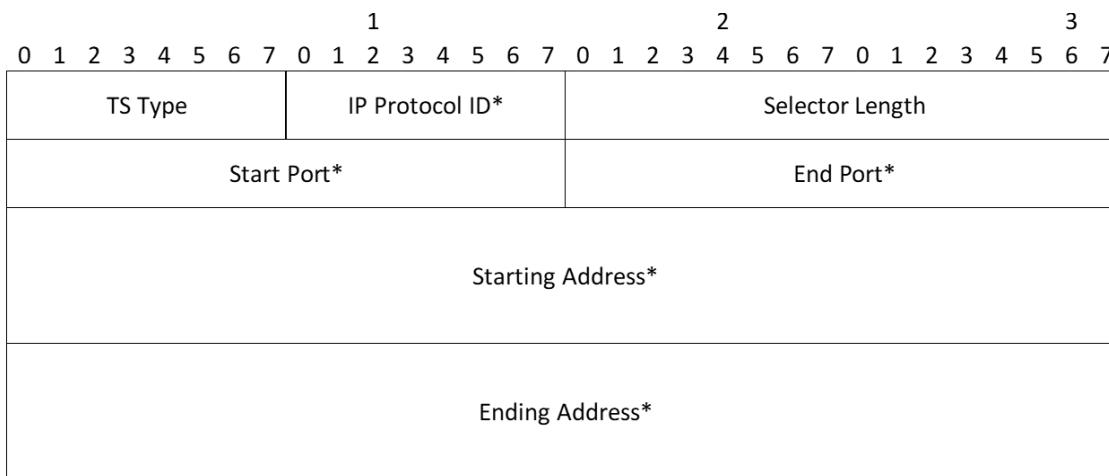


Figure 7.26: Traffic Selector

- TS Type - Specifies the type of Traffic Selector
 - TS_IPV4_ADDR_RANGE - A range of IPv4 addresses, represented by two four-octet values. The first value is the beginning IPv4 address (inclusive) and the second value is the ending IPv4 address (inclusive). All addresses falling between the two specified addresses are considered to be within the list
 - TS_IPV6_ADDR_RANGE - A range of IPv6 addresses, represented by two sixteen-octet values. The first value is the beginning IPv6 address (inclusive) and the second value is the ending IPv6 address (inclusive). All addresses falling between the two specified addresses are considered to be within the list
- IP protocol ID - Value specifying an associated IP protocol ID (such as UDP, TCP, and ICMP). A value of zero means that the protocol ID is not relevant to this Traffic Selector - the SA can carry all protocols

- Selector Length - Specifies the length of this Traffic Selector substructure including the header
- Start Port - Value specifying the smallest port number allowed by this Traffic Selector. For protocols for which port is undefined (including protocol 0), or if all ports are allowed, this field MUST be zero. ICMP and ICMPv6 Type and Code values, as well as Mobile IP version 6 (MIPv6) mobility header (MH) Type values, are represented in this field as specified in Section 4.4.1.1 of [IPSESEARCH]. ICMP Type and Code values are treated as a single 16-bit integer port number, with Type in the most significant eight bits and Code in the least significant eight bits. MIPv6 MH Type values are treated as a single 16-bit integer port number, with Type in the most significant eight bits and the least significant eight bits set to zero
- End Port - Value specifying the largest port number allowed by this Traffic Selector. For protocols for which port is undefined (including protocol 0), or if all ports are allowed, this field MUST be 65535. ICMP and ICMPv6 Type and Code values, as well as MIPv6 MH Type values, are represented in this field as specified in Section 4.4.1.1 of [IPSESEARCH]. ICMP Type and Code values are treated as a single 16-bit integer port number, with Type in the most significant eight bits and Code in the least significant eight bits. MIPv6 MH Type values are treated as a single 16-bit integer port number, with Type in the most significant eight bits and the least significant eight bits set to zero
- Starting Address - The smallest address included in this Traffic Selector (length determined by TS Type)
- Ending Address - The largest address included in this Traffic Selector (length determined by TS Type)

7.52.21 Encrypted Payload Format

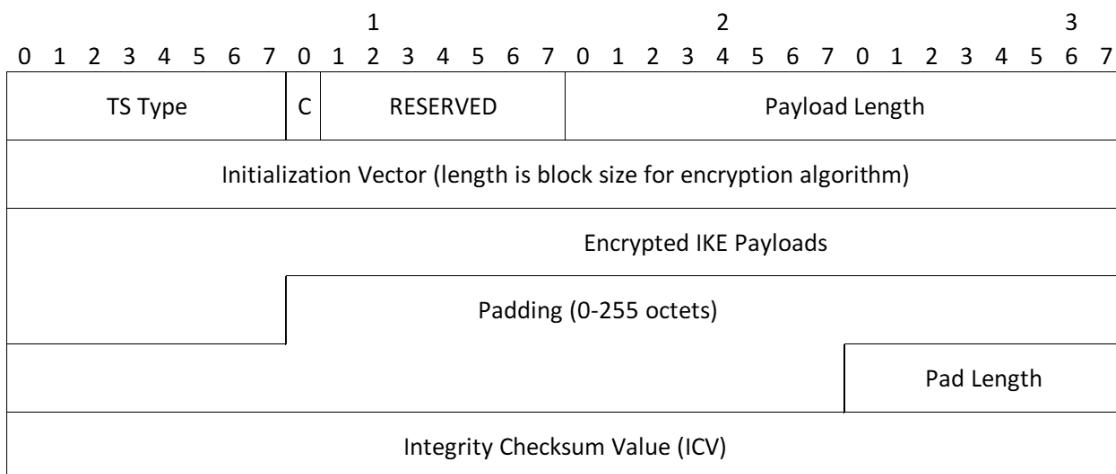


Figure 7.27: Encrypted Payload Format

- Next Payload - The payload type of the first embedded payload. Note that this is an exception in the standard header format, since the Encrypted payload is the last payload in the message and therefore the Next Payload field would normally be zero. But because the content of this payload is embedded payloads and there was no natural place to put the type of the first one, that type is placed here
- Payload Length - Includes the lengths of the header, initialization vector (IV), Encrypted IKE payloads, Padding, Pad Length, and Integrity Checksum Data
- Initialization Vector - For CBC mode ciphers, the length of the initialization vector (IV) is equal to the block length of the underlying encryption algorithm. Senders MUST select a new unpredictable IV for every message; recipients MUST accept any value. The reader is encouraged to consult [MODES] for advice on IV generation. In particular, using the final ciphertext block of the previous message is not considered unpredictable. For modes other than CBC, the IV format and processing is specified in the document specifying the encryption algorithm and mode
- IKE payloads are as specified earlier in this section. This field is encrypted with the negotiated cipher

- Padding MAY contain any value chosen by the sender, and MUST have a length that makes the combination of the payloads, the Padding, and the Pad Length to be a multiple of the encryption block size. This field is encrypted with the negotiated cipher
- Pad Length is the length of the Padding field. The sender SHOULD set the Pad Length to the minimum value that makes the combination of the payloads, the Padding, and the Pad Length a multiple of the block size, but the recipient MUST accept any length that results in proper alignment. This field is encrypted with the negotiated cipher
- Integrity Checksum Data is the cryptographic checksum of the entire message starting with the Fixed IKE header through the Pad Length. The checksum MUST be computed over the encrypted message. Its length is determined by the integrity algorithm negotiated

7.52.22 Configuration Payload

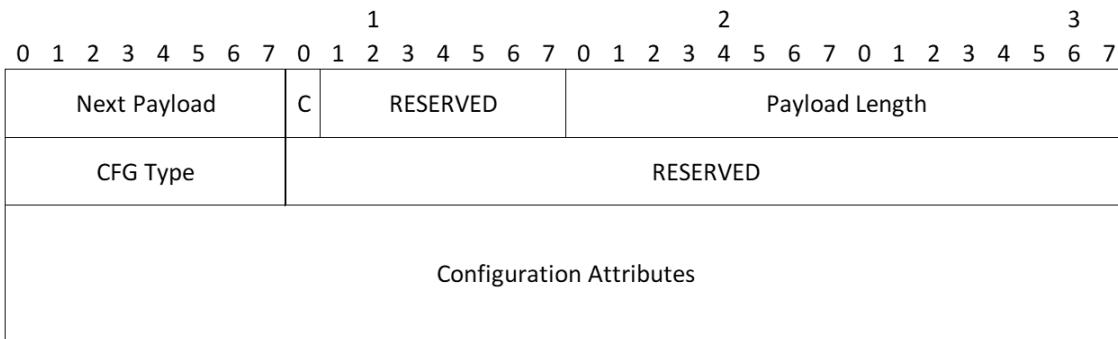


Figure 7.28: Configuration Payload

- CFG Type - The type of exchange represented by the Configuration Attributes. The values in the following table are only current as of the publication date of RFC 4306. Other values may have been added since then or will be added after the publication of this document. Readers should refer to IANA.org for the latest values
 - CFG_REQUEST - 1
 - CFG_REPLY - 2
 - CFG_SET - 3
 - CFG_ACK - 4
- RESERVED - MUST be sent as zero; MUST be ignored on receipt
- Configuration Attributes (variable length) - These are type length value (TLV) structures specific to the Configuration payload and are defined below. There may be zero or more Configuration Attributes in this payload

7.52.23 Configuration Attributes

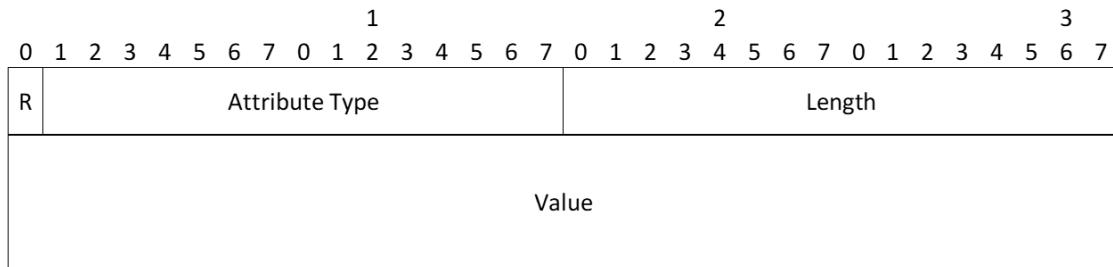


Figure 7.29: Configuration Attribute Format

- Reserved - This bit MUST be set to zero and MUST be ignored on receipt
 - Attribute Type - A unique identifier for each of the Configuration Attribute Types
 - Length - Length in octets of value
 - Value (0 or more octets) - The variable-length value of this Configuration Attribute. The following lists the attribute types

7.52.24 EAP Payload

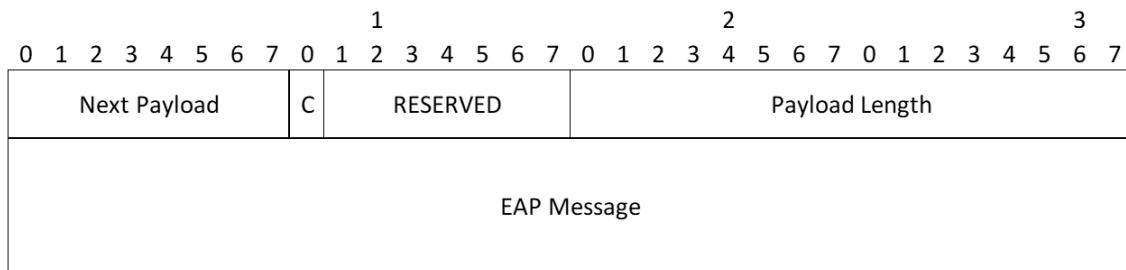


Figure 7.30: EAP Payload Format

7.52.25 EAP Message

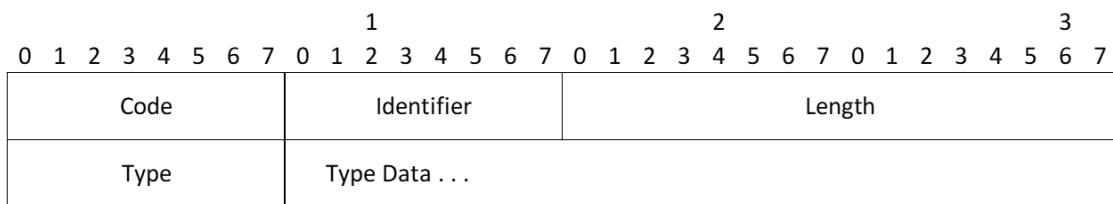


Figure 7.31: EAP Message Format

- Code - Indicates whether this message is a Request (1), Response (2), Success (3), or Failure (4)

- Identifier - Used in PPP to distinguish replayed messages from repeated ones. Since in IKE, EAP runs over a reliable protocol, the Identifier serves no function here. In a response message, this octet MUST be set to match the identifier in the corresponding request

- Length - The length of the EAP message. MUST be four less than the Payload Length of the encapsulating payload

- Type - Present only if the Code field is Request (1) or Response (2). For other codes, the EAP message length MUST be four octets and the Type and Type_Data fields MUST NOT be present. In a Request (1) message, Type indicates the data being requested. In a Response (2) message, Type MUST either be Nak or match the type of the data requested. Note that since IKE passes an indication of initiator identity in the first message in the IKE_AUTH exchange, the responder SHOULD NOT send EAP Identity requests (type 1). The initiator MAY, however, respond to such requests if it receives them

- Type_Data (variable length) - Varies with the Type of Request and the associated Response. For the documentation of the EAP methods

7.52.26 IKEv2 State Diagram

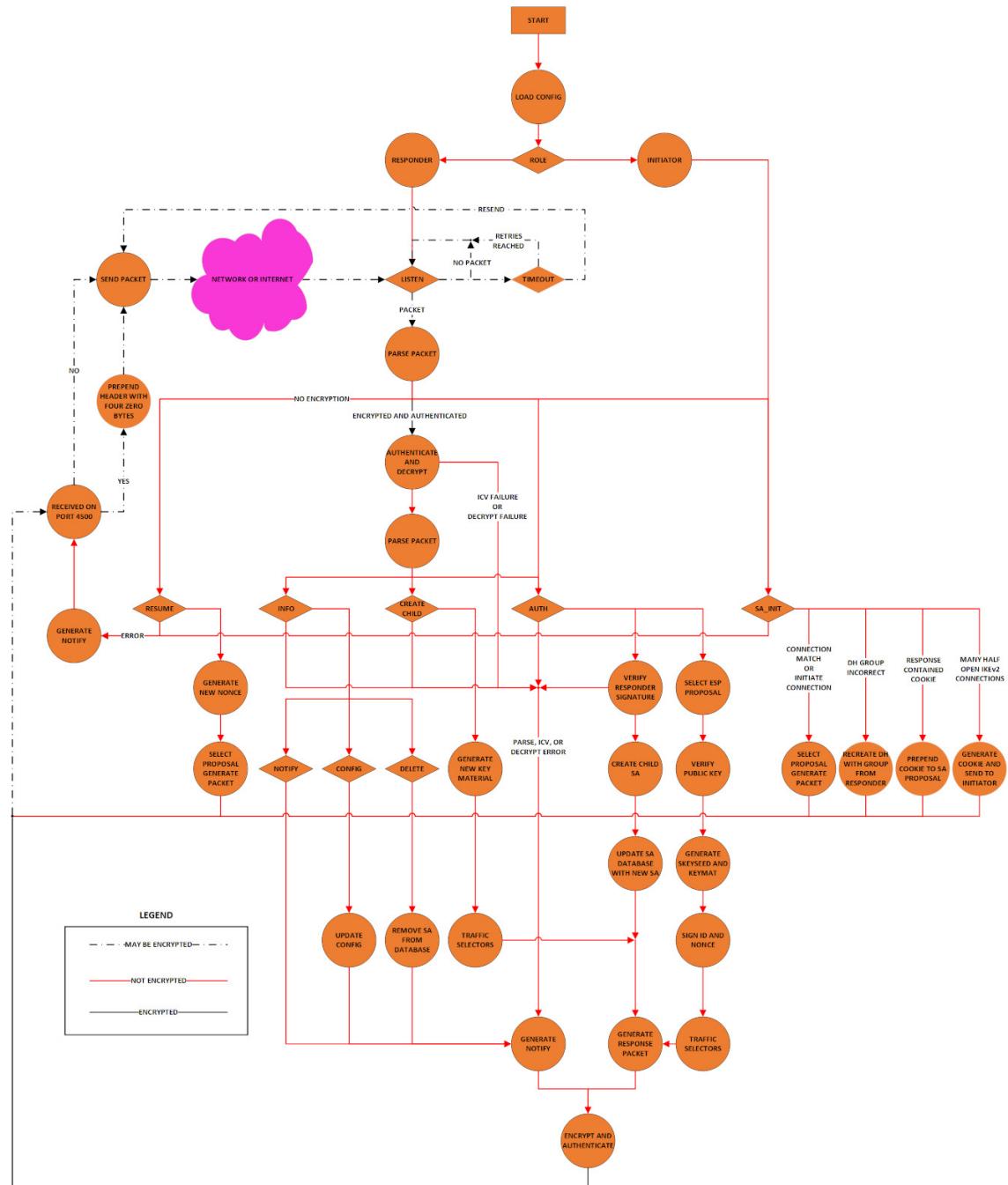


Figure 7.32: IKEv2 State Diagram

For API information:

See Also

[ProtocolPP::jikev2](#)
[ProtocolPP::jrand](#)
[ProtocolPP::jenum](#)
[ProtocolPP::jsecass](#)
[ProtocolPP::jikev2sa](#)
[ProtocolPP::jipsecsa](#)

[ProtocolPP::jdsa](#)
[ProtocolPP::jecdsa](#)
[ProtocolPP::jrsa](#)

For Additional Documentation:

See Also

[jikev2](#)
[jrand](#)
[jenum](#)
[jsecass](#)
[jikev2sa](#)
[jipsecsa](#)
[jdsa](#)
[jecdsa](#)
[jrsa](#)

[1] Giuseppe Bianchi CSE

Protocol++ written by : John Peter Greninger © John Peter Greninger 2015-2019 All Rights Reserved

All copyrights and trademarks are the property of their respective owners

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer
- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution
- Any and all modifications must be returned to John Peter Greninger at GitHub.com <https://github.com/jpgreninger/protocolpp> for evaluation. Inclusion of modifications in the source code shall be determined solely by John Peter Greninger. Failure to provide modifications shall render this license NULL and VOID and revoke any rights to use of Protocol++
- Commercial use requires a fee-based license obtainable at www.protocolpp.com
- Academic use requires written and notarized permission from John Peter and Sheila Rocha Greninger
- US Copyrights at <https://www.copyright.gov/>
 - TXu002059872 (Version 1.0.0)
 - TXu002066632 (Version 1.2.7)
 - TXu002082674 (Version 1.4.0)
 - TXu002097880 (Version 2.0.0)

The names of its contributors may not be used to endorse or promote products derived from this software without specific prior written permission

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE

The documentation for this class was generated from the following file:

- `jikev2/include/jikev2.h`

7.53 ProtocolPP::jikev2dh Class Reference

```
#include <jikev2dh.h>
```

Public Member Functions

- `jikev2dh (std::shared_ptr< InterfacePP::jlogger > &logger)`
Standard constructor.
- `~jikev2dh ()`
Standard deconstructor.
- `bool get_pubkey (dh_id_t diffiehellman, std::shared_ptr< jarray< uint8_t >> &pubkey, std::shared_ptr< CryptoPP::SecByteBlock > &prvKey)`
- `bool verify (dh_id_t diffiehellman, std::shared_ptr< jarray< uint8_t >> &rcvPubKey, std::shared_ptr< CryptoPP::SecByteBlock > &prvKey, std::shared_ptr< jarray< uint8_t >> &sharedSecret)`

7.53.1 Constructor & Destructor Documentation

7.53.1.1 `ProtocolPP::jikev2dh::jikev2dh (std::shared_ptr< InterfacePP::jlogger > & logger)`

Standard constructor.

7.53.1.2 `ProtocolPP::jikev2dh::~jikev2dh () [inline]`

Standard deconstructor.

7.53.2 Member Function Documentation

7.53.2.1 `bool ProtocolPP::jikev2dh::get_pubkey (dh_id_t diffiehellman, std::shared_ptr< jarray< uint8_t >> & pubkey, std::shared_ptr< CryptoPP::SecByteBlock > & prvKey)`

Generate a public key pair

Parameters

<code>diffiehellman</code>	- Diffie-Hellman curve or field to use
<code>pubkey</code>	- array containing public key
<code>prvKey</code>	- Generated private key from key pair

Returns

- True if found and calculated, false otherwise

7.53.2.2 `bool ProtocolPP::jikev2dh::verify (dh_id_t diffiehellman, std::shared_ptr< jarray< uint8_t >> & rcvPubKey, std::shared_ptr< CryptoPP::SecByteBlock > & prvKey, std::shared_ptr< jarray< uint8_t >> & sharedSecret)`

Verify the Diffie-Hellman computation and generate shared secret

Parameters

<i>diffiehellman</i>	- Diffie-Hellman curve or field to use
<i>rcvPubKey</i>	- the received public key
<i>prvKey</i>	- Generated private key from key pair
<i>sharedSecret</i>	- the generated shared secret if verification is successful

Returns

Public key verified

The documentation for this class was generated from the following file:

- jikev2/include/jikev2dh.h

7.54 jikev2dh Class Reference

```
#include "include/jikev2dh.h"
```

7.54.1 Detailed Description

7.54.2 Diffie-Hellman Support for IKEv2

(See RFC7296 for complete details)

IKE performs mutual authentication between two parties and establishes an IKE security association (SA) that includes shared secret information that can be used to efficiently establish SAs for Encapsulating Security Payload (ESP) [RFC4303] and/or Authentication Header (AH) [RFC4302] and a set of cryptographic algorithms to be used by the SAs to protect the traffic that they carry. In this document, the term "suite" or "cryptographic suite" refers to a complete set of algorithms used to protect an SA. An initiator proposes one or more suites by listing supported algorithms that can be combined into suites in a mix-and-match fashion. IKE can also negotiate use of IP Compression (IPComp) [IPCOMP] in connection with an ESP and/or AH SA. We call the IKE SA an "IKE_SA". The SAs for ESP and/or AH that get set up through that IKE_SA we call "CHILD_SAs".

All IKE communications consist of pairs of messages: a request and a response. The pair is called an "exchange". We call the first messages establishing an IKE_SA IKE_SA_INIT and IKE_AUTH exchanges and subsequent IKE exchanges CREATE_CHILD_SA or INFORMATIONAL exchanges. In the common case, there is a single IKE_SA_INIT exchange and a single IKE_AUTH exchange (a total of four messages) to establish the IKE_SA and the first CHILD_SA. In exceptional cases, there may be more than one of each of these exchanges. In all cases, all IKE_SA_INIT exchanges MUST complete before any other exchange type, then all IKE_AUTH exchanges MUST complete, and following that any number of CREATE_CHILD_SA and INFORMATIONAL exchanges may occur in any order. In some scenarios, only a single CHILD_SA is needed between the IPsec endpoints, and therefore there would be no additional exchanges. Subsequent exchanges MAY be used to establish additional CHILD_SAs between the same authenticated pair of endpoints and to perform housekeeping functions.

IKE message flow always consists of a request followed by a response. It is the responsibility of the requester to ensure reliability. If the response is not received within a timeout interval, the requester needs to retransmit the request (or abandon the connection).

The first request/response of an IKE session (IKE_SA_INIT) negotiates security parameters for the IKE_SA, sends nonces, and sends Diffie-Hellman values. The second request/response (IKE_AUTH) transmits identities, proves knowledge of the secrets corresponding to the two identities, and sets up an SA for the first (and often only) AH and/or ESP CHILD_SA. The types of subsequent exchanges are CREATE_CHILD_SA (which creates a CHILD_SA) and INFORMATIONAL (which deletes an SA, reports error conditions, or does other housekeeping). Every request requires a response. An INFORMATIONAL request with no payloads (other than the empty Encrypted payload required by the syntax) is commonly used as a check for liveness. These subsequent exchanges cannot be used until the initial exchanges have completed.

For API information:

See Also

[ProtocolPP::jikev2dh](#)
[ProtocolPP::jikev2](#)
[ProtocolPP::jikev2sa](#)
[ProtocolPP::jikeparse](#)
[ProtocolPP::jenum](#)

For Additional Documentation:**See Also**

[jikev2dh](#)
[jikev2](#)
[jikev2sa](#)
[jikevparse](#)
[jenum](#)

Protocol++ written by : John Peter Greninger © John Peter Greninger 2015-2019 All Rights Reserved

All copyrights and trademarks are the property of their respective owners

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer
- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution
- Any and all modifications must be returned to John Peter Greninger at GitHub.com <https://github.com/jpgreninger/protocolpp> for evaluation. Inclusion of modifications in the source code shall be determined solely by John Peter Greninger. Failure to provide modifications shall render this license NULL and VOID and revoke any rights to use of Protocol++
- Commercial use requires a fee-based license obtainable at www.protocolpp.com
- Academic use requires written and notarized permission from John Peter and Sheila Rocha Greninger
- US Copyrights at [https://www.copyright.gov/](http://www.copyright.gov/)
 - TXu002059872 (Version 1.0.0)
 - TXu002066632 (Version 1.2.7)
 - TXu002082674 (Version 1.4.0)
 - TXu002097880 (Version 2.0.0)

The names of its contributors may not be used to endorse or promote products derived from this software without specific prior written permission

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE

The documentation for this class was generated from the following file:

- jikev2/include/[jikev2dh.h](#)

7.55 jikev2sa Class Reference

```
#include "include/jikev2sa.h"
```

7.55.1 Detailed Description

7.55.2 Internet Key Exchange (IKEv2) Security Association

(See RFC7296 for complete details)

IKE performs mutual authentication between two parties and establishes an IKE security association (SA) that includes shared secret information that can be used to efficiently establish SAs for Encapsulating Security Payload (ESP) [RFC4303] and/or Authentication Header (AH) [RFC4302] and a set of cryptographic algorithms to be used by the SAs to protect the traffic that they carry. In this document, the term "suite" or "cryptographic suite" refers to a complete set of algorithms used to protect an SA. An initiator proposes one or more suites by listing supported algorithms that can be combined into suites in a mix-and-match fashion. IKE can also negotiate use of IP Compression (IPComp) [IPCOMP] in connection with an ESP and/or AH SA. We call the IKE SA an "IKE_SA". The SAs for ESP and/or AH that get set up through that IKE_SA we call "CHILD_SAs"

All IKE communications consist of pairs of messages: a request and a response. The pair is called an "exchange". We call the first messages establishing an IKE_SA IKE_SA_INIT and IKE_AUTH exchanges and subsequent IKE exchanges CREATE_CHILD_SA or INFORMATIONAL exchanges. In the common case, there is a single IKE_SA_INIT exchange and a single IKE_AUTH exchange (a total of four messages) to establish the IKE_SA and the first CHILD_SA. In exceptional cases, there may be more than one of each of these exchanges. In all cases, all IKE_SA_INIT exchanges MUST complete before any other exchange type, then all IKE_AUTH exchanges MUST complete, and following that any number of CREATE_CHILD_SA and INFORMATIONAL exchanges may occur in any order. In some scenarios, only a single CHILD_SA is needed between the IPsec endpoints, and therefore there would be no additional exchanges. Subsequent exchanges MAY be used to establish additional CHILD_SAs between the same authenticated pair of endpoints and to perform housekeeping functions

IKE message flow always consists of a request followed by a response. It is the responsibility of the requester to ensure reliability. If the response is not received within a timeout interval, the requester needs to retransmit the request (or abandon the connection)

The first request/response of an IKE session (IKE_SA_INIT) negotiates security parameters for the IKE_SA, sends nonces, and sends Diffie-Hellman values. The second request/response (IKE_AUTH) transmits identities, proves knowledge of the secrets corresponding to the two identities, and sets up an SA for the first (and often only) AH and/or ESP CHILD_SA. The types of subsequent exchanges are CREATE_CHILD_SA (which creates a CHILD_SA) and INFORMATIONAL (which deletes an SA, reports error conditions, or does other housekeeping). Every request requires a response. An INFORMATIONAL request with no payloads (other than the empty Encrypted payload required by the syntax) is commonly used as a check for liveness. These subsequent exchanges cannot be used until the initial exchanges have completed

7.55.3 Scenarios

Security Gateway to Security Gateway Tunnel

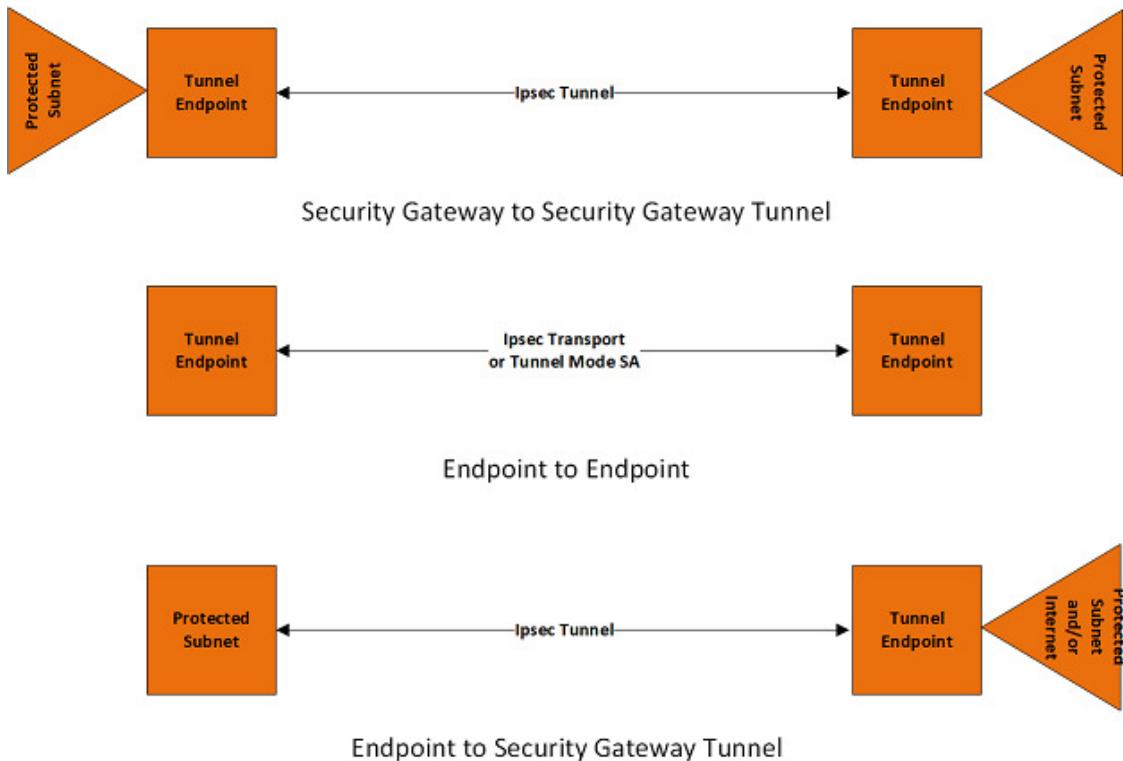
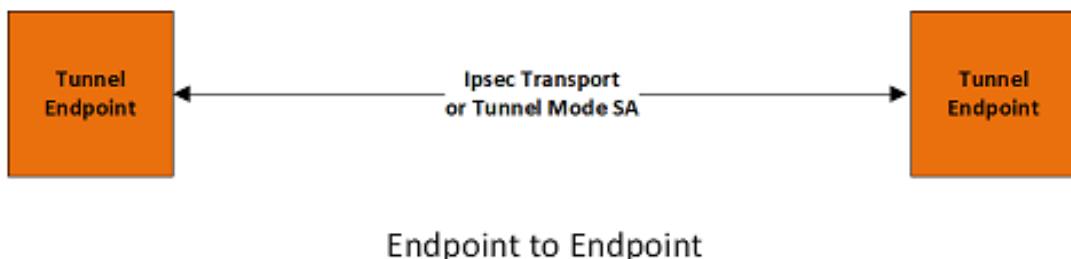


Figure 7.33: Security Gateway to Security Gateway Tunnel

In this scenario, neither endpoint of the IP connection implements IPsec, but network nodes between them protect traffic for part of the way. Protection is transparent to the endpoints, and depends on ordinary routing to send packets through the tunnel endpoints for processing. Each endpoint would announce the set of addresses "behind" it, and packets would be sent in tunnel mode where the inner IP header would contain the IP addresses of the actual endpoints

Endpoint-to-Endpoint Transport



Endpoint to Endpoint

Figure 7.34: Endpoint to Endpoint

In this scenario, both endpoints of the IP connection implement IPsec, as required of hosts in [RFC4301]. Transport mode will commonly be used with no inner IP header. If there is an inner IP header, the inner addresses will be the same as the outer addresses. A single pair of addresses will be negotiated for packets to be protected by this SA. These endpoints MAY implement application layer access controls based on the IPsec authenticated identities of the participants. This scenario enables the end-to-end security that has been a guiding principle for the Internet since [RFC1958], [RFC2775], and a method of limiting the inherent problems with complexity in networks noted by [RFC3439]. Although this scenario may not be fully applicable to the IPv4 Internet, it has been deployed successfully in specific scenarios within intranets using IKEv1. It should be more broadly enabled during the transition to IPv6 and with the adoption of IKEv2

It is possible in this scenario that one or both of the protected endpoints will be behind a network address translation (NAT) node, in which case the tunneled packets will have to be UDP encapsulated so that port numbers in the UDP headers can be used to identify individual endpoints "behind" the NAT (see section 2.23)

Endpoint to Security Gateway Tunnel

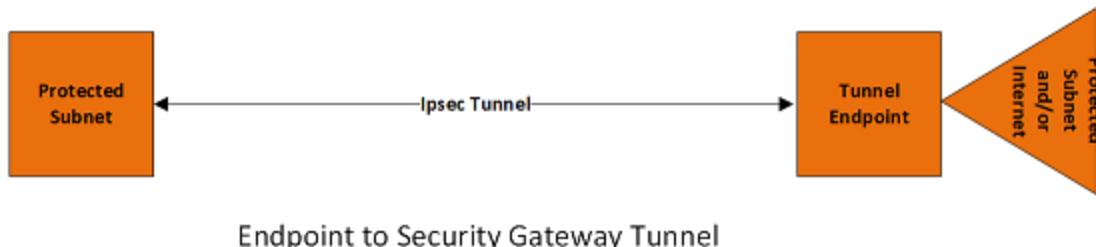


Figure 7.35: Endpoint to Security Gateway Tunnel

In this scenario, a protected endpoint (typically a portable roaming computer) connects back to its corporate network through an IPsec-protected tunnel. It might use this tunnel only to access information on the corporate network, or it might tunnel all of its traffic back through the corporate network in order to take advantage of protection provided by a corporate firewall against Internet-based attacks. In either case, the protected endpoint will want an IP address associated with the security gateway so that packets returned to it will go to the security gateway and be tunneled back. This IP address may be static or may be dynamically allocated by the security gateway. In support of the latter case, IKEv2 includes a mechanism for the initiator to request an IP address owned by the security gateway for use for the duration of its SA. In this scenario, packets will use tunnel mode. On each packet from the protected endpoint, the outer IP header will contain the source IP address associated with its current location (i.e., the address that will get traffic routed to the endpoint directly), while the inner IP header will contain the source IP address assigned by the security gateway (i.e., the address that will get traffic routed to the security gateway for forwarding to the endpoint). The outer destination address will always be that of the security gateway, while the inner destination address will be the ultimate destination for the packet

Generating Keying Material

In the context of the IKE_SA, four cryptographic algorithms are negotiated: an encryption algorithm, an integrity protection algorithm, a Diffie-Hellman group, and a pseudo-random function (prf). The pseudo-random function is used for the construction of keying material for all of the cryptographic algorithms used in both the IKE_SA and the CHILD_SAs

We assume that each encryption algorithm and integrity protection algorithm uses a fixed-size key and that any randomly chosen value of that fixed size can serve as an appropriate key. For algorithms that accept a variable length key, a fixed key size MUST be specified as part of the cryptographic transform negotiated. For algorithms for which not all values are valid keys (such as DES or 3DES with key parity), the algorithm by which keys are derived from arbitrary values MUST be specified by the cryptographic transform. For integrity protection functions based on Hashed Message Authentication Code (HMAC), the fixed key size is the size of the output of the underlying hash function. When the prf function takes a variable length key, variable length data, and produces a fixed-length output (e.g., when using HMAC), the formulas in this document apply. When the key for the prf function has fixed length, the data provided as a key is truncated or padded with zeros as necessary unless exceptional processing is explained following the formula

Keying material will always be derived as the output of the negotiated prf algorithm. Since the amount of keying material needed may be greater than the size of the output of the prf algorithm, we will use the prf iteratively. We will use the terminology prf^+ to describe the function that outputs a pseudo-random stream based on the inputs to a prf as follows: (where $|$ indicates concatenation)

$$\text{prf} + (K, S) = T1 | T2 | T3 | T4 | \dots$$

where:

- $T1 = \text{prf}(K, S | 0x01)$
- $T2 = \text{prf}(K, T1 | S | 0x01)$

- $T3 = \text{prf}(K, T2 | S | 0x01)$
- $T4 = \text{prf}(K, T3 | S | 0x01)$

continuing as needed to compute all required keys. The keys are taken from the output string without regard to boundaries (e.g., if the required keys are a 256-bit Advanced Encryption Standard (AES) key and a 160-bit HMAC key, and the prf function generates 160 bits, the AES key will come from T1 and the beginning of T2, while the HMAC key will come from the rest of T2 and the beginning of T3). The constant concatenated to the end of each string feeding the prf is a single octet. prf+ in this document is not defined beyond 255 times the size of the prf output

Generating Keying Material for CHILD_SAs

The shared keys are computed as follows. A quantity called *SKEYSEED* is calculated from the nonces exchanged during the IKE_SA_INIT exchange and the Diffie-Hellman shared secret established during that exchange. *SKEYSEED* is used to calculate seven other secrets:

- SK_d - used for deriving new keys for the CHILD_SAs established with this IKE_SA
- SK_{ai} and SK_{ar} - used as a key to the integrity protection algorithm for authenticating the component messages of subsequent exchanges
- SK_{ei} and SK_{er} - used for encrypting (and of course decrypting) all subsequent exchanges
- SK_{pi} and SK_{pr} - which are used when generating an AUTH payload

SKEYSEED and its derivatives are computed as follows:

$$SKEYSEED = \text{prf}(Ni | Nr; g^ir)$$

$$SK_d | SK_{ai} | SK_{ar} | SK_{ei} | SK_{er} | SK_{pi} | SK_{pr} = \text{prf} + (SKEYSEED, Ni | Nr | SPIi | SPIr)$$

(indicating that the quantities SK_d , SK_{ai} , SK_{ar} , SK_{ei} , SK_{er} , SK_{pi} , and SK_{pr} are taken in order from the generated bits of the prf+). g^ir is the shared secret from the ephemeral Diffie-Hellman exchange. g^ir is represented as a string of octets in big endian order padded with zeros if necessary to make it the length of the modulus. Ni and Nr are the nonces, stripped of any headers. If the negotiated prf takes a fixed-length key and the lengths of Ni and Nr do not add up to that length, half the bits must come from Ni and half from Nr , taking the first bits of each

The two directions of traffic flow use different keys. The keys used to protect messages from the original initiator are SK_{ai} and SK_{ei} . The keys used to protect messages in the other direction are SK_{ar} and SK_{er} . Each algorithm takes a fixed number of bits of keying material, which is specified as part of the algorithm. For integrity algorithms based on a keyed hash, the key size is always equal to the length of the output of the underlying hash function

A single CHILD_SA is created by the IKE_AUTH exchange, and additional CHILD_SAs can optionally be created in CREATE_CHILD_SA exchanges. Keying material for them is generated as follows:

$$KEYMAT = \text{prf} + (SK_d, Ni | Nr)$$

Where Ni and Nr are the nonces from the IKE_SA_INIT exchange if this request is the first CHILD_SA created or the fresh Ni and Nr from the CREATE_CHILD_SA exchange if this is a subsequent creation.

For CREATE_CHILD_SA exchanges including an optional Diffie-Hellman exchange, the keying material is defined as:

$$KEYMAT = \text{prf} + (SK_d, g^ir(\text{new}) | Ni | Nr)$$

where $g^ir(\text{new})$ is the shared secret from the ephemeral Diffie-Hellman exchange of this CREATE_CHILD_SA exchange (represented as an octet string in big endian order padded with zeros in the high-order bits if necessary to make it the length of the modulus). A single CHILD_SA negotiation may result in multiple security associations. ESP and AH SAs exist in pairs (one in each direction), and four SAs could be created in a single CHILD_SA negotiation if a combination of ESP and AH is being negotiated. Keying material MUST be taken from the expanded KEYMAT in the following order:

- All keys for SAs carrying data from the initiator to the responder are taken before SAs going in the reverse direction

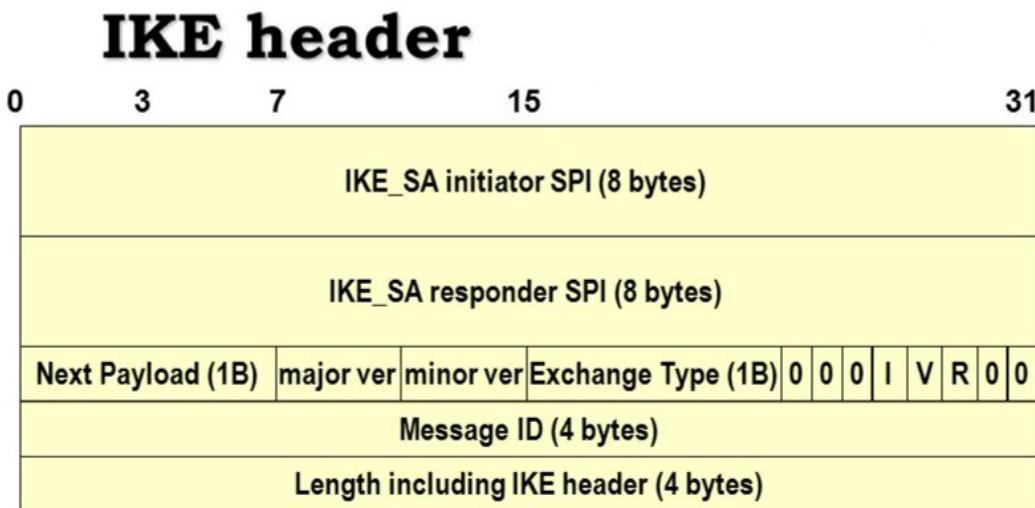
- If multiple IPsec protocols are negotiated, keying material is taken in the order in which the protocol headers will appear in the encapsulated packet
- If a single protocol has both encryption and authentication keys, the encryption key is taken from the first octets of KEYMAT and the authentication key is taken from the next octets

Each cryptographic algorithm takes a fixed number of bits of keying material specified as part of the algorithm

7.55.4 Header and Payload Formats

IKE Header

The format of the IKE header is shown in Figure 4



- **Initiator SPI, responder SPI:**
 - ⇒ SA id for the IKE protocol. Initially, responder SPI set to 0; responder fills it in
 - don't confuse with the SPI for the CHILD AH/ESP SA
- **Versions: major = 2, minor = 0**
 - ⇒ Bit V=1 says that implementation can "speak" higher version than what written in the hdr
- **Exchange type: one of 4 messages: IKE_SA_INIT, IKE_AUTH, CREATE_CHILD_SA, INFORMATIONAL**
 - ⇒ Direction: bit R (response=1, request=0); bit I (original initiator=1, original responder=0; needed to properly match SPIs)
- **Message ID: Sequence number counter, to match requests with responses, to manage retransmission, to combat replay attacks**

— Giuseppe Bianchi —

Figure 7.36: Figure 4: IKE Header Format [1]

- Initiator's SPI (8 octets) - A value chosen by the initiator to identify a unique IKE security association. This value MUST NOT be zero
- Responder's SPI (8 octets) - A value chosen by the responder to identify a unique IKE security association. This value MUST be zero in the first message of an IKE Initial Exchange (including repeats of that message including a cookie) and MUST NOT be zero in any other message
- Next Payload (1 octet) - Indicates the type of payload that immediately follows the header. The format and value of each payload are defined below
- Major Version (4 bits) - Indicates the major version of the IKE protocol in use. Implementations based on this version of IKE MUST set the Major Version to 2. Implementations based on previous versions of IKE and

ISAKMP MUST set the Major Version to 1. Implementations based on this version of IKE MUST reject or ignore messages containing a version number greater than 2

- Minor Version (4 bits) - Indicates the minor version of the IKE protocol in use. Implementations based on this version of IKE MUST set the Minor Version to 0. They MUST ignore the minor version number of received messages
- Exchange Type (1 octet) - Indicates the type of exchange being used. This constrains the payloads sent in each message and orderings of messages in an exchange

- Flags (1 octet) - Indicates specific options that are set for the message. Presence of options are indicated by the appropriate bit in the flags field being set. The bits are defined LSB first, so bit 0 would be the least significant bit of the Flags octet. In the description below, a bit being 'set' means its value is '1', while 'cleared' means its value is '0'
 - X(reserved) (bits 0-2) - These bits MUST be cleared when sending and MUST be ignored on receipt
 - I(initiator) (bit 3 of Flags) - This bit MUST be set in messages sent by the original initiator of the IKE-SA and MUST be cleared in messages sent by the original responder. It is used by the recipient to determine which eight octets of the SPI were generated by the recipient
 - V(ersion) (bit 4 of Flags) - This bit indicates that the transmitter is capable of speaking a higher major version number of the protocol than the one indicated in the major version number field. Implementations of IKEv2 must clear this bit when sending and MUST ignore it in incoming messages
 - R(esponse) (bit 5 of Flags) - This bit indicates that this message is a response to a message containing the same message ID. This bit MUST be cleared in all request messages and MUST be set in all responses. An IKE endpoint MUST NOT generate a response to a message that is marked as being a response
 - X(reserved) (bits 6-7 of Flags) - These bits MUST be cleared when sending and MUST be ignored on receipt
- Message ID (4 octets) - Message identifier used to control retransmission of lost packets and matching of requests and responses. It is essential to the security of the protocol because it is used to prevent message replay attacks
- Length (4 octets) - Length of total message (header + payloads) in octets

7.55.5 Generic Payload Header

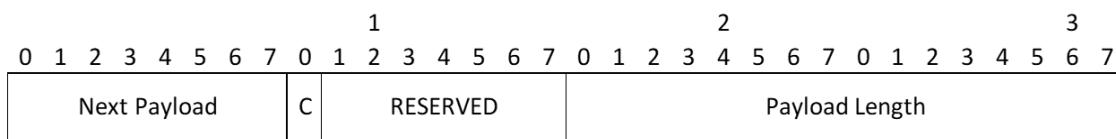


Figure 7.37: Generic Payload Header

- Next Payload - Identifier for the payload type of the next payload in the message. If the current payload is the last in the message, then this field will be 0. This field provides a "chaining" capability whereby additional payloads can be added to a message by appending each one to the end of the message and setting the Next Payload field of the preceding payload to indicate the new payload's type. An Encrypted payload, which must always be the last payload of a message, is an exception. It contains data structures in the format of additional payloads. In the header of an Encrypted payload, the Next Payload field is set to the payload type of the first contained payload (instead of 0); conversely, the Next Payload field of the last contained payload is set to zero. The payload type values are listed here. The values in the following table are only current as of the publication date of RFC 4306. Other values may have been added since then or will be added after the publication of this document. Readers should refer to IANA.org for the latest values

- Critical - MUST be set to zero if the sender wants the recipient to skip this payload if it does not understand the payload type code in the Next Payload field of the previous payload. MUST be set to one if the sender wants the recipient to reject this entire message if it does not understand the payload type. MUST be ignored by the recipient if the recipient understands the payload type code. MUST be set to zero for payload types defined in this document. Note that the critical bit applies to the current payload rather than the "next" payload whose type code appears in the first octet. The reasoning behind not setting the critical bit for payloads defined in this document is that all implementations MUST understand all payload types defined in this document and therefore must ignore the critical bit's value. Skipped payloads are expected to have valid Next Payload and Payload Length fields. See Section 2.5 for more information on this bit
- RESERVED - MUST be set to zero; MUST be ignored on receipt
- Payload Length - Length in octets of the current payload, including the generic payload header

7.55.6 Security Association Payload

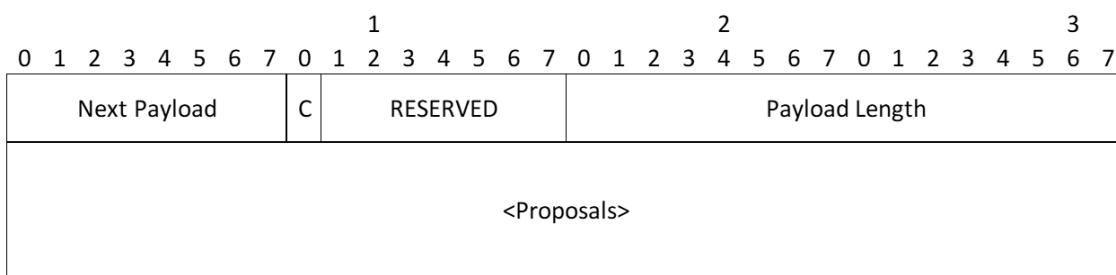


Figure 7.38: Security Association Payload

- Proposals - One or more proposal substructures

7.55.7 Proposal Substructure

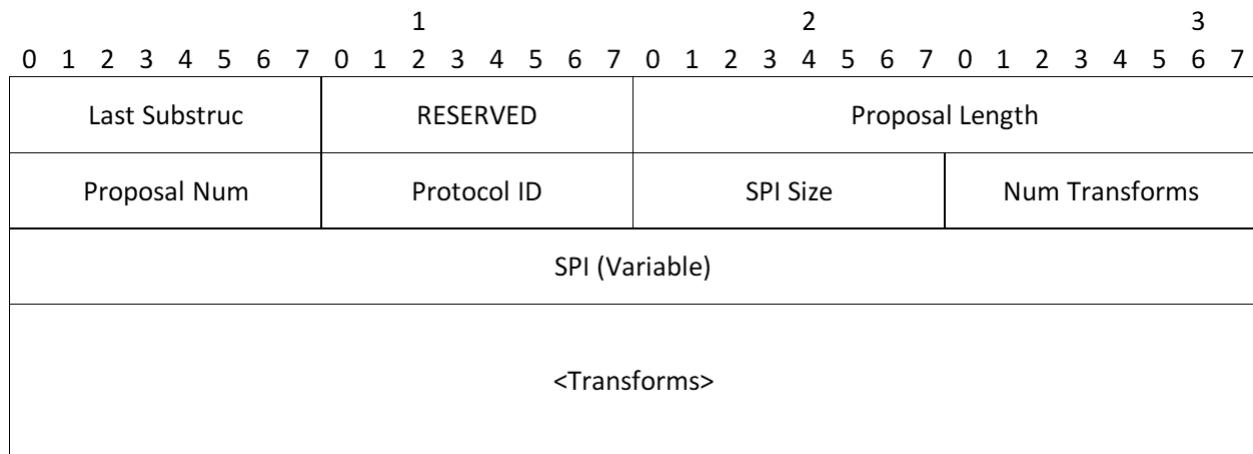


Figure 7.39: Proposal Substructure

- Last Substruc - Specifies whether or not this is the last Proposal Substructure in the SA. This field has a value of 0 if this was the last Proposal Substructure, and a value of 2 if there are more Proposal Substructures. This syntax is inherited from ISAKMP, but is unnecessary because the last Proposal could be identified from the length of the SA. The value (2) corresponds to a payload type of Proposal in IKEv1, and the first four octets of the Proposal structure are designed to look somewhat like the header of a payload

- RESERVED - MUST be sent as zero; MUST be ignored on receipt
- Proposal Length - Length of this proposal, including all transforms and attributes that follow
- Proposal Num - When a proposal is made, the first proposal in an SA payload MUST be 1, and subsequent proposals MUST be one more than the previous proposal (indicating an OR of the two proposals). When a proposal is accepted, the proposal number in the SA payload MUST match the number on the proposal sent that was accepted
- Protocol ID - Specifies the IPsec protocol identifier for the current negotiation. The values in the following table are only current as of the publication date of RFC 4306. Other values may have been added since then or will be added after the publication of this document. Readers should refer to IANA.org for the latest values

- SPI Size - For an initial IKE SA negotiation, this field MUST be zero; the SPI is obtained from the outer header. During subsequent negotiations, it is equal to the size, in octets, of the SPI of the corresponding protocol (8 for IKE, 4 for ESP and AH)
- Num Transforms - Specifies the number of transforms in this proposal
- SPI (variable) - The sending entity's SPI. Even if the SPI Size is not a multiple of 4 octets, there is no padding applied to the payload. When the SPI Size field is zero, this field is not present in the Security Association payload
- Transforms (variable) - One or more transform substructures

7.55.8 Transform Substructure

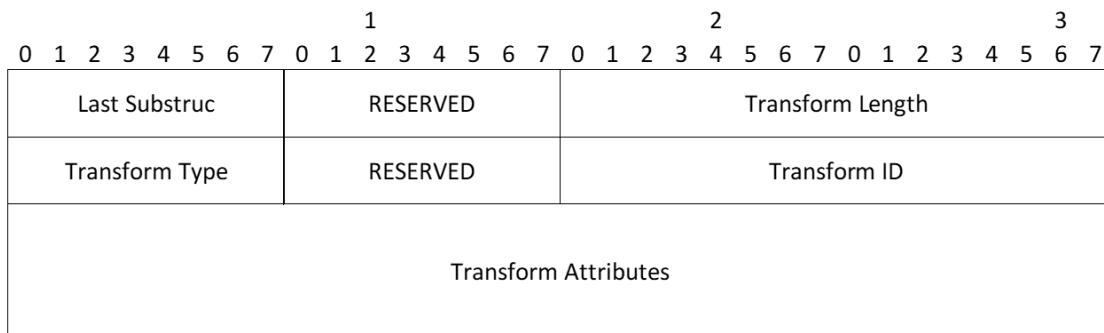


Figure 7.40: Transform Substructure

- Last Substruc - Specifies whether or not this is the last Transform Substructure in the Proposal. This field has a value of 0 if this was the last Transform Substructure, and a value of 3 if there are more Transform Substructures. This syntax is inherited from ISAKMP, but is unnecessary because the last transform could be identified from the length of the proposal. The value (3) corresponds to a payload type of Transform in IKEv1, and the first four octets of the Transform structure are designed to look somewhat like the header of a payload
- RESERVED - MUST be sent as zero; MUST be ignored on receipt
- Transform Length - The length (in octets) of the Transform Substructure including Header and Attributes
- Transform Type - The type of transform being specified in this transform. Different protocols support different Transform Types. For some protocols, some of the transforms may be optional. If a transform is optional and the initiator wishes to propose that the transform be omitted, no transform of the given type is included in the proposal. If the initiator wishes to make use of the transform optional to the responder, it includes a transform substructure with Transform ID = 0 as one of the options
- Transform ID - The specific instance of the Transform Type being proposed see IANA.org for all values

7.55.9 Transform Attributes

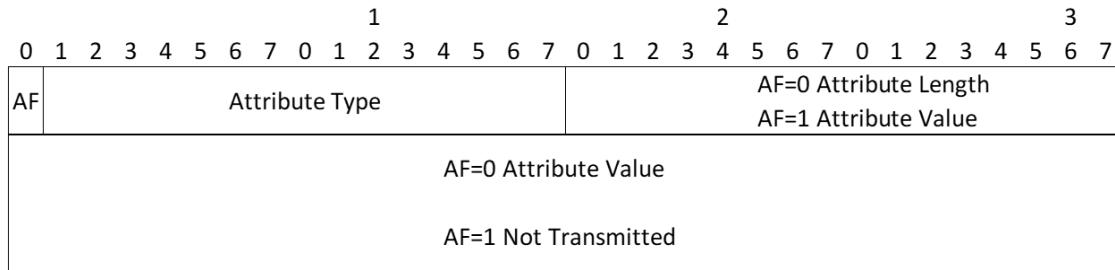


Figure 7.41: Transform Attributes

- Attribute Format (AF) - Indicates whether the data attribute follows the Type/Length/Value (TLV) format or a shortened Type/Value (TV) format. If the AF bit is zero (0), then the attribute uses TLV format; if the AF bit is one (1), the TV format (with two-byte value) is used.
- Attribute Type - Unique identifier for each type of attribute
- Attribute Value (variable length) - Value of the attribute associated with the attribute type. If the AF bit is a zero (0), this field has a variable length defined by the Attribute Length field. If the AF bit is a one (1), the Attribute Value has a length of 2 octets

The only currently defined attribute type (Key Length) is fixed length; the variable-length encoding specification is included only for future extensions. Attributes described as fixed length MUST NOT be encoded using the variable-length encoding unless that length exceeds two bytes. Variable-length attributes MUST NOT be encoded as fixed-length even if their value can fit into two octets. Note: This is a change from IKEv1, where increased flexibility may have simplified the composer of messages but certainly complicated the parser

The Key Length attribute specifies the key length in bits (MUST use network byte order) for certain transforms as follows:

- The Key Length attribute MUST NOT be used with transforms that use a fixed-length key. For example, this includes ENCR_DES, ENCR_IDEA, and all the Type 2 (Pseudorandom Function) and Type 3 (Integrity Algorithm) transforms specified in this document. It is recommended that future Type 2 or 3 transforms do not use this attribute
- Some transforms specify that the Key Length attribute MUST be always included (omitting the attribute is not allowed, and proposals not containing it MUST be rejected). For example, this includes ENCR_AES_CBC and ENCR_AES_CTR
- Some transforms allow variable-length keys, but also specify a default key length if the attribute is not included. For example, these transforms include ENCR_RC5 and ENCR_BLOWFISH

7.55.10 Key Exchange Payload

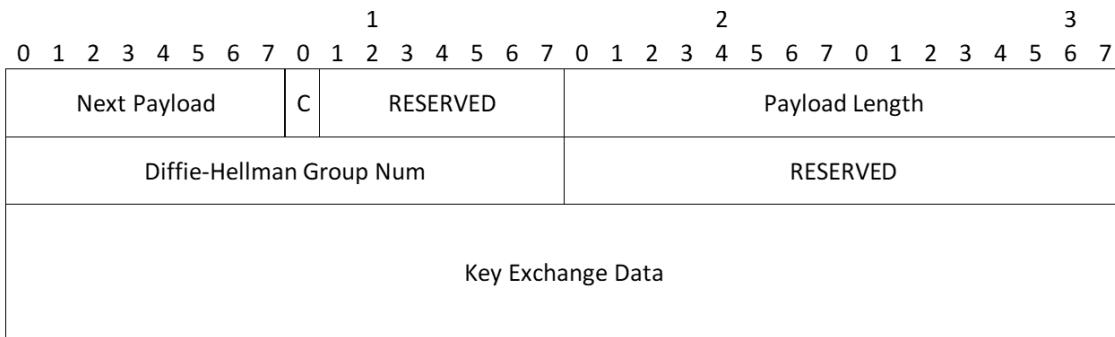


Figure 7.42: Key Exchange

A Key Exchange payload is constructed by copying one's Diffie-Hellman public value into the "Key Exchange Data" portion of the payload. The length of the Diffie-Hellman public value for MODP groups MUST be equal to the length of the prime modulus over which the exponentiation was performed, prepending zero bits to the value if necessary

7.55.11 Identification Payload

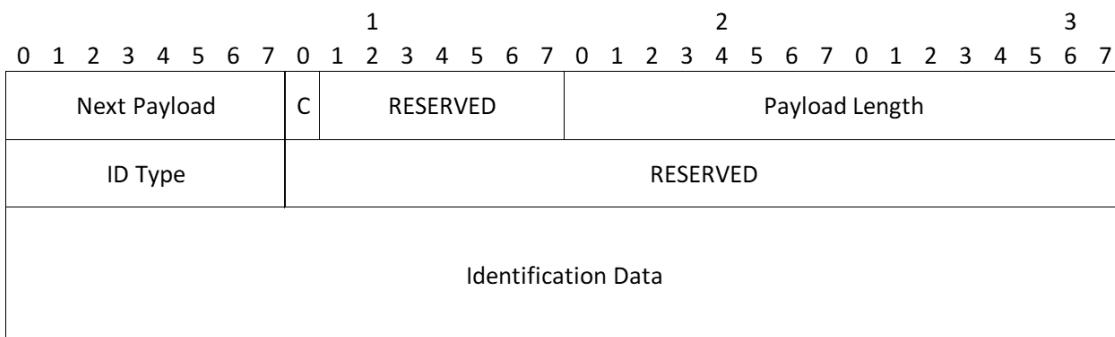


Figure 7.43: Identification Payload

- ID Type - Specifies the type of Identification being used
- RESERVED - MUST be sent as zero; MUST be ignored on receipt
- Identification Data (variable length) - Value, as indicated by the Identification Type. The length of the Identification Data is computed from the size in the ID payload header

7.55.12 Certification Payload

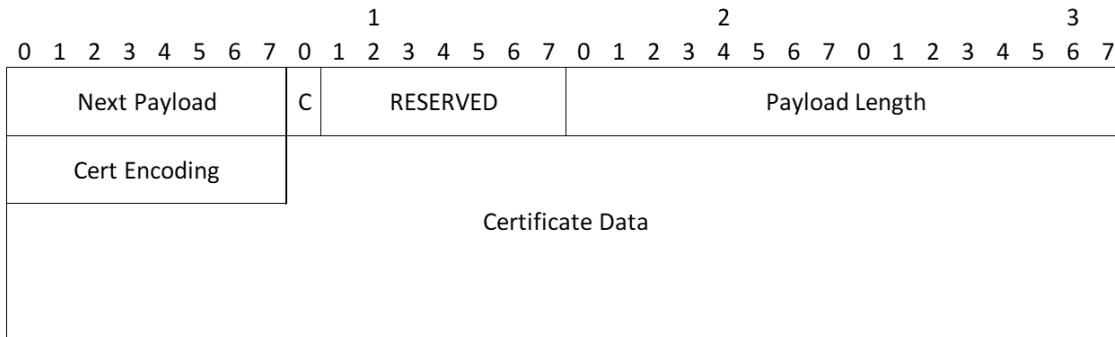


Figure 7.44: Certification Payload

- Certificate Encoding - This field indicates the type of certificate or certificate-related information contained in the Certificate Data field. The values in the following table are only current as of the publication date of RFC 4306. Other values may have been added since then or will be added after the publication of this document. Readers should refer to IANA.org for the latest values
- Certificate Data (variable length) - Actual encoding of certificate data. The type of certificate is indicated by the Certificate Encoding field

7.55.13 Certification Request Payload



Figure 7.45: Certification Request Payload

- Certificate Encoding - Contains an encoding of the type or format of certificate requested. Values are listed in Section 3.6
- Certification Authority (variable length) - Contains an encoding of an acceptable certification authority for the type of certificate requested

7.55.14 Authentication Payload

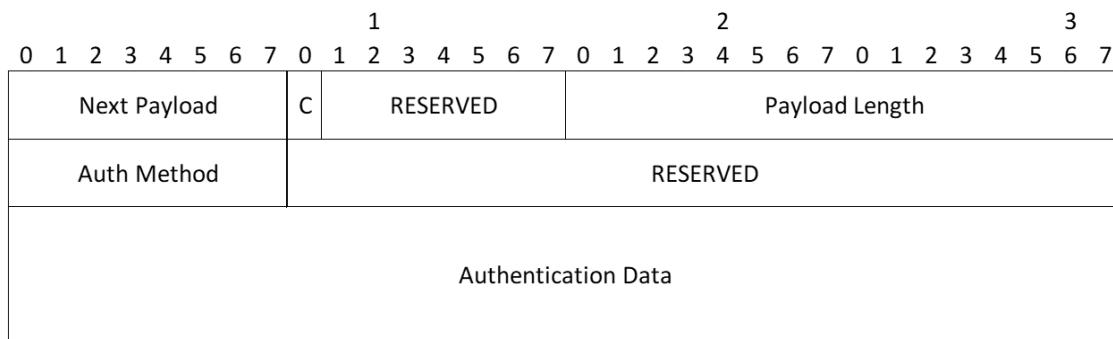


Figure 7.46: Authentication Payload

- Auth Method - Specifies the method of authentication used. The types of signatures are listed here. The values in the following table are only current as of the publication date of RFC 4306. Other values may have been added since then or will be added after the publication of this document. Readers should refer to IANA.org for the latest values
 - RSA Digital Signature - Computed as specified in Section 2.15 using an RSA private key with RSASSA-PKCS1-v1_5 signature scheme specified in [PKCS1] (implementers should note that IKEv1 used a different method for RSA signatures). To promote interoperability, implementations that support this type SHOULD support signatures that use SHA-1 as the hash function and SHOULD use SHA-1 as the default hash function when generating signatures. Implementations can use the certificates received from a given peer as a hint for selecting a mutually understood hash function for the AUTH payload signature. Note, however, that the hash algorithm used in the AUTH payload signature doesn't have to be the same as any hash algorithm(s) used in the certificate(s)
 - Shared Key Message Integrity Code - Computed as specified in Section 2.15 using the shared key associated with the identity in the ID payload and the negotiated PRF
 - DSS Digital Signature - Computed as specified in Section 2.15 using a DSS private key (see [DSS]) over a SHA-1 hash
 - Elliptic Curve DSA (ECDSA) - Computed as specified in Section 2.15 encoding of the computed signature consisting of the concatenation of a pair of integers r and s. The definitions of r and s are given in Section 8 RFC4754
 - Digital Signature - Computer as specified in Section 2.15 with authentication method to support signature methods in a more general way. The current signature-based authentication methods in IKEv2 are per algorithm, i.e., there is one for RSA digital signatures, one for DSS digital signatures (using SHA-1), and three for different ECDSA curves, each tied to exactly one hash algorithm. This design is cumbersome when more signature algorithms, hash algorithms, and elliptic curves need to be supported:
 - In IKEv2, authentication using RSA digital signatures calls for padding based on RSASSA-PKCS1-v1_5, although the newer RSASSA-PSS padding method is now recommended. (See Section 5 of "Additional Algorithms and Identifiers for RSA Cryptography for use in PKIX Profile" [RFC4055].)
 - With ECDSA and the Digital Signature Standard (DSS), there is no way to extract the hash algorithm from the signature. Thus, for each new hash function to be supported with ECDSA or DSA, new authentication methods would be needed. Support for new hash functions is particularly needed for DSS, because the current restriction to SHA-1 limits its security, meaning there is no point of using long keys with SHA-1
 - The tying of ECDSA authentication methods to particular elliptic curve groups requires definition of additional methods for each new group. The combination of new ECDSA groups and hash functions will cause the number of required authentication methods to become unmanageable. Furthermore, the restriction of ECDSA authentication to a specific group is inconsistent with the approach taken with DSS
 - RESERVED - MUST be sent as zero; MUST be ignored on receipt

- Authentication Data (variable length) - see Section 2.15

7.55.15 Nonce Payload

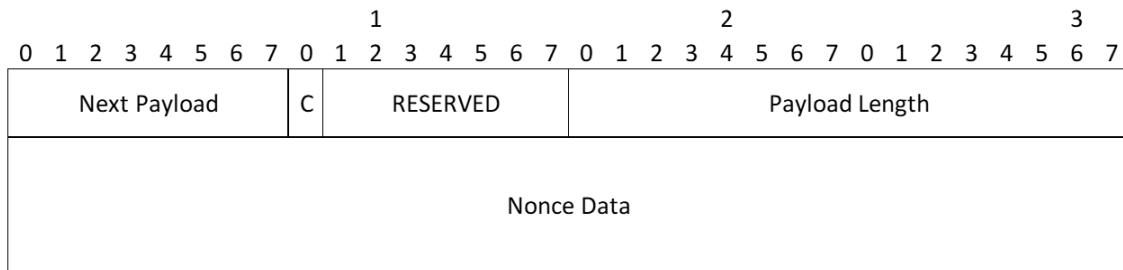


Figure 7.47: Nonce Payload

- Nonce Data (variable length) - Contains the random data generated by the transmitting entity

7.55.16 Notify Payload

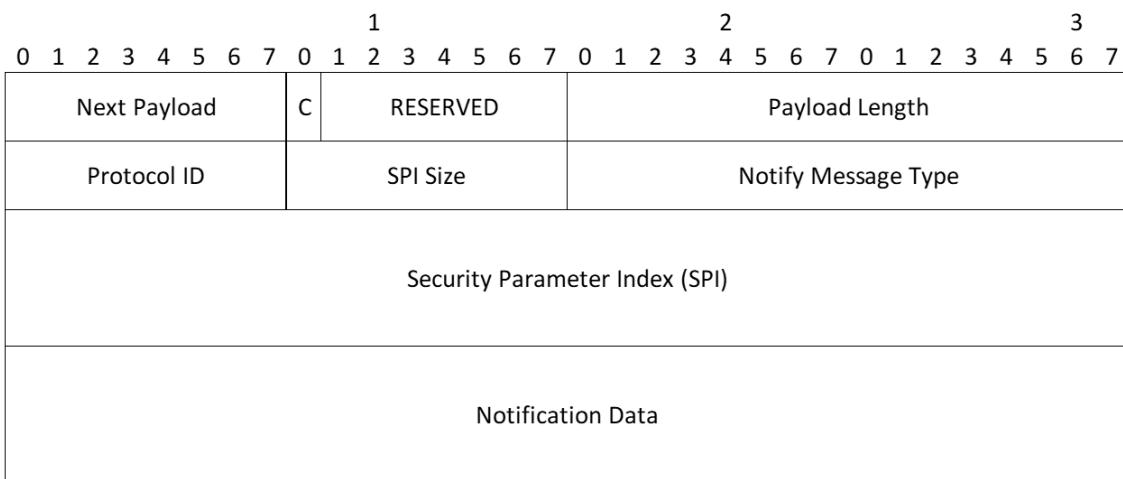


Figure 7.48: Notify Payload

- Protocol ID - If this notification concerns an existing SA whose SPI is given in the SPI field, this field indicates the type of that SA. For notifications concerning Child SAs, this field MUST contain either (2) to indicate AH or (3) to indicate ESP. Of the notifications defined in this document, the SPI is included only with INVALID_SELECTORS, REKEY_SA, and CHILD_SA_NOT_FOUND. If the SPI field is empty, this field MUST be sent as zero and MUST be ignored on receipt
- SPI Size - Length in octets of the SPI as defined by the IPsec protocol ID or zero if no SPI is applicable. For a notification concerning the IKE SA, the SPI Size MUST be zero and the field must be empty
- Notify Message Type - Specifies the type of notification message
- SPI (variable length) - Security Parameter Index
- Notification Data (variable length) - Status or error data transmitted in addition to the Notify Message Type. Values for this field are type specific (see below)

7.55.17 Delete Payload

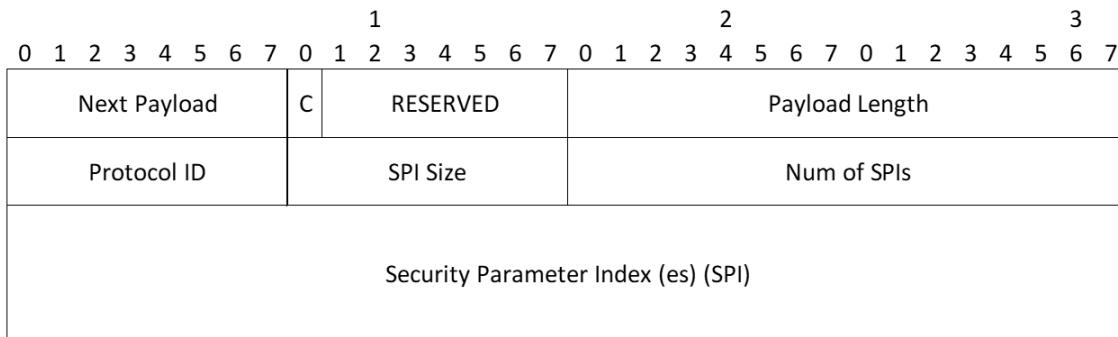


Figure 7.49: Delete Payload

- Protocol ID - Must be 1 for an IKE SA, 2 for AH, or 3 for ESP
- SPI Size - Length in octets of the SPI as defined by the protocol ID. It MUST be zero for IKE (SPI is in message header) or four for AH and ESP
- Num of SPIs (unsigned integer) - The number of SPIs contained in the Delete payload. The size of each SPI is defined by the SPI Size field
- Security Parameter Index(es) (variable length) - Identifies the specific Security Association(s) to delete. The length of this field is determined by the SPI Size and Num of SPIs fields

7.55.18 Vendor ID Payload

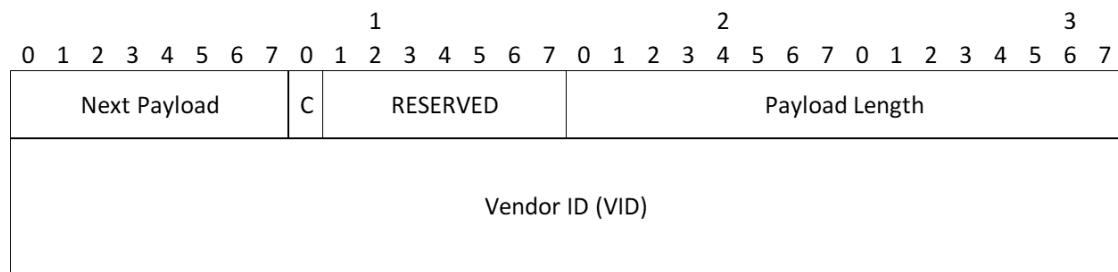


Figure 7.50: Vendor ID Payload

- Vendor ID (variable length) - It is the responsibility of the person choosing the Vendor ID to assure its uniqueness in spite of the absence of any central registry for IDs. Good practice is to include a company name, a person name, or some such information. If you want to show off, you might include the latitude and longitude and time where you were when you chose the ID and some random input. A message digest of a long unique string is preferable to the long unique string itself

7.55.19 Traffic Selectors Payload



Figure 7.51: Traffic Selectors Payload

- Number of TSs - Number of Traffic Selectors being provided
- RESERVED - This field MUST be sent as zero and MUST be ignored on receipt
- Traffic Selectors (variable length) - One or more individual Traffic Selectors

7.55.20 Traffic Selector

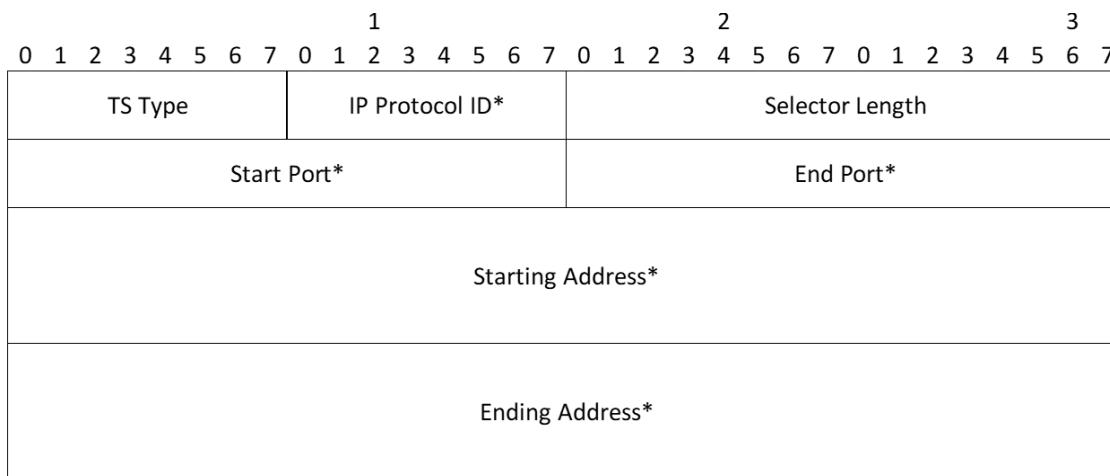


Figure 7.52: Traffic Selector

- TS Type - Specifies the type of Traffic Selector
 - TS_IPV4_ADDR_RANGE - A range of IPv4 addresses, represented by two four-octet values. The first value is the beginning IPv4 address (inclusive) and the second value is the ending IPv4 address (inclusive). All addresses falling between the two specified addresses are considered to be within the list
 - TS_IPV6_ADDR_RANGE - A range of IPv6 addresses, represented by two sixteen-octet values. The first value is the beginning IPv6 address (inclusive) and the second value is the ending IPv6 address (inclusive). All addresses falling between the two specified addresses are considered to be within the list
- IP protocol ID - Value specifying an associated IP protocol ID (such as UDP, TCP, and ICMP). A value of zero means that the protocol ID is not relevant to this Traffic Selector - the SA can carry all protocols

- Selector Length - Specifies the length of this Traffic Selector substructure including the header
- Start Port - Value specifying the smallest port number allowed by this Traffic Selector. For protocols for which port is undefined (including protocol 0), or if all ports are allowed, this field MUST be zero. ICMP and ICMPv6 Type and Code values, as well as Mobile IP version 6 (MIPv6) mobility header (MH) Type values, are represented in this field as specified in Section 4.4.1.1 of [IPSESEARCH]. ICMP Type and Code values are treated as a single 16-bit integer port number, with Type in the most significant eight bits and Code in the least significant eight bits. MIPv6 MH Type values are treated as a single 16-bit integer port number, with Type in the most significant eight bits and the least significant eight bits set to zero
- End Port - Value specifying the largest port number allowed by this Traffic Selector. For protocols for which port is undefined (including protocol 0), or if all ports are allowed, this field MUST be 65535. ICMP and ICMPv6 Type and Code values, as well as MIPv6 MH Type values, are represented in this field as specified in Section 4.4.1.1 of [IPSESEARCH]. ICMP Type and Code values are treated as a single 16-bit integer port number, with Type in the most significant eight bits and Code in the least significant eight bits. MIPv6 MH Type values are treated as a single 16-bit integer port number, with Type in the most significant eight bits and the least significant eight bits set to zero
- Starting Address - The smallest address included in this Traffic Selector (length determined by TS Type)
- Ending Address - The largest address included in this Traffic Selector (length determined by TS Type)

7.55.21 Encrypted Payload Format

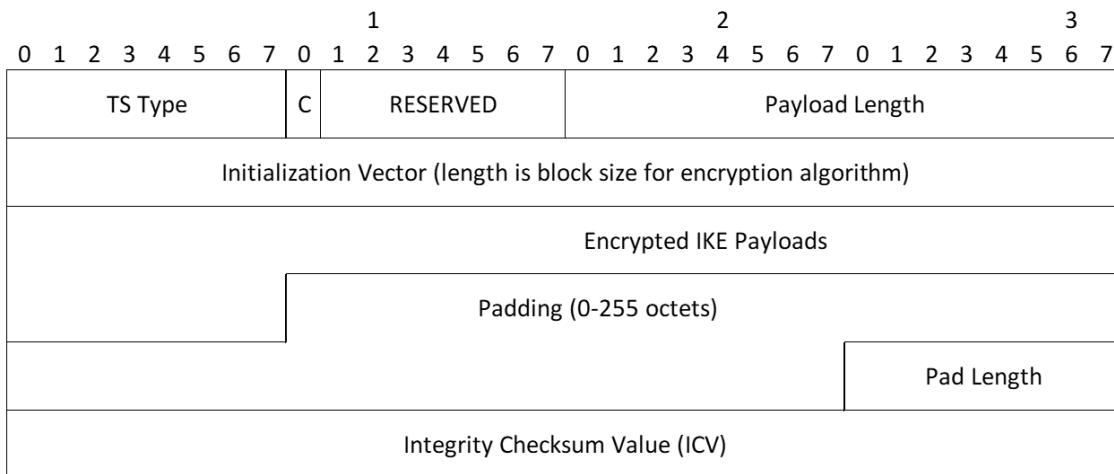


Figure 7.53: Encrypted Payload Format

- Next Payload - The payload type of the first embedded payload. Note that this is an exception in the standard header format, since the Encrypted payload is the last payload in the message and therefore the Next Payload field would normally be zero. But because the content of this payload is embedded payloads and there was no natural place to put the type of the first one, that type is placed here
- Payload Length - Includes the lengths of the header, initialization vector (IV), Encrypted IKE payloads, Padding, Pad Length, and Integrity Checksum Data
- Initialization Vector - For CBC mode ciphers, the length of the initialization vector (IV) is equal to the block length of the underlying encryption algorithm. Senders MUST select a new unpredictable IV for every message; recipients MUST accept any value. The reader is encouraged to consult [MODES] for advice on IV generation. In particular, using the final ciphertext block of the previous message is not considered unpredictable. For modes other than CBC, the IV format and processing is specified in the document specifying the encryption algorithm and mode
- IKE payloads are as specified earlier in this section. This field is encrypted with the negotiated cipher

- Padding MAY contain any value chosen by the sender, and MUST have a length that makes the combination of the payloads, the Padding, and the Pad Length to be a multiple of the encryption block size. This field is encrypted with the negotiated cipher
- Pad Length is the length of the Padding field. The sender SHOULD set the Pad Length to the minimum value that makes the combination of the payloads, the Padding, and the Pad Length a multiple of the block size, but the recipient MUST accept any length that results in proper alignment. This field is encrypted with the negotiated cipher
- Integrity Checksum Data is the cryptographic checksum of the entire message starting with the Fixed IKE header through the Pad Length. The checksum MUST be computed over the encrypted message. Its length is determined by the integrity algorithm negotiated

7.55.22 Configuration Payload

0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7																		
Next Payload	C	RESERVED	Payload Length																		3	2	1																		
CFG Type	RESERVED																																								
Configuration Attributes																																									

Figure 7.54: Configuration Payload

- CFG Type - The type of exchange represented by the Configuration Attributes. The values in the following table are only current as of the publication date of RFC 4306. Other values may have been added since then or will be added after the publication of this document. Readers should refer to IANA.org for the latest values
 - CFG_REQUEST - 1
 - CFG_REPLY - 2
 - CFG_SET - 3
 - CFG_ACK - 4
- RESERVED - MUST be sent as zero; MUST be ignored on receipt
- Configuration Attributes (variable length) - These are type length value (TLV) structures specific to the Configuration payload and are defined below. There may be zero or more Configuration Attributes in this payload

7.55.23 Configuration Attributes

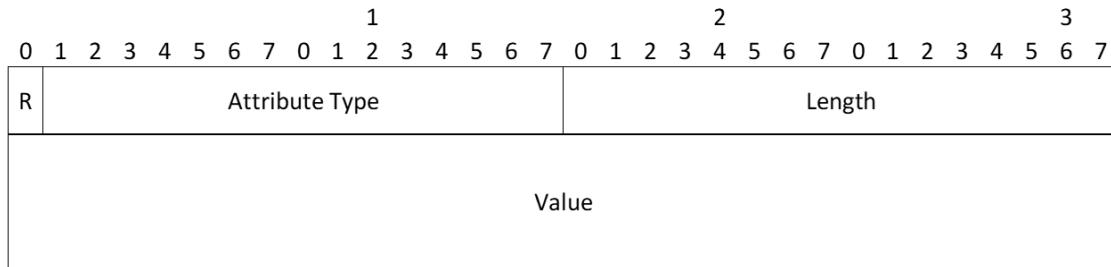


Figure 7.55: Configuration Attribute Format

- Reserved - This bit MUST be set to zero and MUST be ignored on receipt
 - Attribute Type - A unique identifier for each of the Configuration Attribute Types
 - Length - Length in octets of value
 - Value (0 or more octets) - The variable-length value of this Configuration Attribute. The following lists the attribute types

7.55.24 EAP Payload

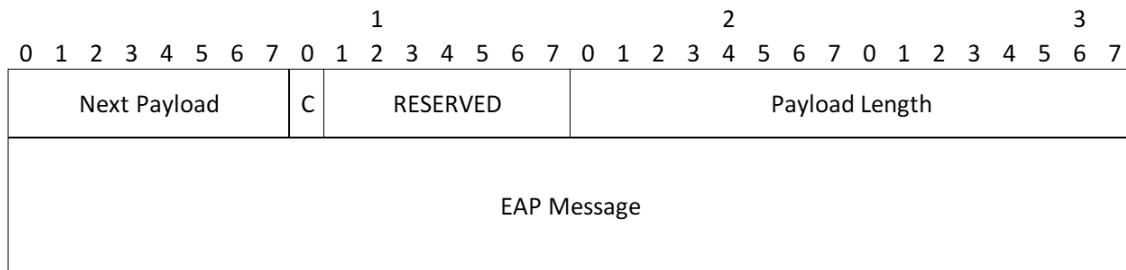


Figure 7.56: EAP Payload Format

7.55.25 EAP Message

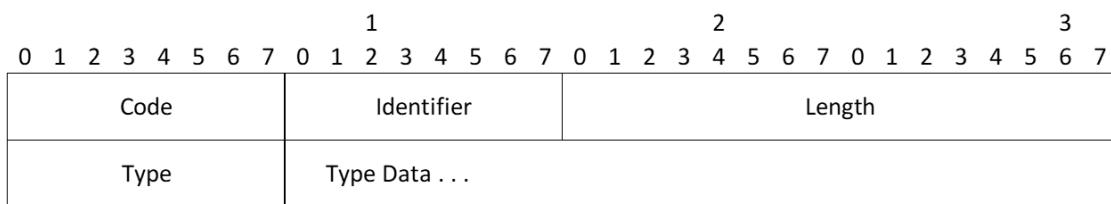


Figure 7.57: EAP Message Format

- Code - Indicates whether this message is a Request (1), Response (2), Success (3), or Failure (4)

- Identifier - Used in PPP to distinguish replayed messages from repeated ones. Since in IKE, EAP runs over a reliable protocol, the Identifier serves no function here. In a response message, this octet MUST be set to match the identifier in the corresponding request
- Length - The length of the EAP message. MUST be four less than the Payload Length of the encapsulating payload
- Type - Present only if the Code field is Request (1) or Response (2). For other codes, the EAP message length MUST be four octets and the Type and Type_Data fields MUST NOT be present. In a Request (1) message, Type indicates the data being requested. In a Response (2) message, Type MUST either be Nak or match the type of the data requested. Note that since IKE passes an indication of initiator identity in the first message in the IKE_AUTH exchange, the responder SHOULD NOT send EAP Identity requests (type 1). The initiator MAY, however, respond to such requests if it receives them
- Type_Data (variable length) - Varies with the Type of Request and the associated Response. For the documentation of the EAP methods

7.55.26 IKEv2 State Diagram

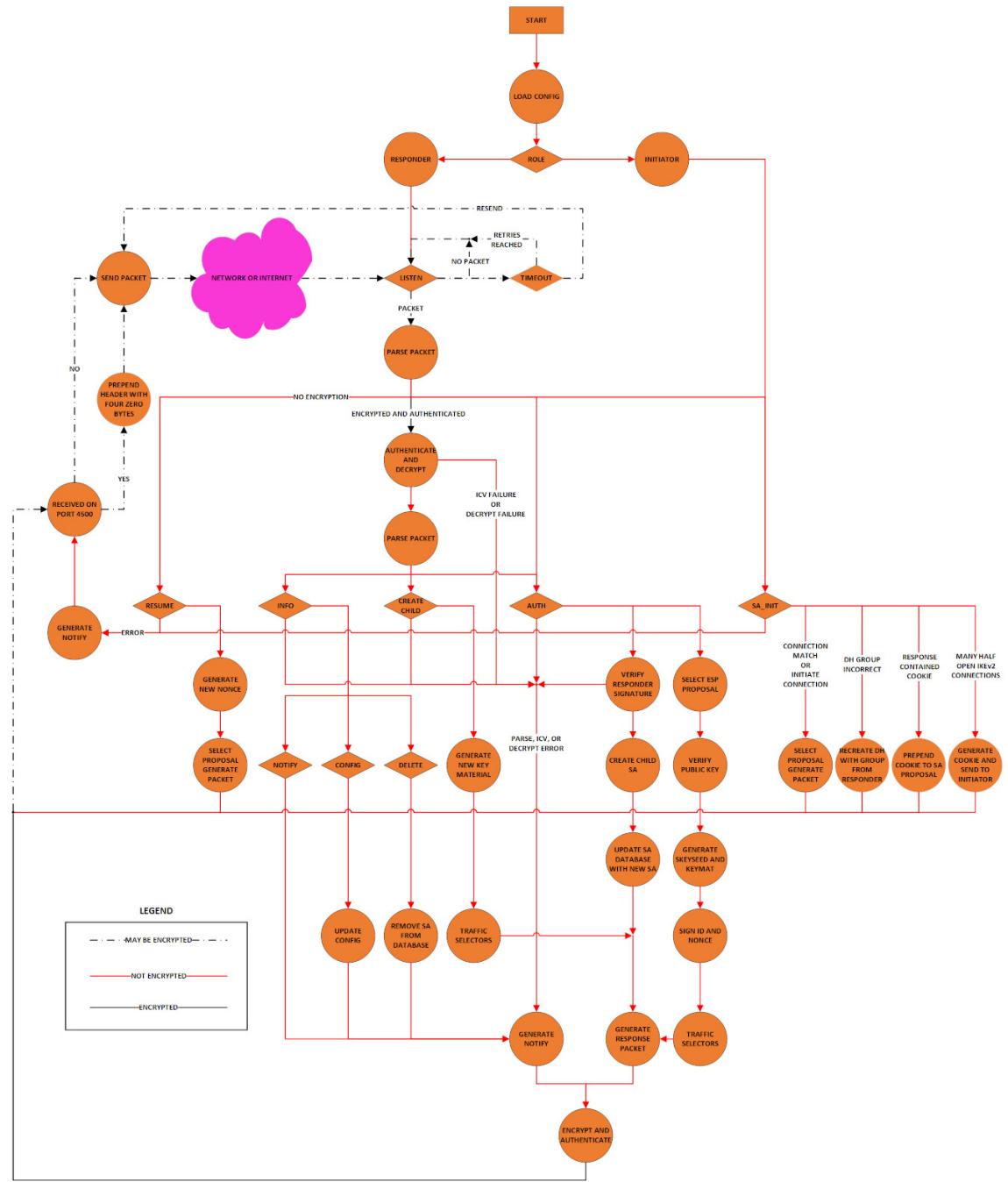


Figure 7.58: IKEv2 State Diagram

For API Documentation:

See Also

ProtocolPP::jsecass
ProtocolPP::jikev2sa
ProtocolPP::jproto
ProtocolPP::jikev2
ProtocolPP::jmodes
ProtocolPP::sm4

For Additional Documentation:**See Also**

[jsecass](#)
[jikev2sa](#)
[jprotocol](#)
[jikev2](#)
[jmodes](#)
[sm4](#)

Protocol++ written by : John Peter Greninger © John Peter Greninger 2015-2019 All Rights Reserved

All copyrights and trademarks are the property of their respective owners

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer
- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution
- Any and all modifications must be returned to John Peter Greninger at GitHub.com <https://github.com/jpgreninger/protocolpp> for evaluation. Inclusion of modifications in the source code shall be determined solely by John Peter Greninger. Failure to provide modifications shall render this license NULL and VOID and revoke any rights to use of Protocol++
- Commercial use requires a fee-based license obtainable at www.protocolpp.com
- Academic use requires written and notarized permission from John Peter and Sheila Rocha Greninger
- US Copyrights at <https://www.copyright.gov/>
 - TXU002059872 (Version 1.0.0)
 - TXU002066632 (Version 1.2.7)
 - TXU002082674 (Version 1.4.0)
 - TXU002097880 (Version 2.0.0)

The names of its contributors may not be used to endorse or promote products derived from this software without specific prior written permission

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE

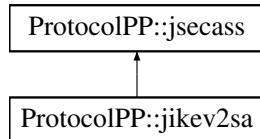
The documentation for this class was generated from the following file:

- jikev2/include/[jikev2sa.h](#)

7.56 ProtocolPP::jikev2sa Class Reference

```
#include <jikev2sa.h>
```

Inheritance diagram for ProtocolPP::jikev2sa:



Public Member Functions

- [jikev2sa \(\)](#)
- [jikev2sa \(direction_t dir, std::string ikecnxt, exchg_t exchg, ike_pyld_t ntxpyld, uint8_t majorev, uint8_t minorev, uint8_t flags, uint32_t msgid_init, uint32_t msgid_resp, jarray< uint8_t > SPli, jarray< uint8_t > SPlr, jarray< uint8_t > Ni, jarray< uint8_t > Nr, jarray< uint8_t > SKd, encr_id_t cipher, unsigned int ckeylen, unsigned int ivlen, unsigned int saltlen, jarray< uint8_t > SKei, jarray< uint8_t > SKer, jarray< uint8_t > iv, jarray< uint8_t > SKsi, jarray< uint8_t > SKsr, integ_id_t integ, unsigned int akeylen, unsigned int icvlen, jarray< uint8_t > SKai, jarray< uint8_t > SKar, prf_id_t prf, unsigned int prflen, jarray< uint8_t > SKpi, jarray< uint8_t > SKpr, dh_id_t dh, auth_method_t ikeauth, unsigned int arlen\)](#)
- [jikev2sa \(jikev2sa &rhs\)](#)
- [jikev2sa \(std::shared_ptr< jikev2sa > &rhs\)](#)
- [virtual ~jikev2sa \(\)](#)
- [template<typename T > void set_field \(field_t field, T value\)](#)
- [template<typename T > T get_field \(field_t field\)](#)
- [void to_xml \(tinyxml2::XMLPrinter &myxml, direction_t direction\)](#)

7.56.1 Constructor & Destructor Documentation

7.56.1.1 ProtocolPP::jikev2sa::jikev2sa ()

Standard constructor with defaults

7.56.1.2 ProtocolPP::jikev2sa::jikev2sa (direction_t dir, std::string ikecnxt, exchg_t exchg, ike_pyld_t ntxpyld, uint8_t majorev, uint8_t minorev, uint8_t flags, uint32_t msgid_init, uint32_t msgid_resp, jarray< uint8_t > SPli, jarray< uint8_t > SPlr, jarray< uint8_t > Ni, jarray< uint8_t > Nr, jarray< uint8_t > SKd, encr_id_t cipher, unsigned int ckeylen, unsigned int ivlen, unsigned int saltlen, jarray< uint8_t > SKei, jarray< uint8_t > SKer, jarray< uint8_t > iv, jarray< uint8_t > SKsi, jarray< uint8_t > SKsr, integ_id_t integ, unsigned int akeylen, unsigned int icvlen, jarray< uint8_t > SKai, jarray< uint8_t > SKar, prf_id_t prf, unsigned int prflen, jarray< uint8_t > SKpi, jarray< uint8_t > SKpr, dh_id_t dh, auth_method_t ikeauth, unsigned int arlen)

See RFC7296 for required fields and their meanings

Security Association for IKEv2. Initialization Vectors are generated randomly using the Mersenne Twister algorithm or are passed to the SA during key negotiation depending on the randiv setting

Parameters

<i>dir</i>	- Direction of processing ENCAP (or out) and DECAP (or in)
<i>ikecnxt</i>	- IKE connection name
<i>exchg</i>	- Exchange type
<i>nxtpyld</i>	- Next payload after header
<i>majorev</i>	- Major IKE version (currently 2)
<i>minorev</i>	- Minor IKE version (currently 0)
<i>flags</i>	- Flags for responder, initiator, and version check
<i>msgid_init</i>	- Message ID for INITIATOR (default is 0)
<i>msgid_resp</i>	- Message ID for RESPONDER (default is 0)
<i>SPli</i>	- Initiator Security Protocol Index
<i>SPIr</i>	- Responder Security Protocol Index
<i>Ni</i>	- Initiator Nonce
<i>Nr</i>	- Responder Nonce
<i>SKd</i>	- IKE D Key
<i>cipher</i>	- Encryption algorithm
<i>ckeylen</i>	- IKE Cipher Algorithm Key Length
<i>ivlen</i>	- IKE Cipher Initialization Vecotr Length
<i>saltlen</i>	- IKE Cipher Salt Length
<i>SKei</i>	- IKE Cipher Initiator Key
<i>SKer</i>	- IKE Cipher Responder Key
<i>iv</i>	- IKE Cipher Initialization Vector
<i>SKSi</i>	- IKE Cipher Initiator Salt Value
<i>SKSr</i>	- IKE Cipher Initiator Salt Value
<i>integ</i>	- IKE Integrity Algorithm
<i>SKAi</i>	- IKE Integrity Algorithm Initiator Key
<i>SKAr</i>	- IKE Integrity Algorithm Responder Key
<i>akeylen</i>	- IKE Integrity Algorithm Key Length
<i>icvlen</i>	- IKE Integerity Check Value
<i>prf</i>	- IKE Psuedo Random Function
<i>prflen</i>	- IKE Psuedo Random Function Key Length
<i>SKpi</i>	- IKE Psuedo Random Function Initiator Key
<i>SKpr</i>	- IKE Psuedo Random Function Responder Key
<i>dh</i>	- IKE Diffie-Hellman Method
<i>ikeauth</i>	- IKE Authentication Algorithm
<i>arlen</i>	- IKE Anti-Replay Length

7.56.1.3 ProtocolPP::jikev2sa::jikev2sa (*jikev2sa & rhs*)

Constructor for IKEv2

Parameters

<i>rhs</i>	- Security association (SA) for this IKEv2 flow
------------	---

7.56.1.4 ProtocolPP::jikev2sa::jikev2sa (*std::shared_ptr<jikev2sa> & rhs*)

Constructor for IKEv2

Parameters

<i>rhs</i>	- Security association (SA) for this IKEv2 flow
------------	---

7.56.1.5 virtual ProtocolPP::jikev2sa::~jikev2sa () [inline], [virtual]

Standard deconstructor flush and close the auditlog if present

	field name	Example
7.56.2.1	DIRECTION template<typename T > T ProtocolPP::ikev2sa::get_field (field_t field) Due to their dynamic nature, some fields are excluded from IKE_SA which include the following fields	set_field<direction_t>(ProtocolPP::DIRECTION, direction_t::DECAP)
	EXCHG • LENGTH	set_field<exchg_t>(ProtocolPP::EXCHG, exchg_t::PYLD_SA)
Parameters	MAJOR_REVISION	set_field<uint8_t>(ProtocolPP::MAJOR_REVISION, 0x00)
	<i>field</i> - field to return from the security association	
Returns	MINOR_REVISION	set_field<uint8_t>(ProtocolPP::MINOR_REVISION, 0x00)
	MSGID	set_field<uint32_t>(ProtocolPP::MSGID, 0x00000011)
7.56.2.2	SPIei Update IKEv2 field with the new value	set_field<jarray<uint8_t>>(ProtocolPP::SPIei, jarray<uint8_t>("AABBCCDDEE-FF00112233445566778899"))
	ENCR Due to ENCR dynamic nature, some fields are excluded from IKE_SA which include the following fields	set_field<encr_id_t>(ProtocolPP::ENCR, encr_id_t::ENCR_DES)
Parameters	• LENGTH INTEG	set_field<integ_id_t>(ProtocolPP::INTEG, integ_id_t::AUTH_HMAC_MD5 - 96)
	<i>field</i> - field to update	set_field<prf_id_t>(ProtocolPP::PRF,
	<i>value</i> - new value for the field	prf_id_t::PRF_HMAC_SHA2_256)
7.56.2.3	DH Return the protocol and security fields as XML	set_field<dh_id_t>(ProtocolPP::DH, dh_id_t::DH_ECP_384)
Parameters	IKEPRF	set_field<auth_method_t>(-ProtocolPP::IKEPRF,
	<i>myxml</i> - XMLPrinter object	auth_method_t::AUTH_ECDSA - P256)
	<i>direction</i> - randomization	

Table 7.32: IKEv2 Set Fields
Implements [ProtocolPP::jsecass](#).

The documentation for this class was generated from the following file:

- [jikev2/include/jikev2sa.h](#)

7.57 jip Class Reference

```
#include "include/jip.h"
```

7.57.1 Detailed Description

7.57.2 Internet Protocol (IP)

See https://en.wikipedia.org/wiki/Internet_Protocol

The Internet Protocol (IP) is the principal communications protocol in the Internet protocol suite for relaying datagrams across network boundaries. Its routing function enables internetworking, and essentially establishes the Internet.

IP has the task of delivering packets from the source host to the destination host solely based on the IP addresses in the packet headers. For this purpose, IP defines packet structures that encapsulate the data to be delivered. It also defines addressing methods that are used to label the datagram with source and destination information.

Historically, IP was the connectionless datagram service in the original Transmission Control Program introduced by Vint Cerf and Bob Kahn in 1974; the other being the connection-oriented Transmission Control Protocol (TCP). The Internet protocol suite is therefore often referred to as TCP/IP.

The first major version of IP, Internet Protocol Version 4 (IPv4), is the dominant protocol of the Internet. Its successor is Internet Protocol V6

The header for IPv4 is configured as follows (See RFC 791) [1]

Bit	+0..7		+8..15		+16..23		+24..31							
0	Version	Header Length	DSCP	ECN	Total Length									
32	Identification				Flags	Fragment Offset								
64	Time To Live		Protocol		Header Checksum									
96	Source IP Address													
128	Destination IP Address													
160	Options (if present)													
...	Payload													

Figure 7.59: IP Version 4 Header

- Version: 4 bits

The Version field indicates the format of the internet header

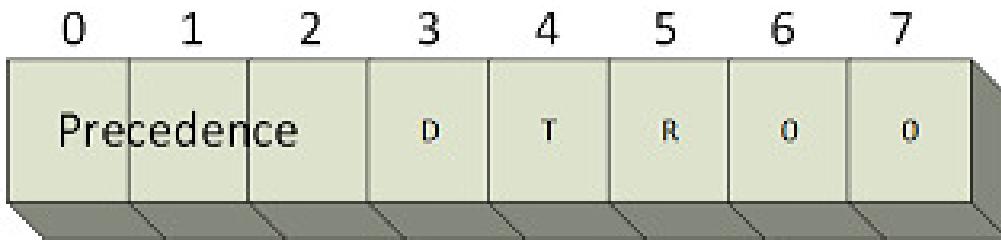
- IHL: 4 bits

Internet Header Length is the length of the internet header in 32 bit words, and thus points to the beginning of the data. Note that the minimum value for a correct header is 5.

- Type of Service: 8 bits

The Type of Service provides an indication of the abstract parameters of the quality of service desired. These parameters are to be used to guide the selection of the actual service parameters when transmitting a datagram through a particular network. Several networks offer service precedence, which somehow treats high precedence traffic as more important than other traffic (generally by accepting only traffic above a certain precedence at time of high load). The major choice is a three way tradeoff between low-delay, high-reliability, and high-throughput.

Bit Position	Usage	
0-2	Precedence	
3	0 = Normal Delay	1 = Low Delay
4	0 = Normal Throughput	1 = High Throughput
5	0 = Normal Reliability	1 = High Reliability
6-7	Reserved for Future Use	



Precedence	Type
000	Routine
001	Priority
010	Immediate
011	Flash
100	Flash Override
101	CRITIC/ECP
110	Internetwork Control
111	Network Control

Figure 7.60: IP Version 4 ToS

The use of the Delay, Throughput, and Reliability indications may increase the cost (in some sense) of the service. In many networks better performance for one of these parameters is coupled with worse performance on another. Except for very unusual cases at most two of these three indications should be set.

The type of service is used to specify the treatment of the datagram during its transmission through the internet system. Example mappings of the internet type of service to the actual service provided on networks such as AUTODIN II, ARPANET, SATNET, and PRNET is given in "Service Mappings".

The Network Control precedence designation is intended to be used within a network only. The actual use and control of that designation is up to each network. The Internetwork Control designation is intended for use by gateway control originators only. If the actual use of these precedence designations is of concern to a particular network, it is the responsibility of that network to control the access to, and use of, those precedence designations.

- Total Length: 16 bits

Total Length is the length of the datagram, measured in octets, including internet header and data. This field

allows the length of a datagram to be up to 65,535 octets. Such long datagrams are impractical for most hosts and networks. All hosts must be prepared to accept datagrams of up to 576 octets (whether they arrive whole or in fragments). It is recommended that hosts only send datagrams larger than 576 octets if they have assurance that the destination is prepared to accept the larger datagrams.

The number 576 is selected to allow a reasonable sized data block to be transmitted in addition to the required header information. For example, this size allows a data block of 512 octets plus 64 header octets to fit in a datagram. The maximal internet header is 60 octets, and a typical internet header is 20 octets, allowing a margin for headers of higher level protocols.

- Identification: 16 bits

An identifying value assigned by the sender to aid in assembling the fragments of a datagram.

- Flags: 3 bits

Flags	Usage	
0	Reserved	
1	0 = May Fragment	1 = Don't Fragment
2	0 = Last Fragment	1 = More Fragments

Figure 7.61: IP Version 4 Control Flags

- Fragment Offset: 13 bits

This field indicates where in the datagram this fragment belongs. The fragment offset is measured in units of 8 octets (64 bits). The first fragment has offset zero.

- Time to Live: 8 bits

This field indicates the maximum time the datagram is allowed to remain in the internet system. If this field contains the value zero, then the datagram must be destroyed. This field is modified in internet header processing. The time is measured in units of seconds, but since every module that processes a datagram must decrease the TTL by at least one even if it processes the datagram in less than a second, the TTL must be thought of only as an upper bound on the time a datagram may exist. The intention is to cause undeliverable datagrams to be discarded, and to bound the maximum datagram lifetime.

- Protocol: 8 bits

This field indicates the next level protocol used in the data portion of the internet datagram. The values for various protocols are specified in "Assigned Numbers".

- Header Checksum: 16 bits

A checksum on the header only. Since some header fields change (e.g., time to live), this is recomputed and verified at each point that the internet header is processed. The checksum algorithm is:

The checksum field is the 16 bit one's complement of the one's complement sum of all 16 bit words in the header. For purposes of computing the checksum, the value of the checksum field is zero.

This is a simple to compute checksum and experimental evidence indicates it is adequate, but it is provisional and may be replaced by a CRC procedure, depending on further experience.

- Source Address: 32 bits

The source address. See section 3.2.

- Destination Address: 32 bits

The destination address. See section 3.2.

The header for IPv6 is configured as follows (see RFC 2460) [2]

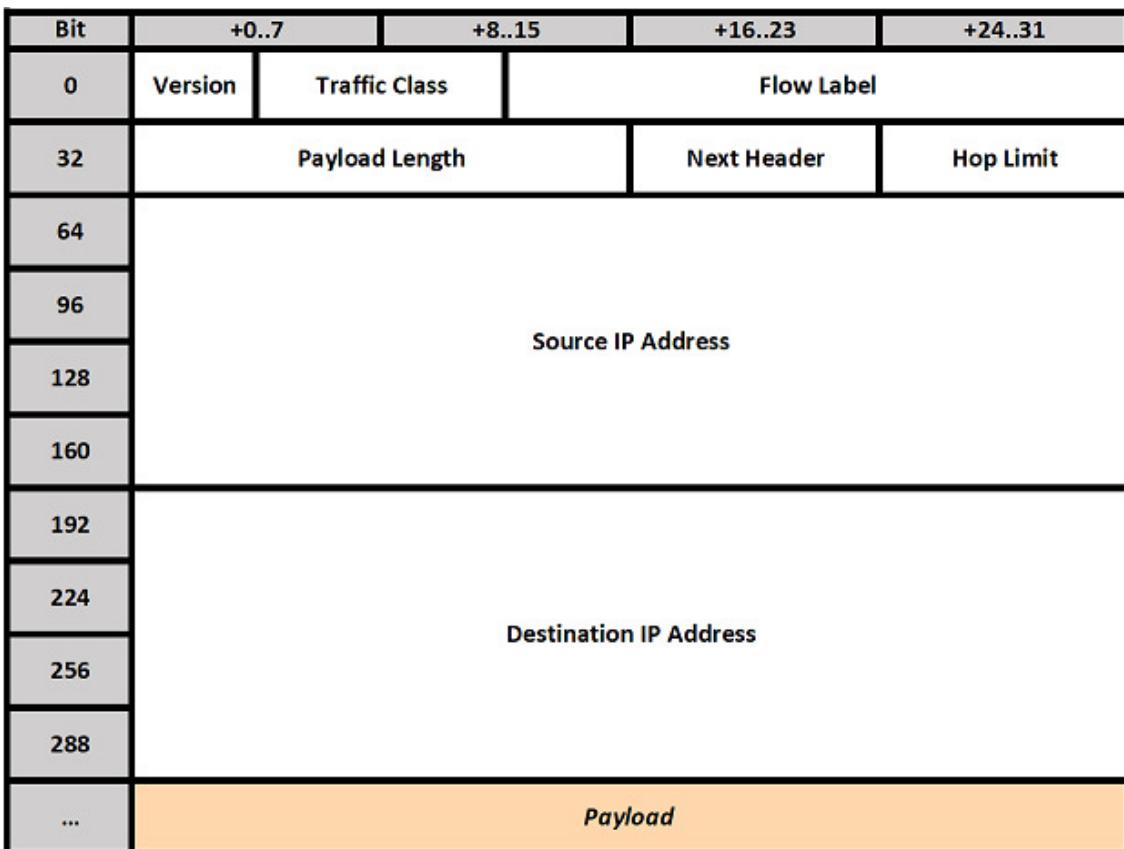


Figure 7.62: IP Version 6 Header

- Version:
4-bit Internet Protocol version number = 6.
- Traffic Class:
8-bit traffic class field. See section 7.
- Flow Label:
20-bit flow label. See section 6.
- Payload Length:
16-bit unsigned integer. Length of the IPv6 payload, i.e., the rest of the packet following this IPv6 header, in octets. (Note that any extension headers present are considered part of the payload, i.e., included in the length count.)
- Next Header:
8-bit selector. Identifies the type of header immediately following the IPv6 header. Uses the same values as the IPv4 Protocol field [RFC-1700 et seq.]
- Hop Limit:
8-bit unsigned integer. Decremented by 1 by each node that forwards the packet. The packet is discarded if Hop Limit is decremented to zero.
- Source Address:
128-bit address of the originator of the packet

- Destination Address:

128-bit address of the intended recipient of the packet (possibly not the ultimate recipient, if a Routing header is present)

IPv4 vs IPv6 Header Comparison [3]

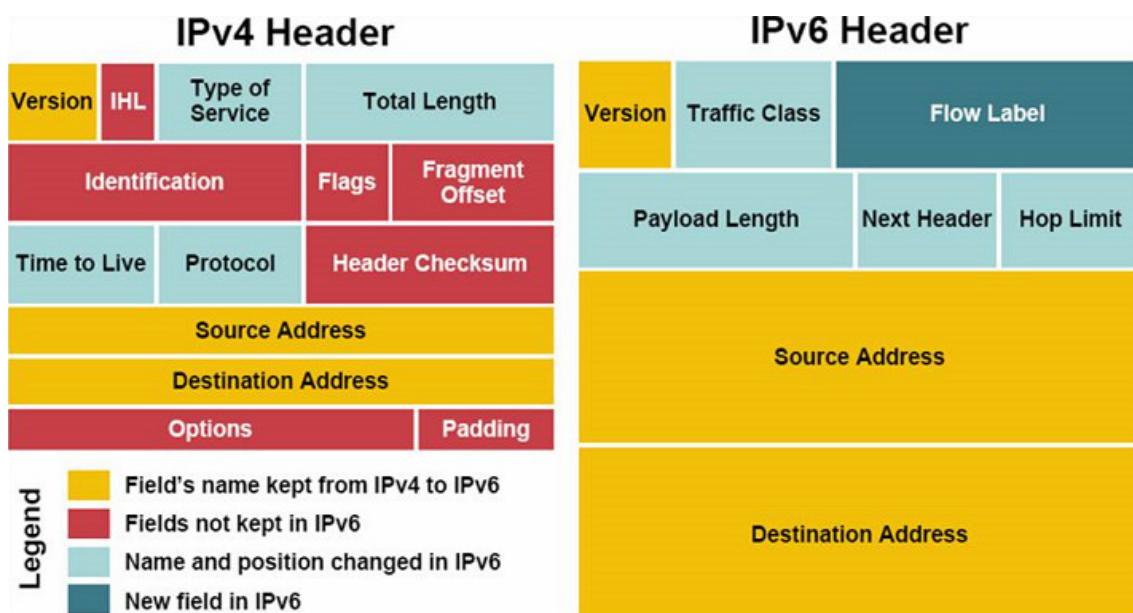


Figure 7.63: Comparison of IPv4 and IPv6 Header Fields

7.57.3 IPv6 Extension Headers

In IPv6, optional internet-layer information is encoded in separate headers that may be placed between the IPv6 header and the upper layer header in a packet. There are a small number of such extension headers, each identified by a distinct Next Header value. As illustrated in these examples, an IPv6 packet may carry zero, one, or more extension headers, each identified by the Next Header field of the preceding header:

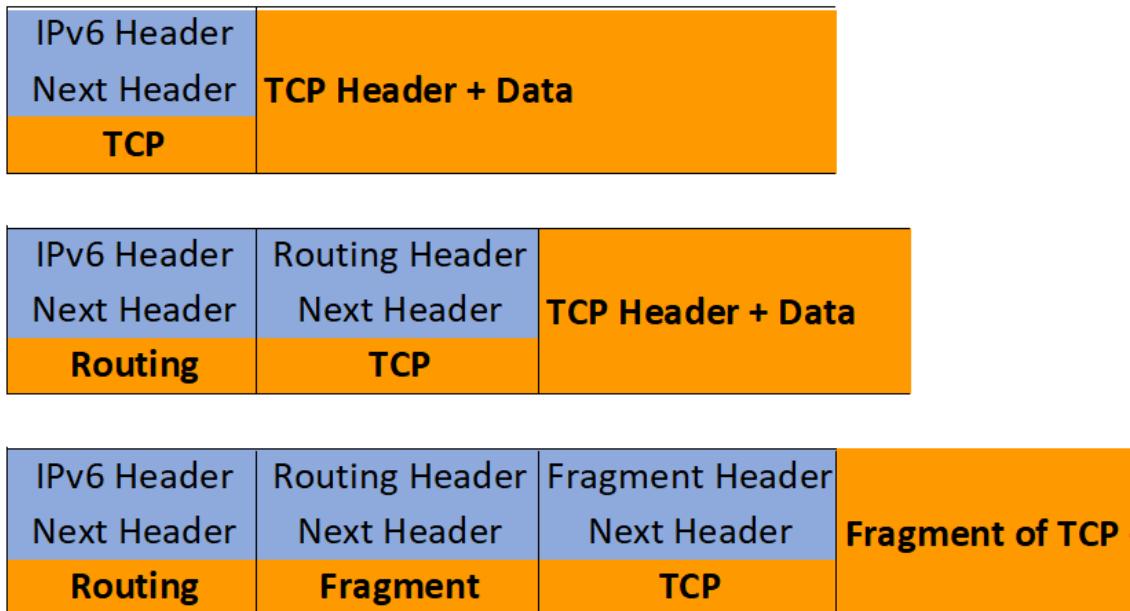


Figure 7.64: IPv6 Extension Headers

With one exception, extension headers are not examined or processed by any node along a packet's delivery path, until the packet reaches the node (or each of the set of nodes, in the case of multicast) identified in the Destination Address field of the IPv6 header. There, normal demultiplexing on the Next Header field of the IPv6 header invokes the module to process the first extension header, or the upper-layer header if no extension header is present. The contents and semantics of each extension header determine whether or not to proceed to the next header. Therefore, extension headers must be processed strictly in the order they appear in the packet; a receiver must not, for example, scan through a packet looking for a particular kind of extension header and process that header prior to processing all preceding ones.

The exception referred to in the preceding paragraph is the Hop-by-Hop Options header, which carries information that must be examined and processed by every node along a packet's delivery path, including the source and destination nodes. The Hop-by-Hop Options header, when present, must immediately follow the IPv6 header. Its presence is indicated by the value zero in the Next Header field of the IPv6 header. If, as a result of processing a header, a node is required to proceed to the next header but the Next Header value in the current header is unrecognized by the node, it should discard the packet and send an ICMP Parameter Problem message to the source of the packet, with an ICMP Code value of 1 ("unrecognized Next Header type encountered") and the ICMP Pointer field containing the offset of the unrecognized value within the original packet. The same action should be taken if a node encounters a Next Header value of zero in any header other than an IPv6 header. Each extension header is an integer multiple of 8 octets long, in order to retain 8-octet alignment for subsequent headers. Multi-octet fields within each extension header are aligned on their natural boundaries, i.e., fields of width n octets are placed at an integer multiple of n octets from the start of the header, for $n = 1, 2, 4$, or 8 . A full implementation of IPv6 includes implementation of the following extension headers:

Hop-by-Hop Options (including Jumbogram [RFC-2675]) Routing (Type 0) Fragment Destination Options Authentication (RFC-2402) Encapsulating Security Payload (RFC-2406)

Extension Header Order

When more than one extension header is used in the same packet, it is recommended that those headers appear in the following order:

- IPv6 header
- Hop-by-Hop Options header
- Destination Options header
- Routing header

- Fragment header
- Authentication header
- Encapsulating Security Payload header
- Destination Options header
- upper-layer header

Extension Header Formats

Hop-by-Hop Options Header

The Hop-by-Hop Options header is used to carry optional information that must be examined by every node along a packet's delivery path. The Hop-by-Hop Options header is identified by a Next Header value of 0 in the IPv6 header, and has the following format:

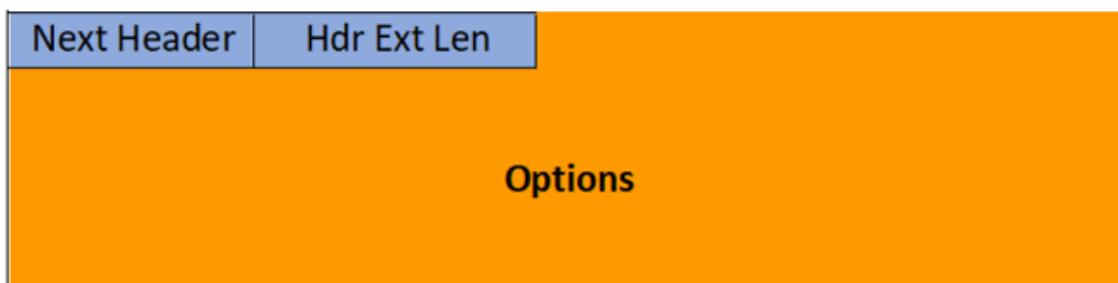


Figure 7.65: IPv6 Options Header

- Next Header 8-bit selector
Identifies the type of header immediately following the Hop-by-Hop Options header. Uses the same values as the IPv4 Protocol field
- Hdr Ext Len 8-bit unsigned integer
Length of the Hop-by-Hop Options header in 8-octet units, not including the first 8 octets.
- Options Variable-length field
Length such that the complete Hop-by-Hop Options header is an integer multiple of 8 octets long. Contains one or more TLV-encoded options

Jumbo Payload Option

The Jumbo Payload option is carried in an IPv6 Hop-by-Hop Options header, immediately following the IPv6 header. This option has an alignment requirement of $4n + 2$. (See [IPv6, Section 4.2] for discussion of option alignment.) The option has the following format:



Figure 7.66: IPv6 Jumbogram Header

- Option Type 8-bit value of 0xC2 (hexadecimal)
- Opt Data Len 8-bit value of 0x04

- Jumbo Payload Length 32-bit unsigned integer

Length of the IPv6 packet in octets, excluding the IPv6 header but including the Hop-by-Hop Options header and any other extension headers present. Must be greater than 65,535.

Routing Options Header

The Routing header is used by an IPv6 source to list one or more intermediate nodes to be "visited" on the way to a packet's destination. This function is very similar to IPv4's Loose Source and Record Route option

The Routing header is identified by a Next Header value of 43 in the immediately preceding header, and has the following format:

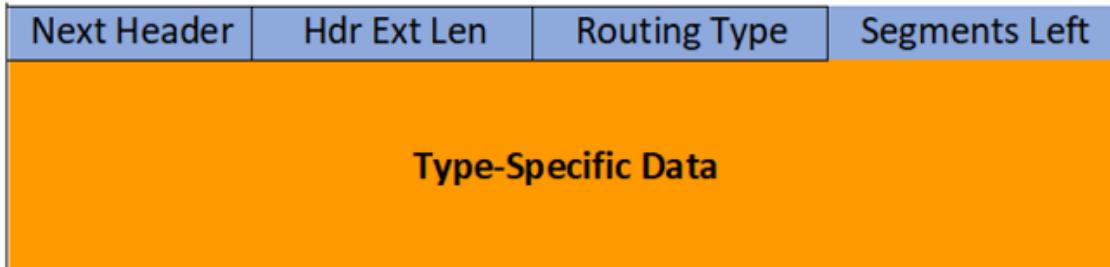


Figure 7.67: IPv6 Routing Header

- Next Header 8-bit selector

Identifies the type of header immediately following the Routing header. Uses the same values as the IPv4 Protocol field

- Hdr Ext Len 8-bit unsigned integer

Length of the Routing header in 8-octet units, not including the first 8 octets.

- Routing Type 8-bit identifier

Identifies a particular Routing header variant

- Segments Left 8-bit unsigned integer

Number of route segments remaining, i.e., number of explicitly listed intermediate nodes still to be visited before reaching the final destination.

- type-specific data Variable-length field

Format determined by the Routing Type, and of length such that the complete Routing header is an integer multiple of 8 octets long

If, while processing a received packet, a node encounters a Routing header with an unrecognized Routing Type value, the required behavior of the node depends on the value of the Segments Left field, as follows:

If Segments Left is zero, the node must ignore the Routing header and proceed to process the next header in the packet, whose type is identified by the Next Header field in the Routing header. If Segments Left is non-zero, the node must discard the packet and send an ICMP Parameter Problem, Code 0, message to the packet's Source Address, pointing to the unrecognized Routing Type. If, after processing a Routing header of a received packet, an intermediate node determines that the packet is to be forwarded onto a link whose link MTU is less than the size of the packet, the node must discard the packet and send an ICMP Packet Too Big message to the packet's Source Address

The Type 0 Routing header has the following format:

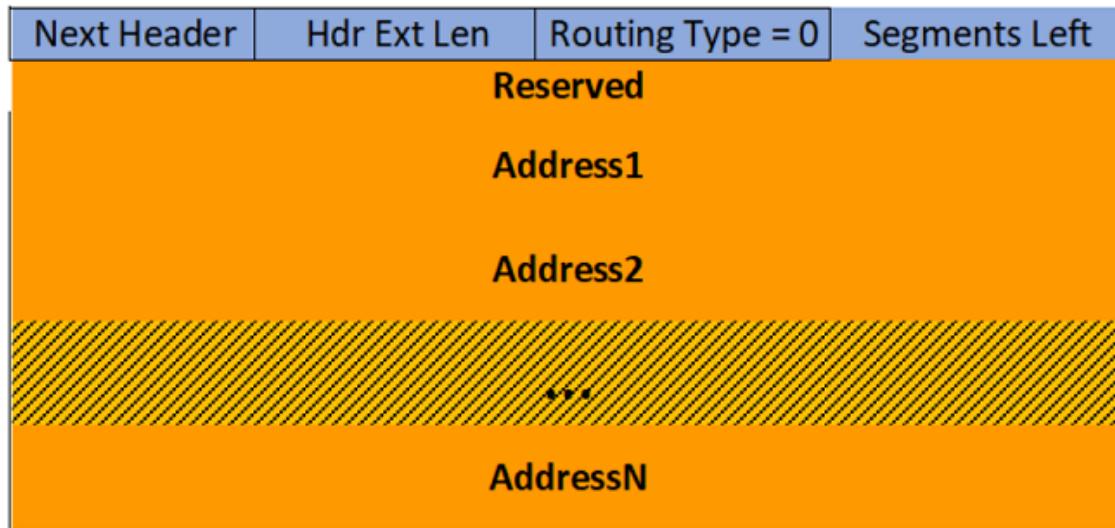


Figure 7.68: IPv6 Routing Type 0 Header

Multicast addresses must not appear in a Routing header of Type 0, or in the IPv6 Destination Address field of a packet carrying a Routing header of Type 0

A Routing header is not examined or processed until it reaches the node identified in the Destination Address field of the IPv6 header. In that node, dispatching on the Next Header field of the immediately preceding header causes the Routing header module to be invoked, which, in the case of Routing Type 0, performs the following algorithm:

```

*   if Segments Left = 0 {
*       proceed to process the next header in the packet, whose type is
*       identified by the Next Header field in the Routing header
*   }
*   else if Hdr Ext Len is odd {
*       send an ICMP Parameter Problem, Code 0, message to the Source
*       Address, pointing to the Hdr Ext Len field, and discard the
*       packet
*   }
*   else {
*       compute n, the number of addresses in the Routing header, by
*       dividing Hdr Ext Len by 2
*
*       if Segments Left is greater than n {
*           send an ICMP Parameter Problem, Code 0, message to the Source
*           Address, pointing to the Segments Left field, and discard the
*           packet
*       }
*       else {
*           decrement Segments Left by 1;
*           compute i, the index of the next address to be visited in
*           the address vector, by subtracting Segments Left from n
*
*           if Address [i] or the IPv6 Destination Address is multicast {
*               discard the packet
*           }
*           else {
*               swap the IPv6 Destination Address and Address[i]
*
*               if the IPv6 Hop Limit is less than or equal to 1 {
*                   send an ICMP Time Exceeded -- Hop Limit Exceeded in
*                   Transit message to the Source Address and discard the
*                   packet
*               }
*               else {
*                   decrement the Hop Limit by 1
*
*                   resubmit the packet to the IPv6 module for transmission
*                   to the new destination
*               }
*           }
*       }
*   }
*
* }
```

Fragment Options Header

The Fragment header is used by an IPv6 source to send a packet larger than would fit in the path MTU to its destination. (Note: unlike IPv4, fragmentation in IPv6 is performed only by source nodes, not by routers along a packet's delivery path – see section 5.) The Fragment header is identified by a Next Header value of 44 in the immediately preceding header, and has the following format:



Figure 7.69: IPv6 Fragmentation Header

- Next Header 8-bit selector

Identifies the initial header type of the Fragmentable Part of the original packet (defined below). Uses the same values as the IPv4 Protocol field

- Reserved 8-bit reserved field

Initialized to zero for transmission; ignored on reception

- Fragment Offset 13-bit unsigned integer

The offset, in 8-octet units, of the data following this header, relative to the start of the Fragmentable Part of the original packet.

- Res 2-bit reserved field.

Initialized to zero for transmission; ignored on reception.

- M flag 1 = more fragments; 0 = last fragment

- Identification 32 bits

Destination Options Header

The Destination Options header is used to carry optional information that need be examined only by a packet's destination node(s). The Destination Options header is identified by a Next Header value of 60 in the immediately preceding header, and has the following format:

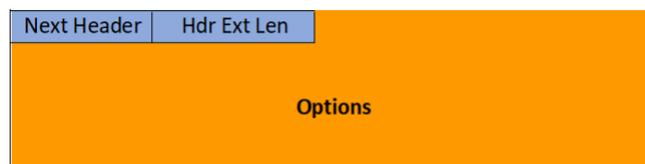


Figure 7.70: IPv6 Destinations Options Header

- Next Header 8-bit selector

Identifies the type of header immediately following the Destination Options header. Uses the same values as the IPv4 Protocol field

- Hdr Ext Len 8-bit unsigned integer

Length of the Destination Options header in 8-octet units, not including the first 8 octets

- Options Variable-length field

Length such that the complete Destination Options header is an integer multiple of 8 octets long. Contains one or more TLV-encoded options, as described in section 4.2

```
[1] http://orm-chimera-prod.s3.amazonaws.com/1230000000545/ch03.html
[2] https://www.petri.com/ipv6-header-vs-ipv4
[3] http://help.mysonicwall.com/sw/eng/5505/ui2/25560/IPv6.html
```

For API Documentation:**See Also**

[ProtocolPP::jprotocol](#)
[ProtocolPP::jicmp](#)
[ProtocolPP::jarray](#)
[ProtocolPP::jenum](#)

For Additional Documentation:**See Also**

[jprotocol](#)
[jicmp](#)
[jarray](#)
[jenum](#)

Protocol++ modified by : John Peter Greninger © John Peter Greninger 2015-2019 All Rights Reserved

All copyrights and trademarks are the property of their respective owners

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer
- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution
- Any and all modifications must be returned to John Peter Greninger at GitHub.com <https://github.com/jpgreninger/protocolpp> for evaluation. Inclusion of modifications in the source code shall be determined solely by John Peter Greninger. Failure to provide modifications shall render this license NULL and VOID and revoke any rights to use of Protocol++
- Commercial use requires a fee-based license obtainable at www.protocolpp.com
- Academic use requires written and notarized permission from John Peter and Sheila Rocha Greninger
- US Copyrights at <https://www.copyright.gov/>
 - TXu002059872 (Version 1.0.0)
 - TXu002066632 (Version 1.2.7)
 - TXu002082674 (Version 1.4.0)
 - TXu002097880 (Version 2.0.0)

The names of its contributors may not be used to endorse or promote products derived from this software without specific prior written permission

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT

OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE

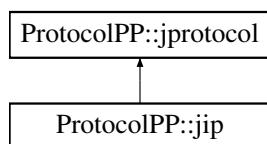
The documentation for this class was generated from the following file:

- [include/jip.h](#)

7.58 ProtocolPP::jip Class Reference

```
#include <jip.h>
```

Inheritance diagram for ProtocolPP::jip:



Public Member Functions

- [jip \(std::shared_ptr< \[jipsa\]\(#\) > &security\)](#)
- [jip \(std::shared_ptr< \[jipsa\]\(#\) > &security, std::string &file\)](#)
- virtual [~jip \(\)](#)
Standard deconstructor.
- void [encap_packet \(std::shared_ptr< \[jarray\]\(#\)< uint8_t >> &output\)](#)
- void [encap_packet \(std::shared_ptr< \[jarray\]\(#\)< uint8_t >> &input, std::shared_ptr< \[jarray\]\(#\)< uint8_t >> &output\)](#)
- void [decap_packet \(std::shared_ptr< \[jarray\]\(#\)< uint8_t >> &input\)](#)
- void [decap_packet \(std::shared_ptr< \[jarray\]\(#\)< uint8_t >> &input, std::shared_ptr< \[jarray\]\(#\)< uint8_t >> &output\)](#)
- void [set_hdr \(\[jarray\]\(#\)< uint8_t > &hdr\)](#)
- void [set_exthdr \(\[jarray\]\(#\)< uint8_t > &exthdr\)](#)
- void [set_field \(field_t field, uint64_t value\)](#)
- [jarray< uint8_t > format_exthdr \(iana_t extension, iana_t nh, \[jarray\]\(#\)< uint8_t > &data, uint16_t type_offset=0, uint8_t segments=0\)](#)
- [jarray< uint8_t > get_hdr \(\)](#)
- [jarray< uint8_t > get_exthdr \(\)](#)
- uint64_t [get_field \(field_t field, \[jarray\]\(#\)< uint8_t > &iphdr\)](#)
- uint64_t [get_field \(field_t field\)](#)
- void [to_xml \(tinyxml2::XMLPrinter &myxml, direction_t direction\)](#)

Additional Inherited Members

7.58.1 Constructor & Destructor Documentation

7.58.1.1 ProtocolPP::jip::jip (std::shared_ptr< [jipsa](#) > & security)

constructor for normal IP

Parameters

<i>security</i>	- parameters necessary to setup and IP flow such as NH, SRC, DST, DSECN, TTL, FLAGS(-IPv4), ID(IPv4), and LABEL(IPv6)
-----------------	---

7.58.1.2 `ProtocolPP::jip::jip (std::shared_ptr<jipsa> &security, std::string &file)`

constructor for normal IP with file containing input data

Parameters

<i>security</i>	- parameters necessary to setup and IP flow such as NH, SRC, DST, DSECN, TTL, FLAGS(-IPv4), ID(IPv4), and LABEL(IPv6)
<i>file</i>	- File containing input data

7.58.1.3 `virtual ProtocolPP::jip::~jip () [inline], [virtual]`

Standard deconstructor.

7.58.2 Member Function Documentation

7.58.2.1 `void ProtocolPP::jip::decap_packet (std::shared_ptr<jarray<uint8_t>> &input) [virtual]`

This function is for use with the constructor with a file handle. Decap will produce a payload from the packet passed in and write it to the output file

Parameters

<i>input</i>	- IP encapsulated packet to decapsulate
--------------	---

Reimplemented from [ProtocolPP::jprotocol](#).

7.58.2.2 `void ProtocolPP::jip::decap_packet (std::shared_ptr<jarray<uint8_t>> &input, std::shared_ptr<jarray<uint8_t>> &output) [virtual]`

This function is for use with the constructor without a file handle. Decap will produce a payload from the packet passed in.

Parameters

<i>input</i>	- IP encapsulated packet to decapsulate
<i>output</i>	- decapsulated payload

Implements [ProtocolPP::jprotocol](#).

7.58.2.3 `void ProtocolPP::jip::encap_packet (std::shared_ptr<jarray<uint8_t>> &output) [virtual]`

This function is for use with the constructor with a file handle. Encap will produce a packet with a payload from the file handle passed in

Parameters

<i>output</i>	- IP encapsulated packet
---------------	--------------------------

Reimplemented from [ProtocolPP::jprotocol](#).

```
7.58.2.4 void ProtocolPP::jip::encap_packet( std::shared_ptr<jarray< uint8_t >> & input, std::shared_ptr<jarray< uint8_t >> & output ) [virtual]
```

This function is for use with the constructor without a file handle. Encap will produce a packet with the payload passed in

Parameters

<i>input</i>	- payload to encapsulate with IP
<i>output</i>	- IP encapsulated packet

Implements [ProtocolPP::jprotocol](#).

7.58.2.5 `jarray<uint8_t> ProtocolPP::jip::format_exthdr (iana_t extension, iana_t nh, jarray<uint8_t> & data, uint16_t type_offset = 0, uint8_t segments = 0)`

Adds extension headers to the packet

Parameters

<i>extension</i>	- extension type to add to IP header
<i>nh</i>	- Next Header (NH) for this extension header
<i>data</i>	- Data necessary for the extension header (32-bit length for JUMBOGRAM, Route header, Identification for Fragment, etc.)
<i>type_offset</i>	- Type (8-bits) for Routing header, offset (13-bits) for Fragment header
<i>segments</i>	- Number of segments left for Routing header (8-bits), "More" bit for Fragment header (1-bit)

Returns

the formatted extension header

7.58.2.6 `jarray<uint8_t> ProtocolPP::jip::get_exthdr ()`

Returns the extension header

Returns

current extension header

7.58.2.7 `uint64_t ProtocolPP::jip::get_field (field_t field, jarray<uint8_t> & iphdr) [virtual]`

Returns the version field of the IP header

Parameters

<i>field</i>	- field to retrieve from the IP header
<i>iphdr</i>	- header to retrieve the field from

Returns

version field of the IP header

Implements [ProtocolPP::jprotocol](#).

7.58.2.8 `uint64_t ProtocolPP::jip::get_field (field_t field) [virtual]`

Returns the field from the security association

Parameters

<i>field</i>	- field to retrieve from the security association
--------------	---

Returns

field from the security association

Reimplemented from [ProtocolPP::jprotocol](#).

7.58.2.9 `jarray<uint8_t> ProtocolPP::jip::get_hdr() [virtual]`

Returns the complete IP header

Returns

current IP header

Implements [ProtocolPP::jprotocol](#).

7.58.2.10 `void ProtocolPP::jip::set_exthdr(jarray< uint8_t > & exthdr)`

Allows the user to update the IP extension header with their own header

Parameters

<i>exthdr</i>	- new extension header to use
---------------	-------------------------------

7.58.2.11 `void ProtocolPP::jip::set_field(field_t field, uint64_t value) [virtual]`

Allows the user to update the field of the IP header

Parameters

<i>field</i>	- field to update the IP header with
<i>value</i>	- value to update the IP header with

Implements [ProtocolPP::jprotocol](#).

7.58.2.12 `void ProtocolPP::jip::set_hdr(jarray< uint8_t > & hdr) [virtual]`

Allows the user to update the IP header with their own header Note : This function will NOT save the previous header. If it is desired to keep the current header the user would need to use the `get_iphdr()` function and save the current header before updating the IP header with a custom header. The header would then need to be restored when the custom header is no longer needed

Parameters

<i>hdr</i>	- new IP header to use
------------	------------------------

Implements [ProtocolPP::jprotocol](#).

7.58.2.13 `void ProtocolPP::jip::to_xml(tinyxml2::XMLPrinter & myxml, direction_t direction) [virtual]`

Prints the protocol object in XML

Parameters

<i>myxml</i>	- XMLPrinter object to print to
<i>direction</i>	- randomization

Implements [ProtocolPP::jprotocol](#).

The documentation for this class was generated from the following file:

- [include/jip.h](#)

7.59 jipsa Class Reference

```
#include "include/jipsa.h"
```

7.59.1 Detailed Description

7.59.2 Internet Protocol (IP) Security Association

See https://en.wikipedia.org/wiki/Internet_Protocol

The Internet Protocol (IP) is the principal communications protocol in the Internet protocol suite for relaying datagrams across network boundaries. Its routing function enables internetworking, and essentially establishes the Internet.

IP has the task of delivering packets from the source host to the destination host solely based on the IP addresses in the packet headers. For this purpose, IP defines packet structures that encapsulate the data to be delivered. It also defines addressing methods that are used to label the datagram with source and destination information.

Historically, IP was the connectionless datagram service in the original Transmission Control Program introduced by Vint Cerf and Bob Kahn in 1974; the other being the connection-oriented Transmission Control Protocol (TCP). The Internet protocol suite is therefore often referred to as TCP/IP.

The first major version of IP, Internet Protocol Version 4 (IPv4), is the dominant protocol of the Internet. Its successor is Internet Protocol V6

The header for IPv4 is configured as follows (See RFC 791) [1]

Bit	+0..7		+8..15		+16..23		+24..31							
0	Version	Header Length	DSCP	ECN	Total Length									
32	Identification				Flags	Fragment Offset								
64	Time To Live		Protocol		Header Checksum									
96	Source IP Address													
128	Destination IP Address													
160	Options (if present)													
...	Payload													

Figure 7.71: IP Version 4 Header

- Version: 4 bits

The Version field indicates the format of the internet header

- IHL: 4 bits

Internet Header Length is the length of the internet header in 32 bit words, and thus points to the beginning of the data. Note that the minimum value for a correct header is 5.

- Type of Service: 8 bits

The Type of Service provides an indication of the abstract parameters of the quality of service desired. These parameters are to be used to guide the selection of the actual service parameters when transmitting a datagram through a particular network. Several networks offer service precedence, which somehow treats high precedence traffic as more important than other traffic (generally by accepting only traffic above a certain precedence at time of high load). The major choice is a three way tradeoff between low-delay, high-reliability, and high-throughput.

```

* Bits 0-2: Precedence
* Bit 3: 0 = Normal Delay, 1 = Low Delay
* Bit 4: 0 = Normal Throughput, 1 = High Throughput
* Bit 5: 0 = Normal Reliability, 1 = High Reliability
* Bits 6-7: Reserved for Future Use.

*
*      0   1   2   3   4   5   6   7
* +---+---+---+---+---+---+---+
* |   |   |   |   |   |   |   |
* | PRECEDENCE | D | T | R | 0 | 0 | 0 | 0
* |   |   |   |   |   |   |   |
* +---+---+---+---+---+---+---+
*
* Precedence
*
*    111 - Network Control
*    110 - Internetwork Control
*    101 - CRITIC/ECP
*    100 - Flash Override
*    011 - Flash
*    010 - Immediate
*    001 - Priority
*    000 - Routine
*
```

The use of the Delay, Throughput, and Reliability indications may increase the cost (in some sense) of the service. In many networks better performance for one of these parameters is coupled with worse performance on another. Except for very unusual cases at most two of these three indications should be set.

The type of service is used to specify the treatment of the datagram during its transmission through the internet system. Example mappings of the internet type of service to the actual service provided on networks such as AUTODIN II, ARPANET, SATNET, and PRNET is given in "Service Mappings"

The Network Control precedence designation is intended to be used within a network only. The actual use and control of that designation is up to each network. The Internetwork Control designation is intended for use by gateway control originators only. If the actual use of these precedence designations is of concern to a particular network, it is the responsibility of that network to control the access to, and use of, those precedence designations.

- Total Length: 16 bits

Total Length is the length of the datagram, measured in octets, including internet header and data. This field allows the length of a datagram to be up to 65,535 octets. Such long datagrams are impractical for most hosts and networks. All hosts must be prepared to accept datagrams of up to 576 octets (whether they arrive whole or in fragments). It is recommended that hosts only send datagrams larger than 576 octets if they have assurance that the destination is prepared to accept the larger datagrams.

The number 576 is selected to allow a reasonable sized data block to be transmitted in addition to the required header information. For example, this size allows a data block of 512 octets plus 64 header octets to fit in a datagram. The maximal internet header is 60 octets, and a typical internet header is 20 octets, allowing a margin for headers of higher level protocols.

- Identification: 16 bits

An identifying value assigned by the sender to aid in assembling the fragments of a datagram.

- Flags: 3 bits

Various Control Flags Bit 0: reserved, must be zero Bit 1: (DF) 0 = May Fragment, 1 = Don't Fragment Bit 2: (MF) 0 = Last Fragment, 1 = More Fragments

```

*      0   1   2
*      +---+---+---+
*      |   | D | M |
*      | 0 | F | F |
*      +---+---+---+
*

```

- Fragment Offset: 13 bits

This field indicates where in the datagram this fragment belongs. The fragment offset is measured in units of 8 octets (64 bits). The first fragment has offset zero.

- Time to Live: 8 bits

This field indicates the maximum time the datagram is allowed to remain in the internet system. If this field contains the value zero, then the datagram must be destroyed. This field is modified in internet header processing. The time is measured in units of seconds, but since every module that processes a datagram must decrease the TTL by at least one even if it processes the datagram in less than a second, the TTL must be thought of only as an upper bound on the time a datagram may exist. The intention is to cause undeliverable datagrams to be discarded, and to bound the maximum datagram lifetime.

- Protocol: 8 bits

This field indicates the next level protocol used in the data portion of the internet datagram. The values for various protocols are specified in "Assigned Numbers"

- Header Checksum: 16 bits

A checksum on the header only. Since some header fields change (e.g., time to live), this is recomputed and verified at each point that the internet header is processed. The checksum algorithm is:

The checksum field is the 16 bit one's complement of the one's complement sum of all 16 bit words in the header. For purposes of computing the checksum, the value of the checksum field is zero.

This is a simple to compute checksum and experimental evidence indicates it is adequate, but it is provisional and may be replaced by a CRC procedure, depending on further experience.

- Source Address: 32 bits

The source address. See section 3.2.

- Destination Address: 32 bits

The destination address. See section 3.2.

The header for IPv6 is configured as follows (see RFC 2460) [2]

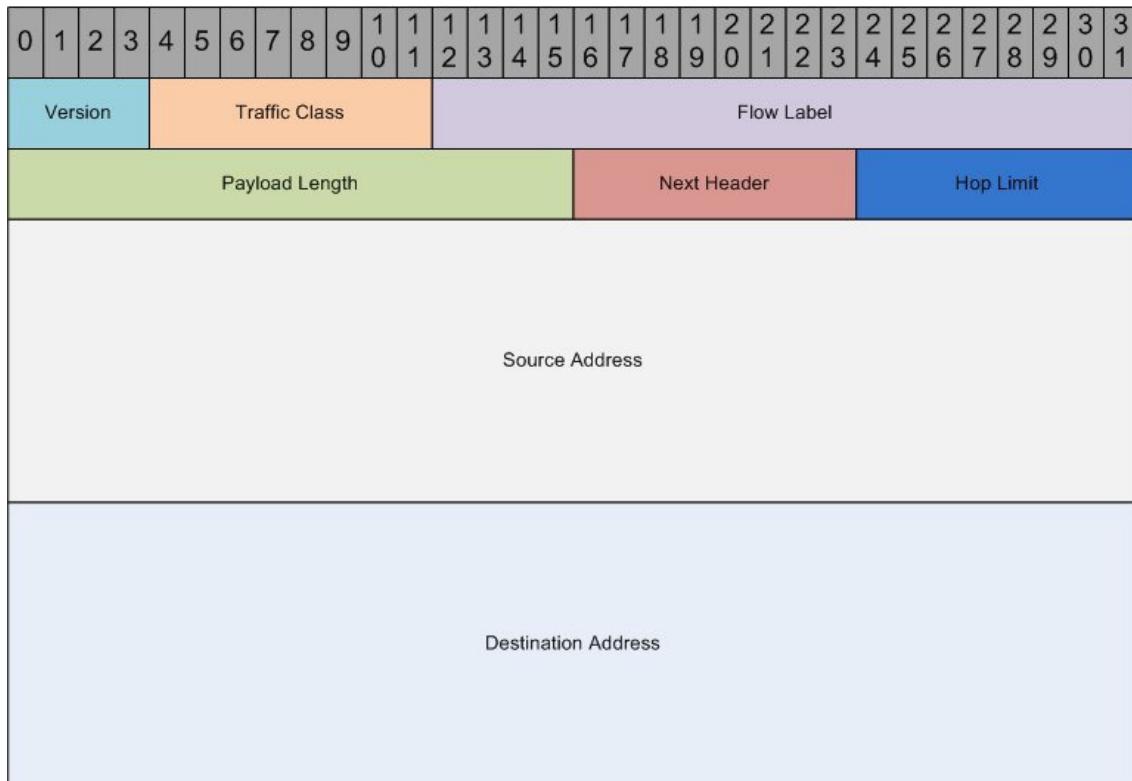


Figure 7.72: IP Version 6 Header

- **Version:**
4-bit Internet Protocol version number = 6.
- **Traffic Class:**
8-bit traffic class field. See section 7.
- **Flow Label:**
20-bit flow label. See section 6.
- **Payload Length:**
16-bit unsigned integer. Length of the IPv6 payload, i.e., the rest of the packet following this IPv6 header, in octets. (Note that any extension headers present are considered part of the payload, i.e., included in the length count.)
- **Next Header:**
8-bit selector. Identifies the type of header immediately following the IPv6 header. Uses the same values as the IPv4 Protocol field [RFC-1700 et seq.]
- **Hop Limit:**
8-bit unsigned integer. Decremented by 1 by each node that forwards the packet. The packet is discarded if Hop Limit is decremented to zero.
- **Source Address:**
128-bit address of the originator of the packet
- **Destination Address:**
128-bit address of the intended recipient of the packet (possibly not the ultimate recipient, if a Routing header is present)

IPv4 vs IPv6 Header Comparison [3]

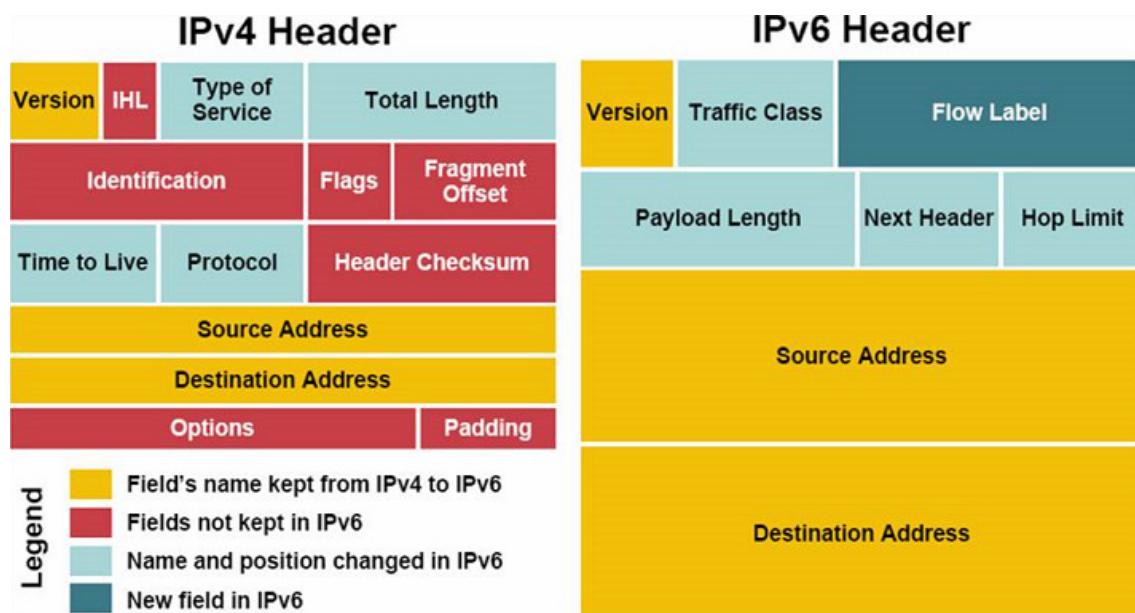


Figure 7.73: Comparison of IPv4 and IPv6 Header Fields

For API Documentation:

See Also

[ProtocolPP::jsecass](#)
[ProtocolPP::jprotocol](#)
[ProtocolPP::jicmp](#)
[ProtocolPP::jarray](#)
[ProtocolPP::jenum](#)

For Additional Documentation:

See Also

[jsecass](#)
[jprotocol](#)
[jicmp](#)
[jarray](#)
[jenum](#)

Protocol++ modified by : John Peter Greninger © John Peter Greninger 2015-2019 All Rights Reserved

All copyrights and trademarks are the property of their respective owners

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer
- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution

- Any and all modifications must be returned to John Peter Greninger at GitHub.com <https://github.com/jpgreninger/protocolpp> for evaluation. Inclusion of modifications in the source code shall be determined solely by John Peter Greninger. Failure to provide modifications shall render this license NULL and VOID and revoke any rights to use of Protocol++
- Commercial use requires a fee-based license obtainable at www.protocolpp.com
- Academic use requires written and notarized permission from John Peter and Sheila Rocha Greninger
- US Copyrights at [https://www.copyright.gov/](http://www.copyright.gov/)
 - TXU002059872 (Version 1.0.0)
 - TXU002066632 (Version 1.2.7)
 - TXU002082674 (Version 1.4.0)
 - TXU002097880 (Version 2.0.0)

The names of its contributors may not be used to endorse or promote products derived from this software without specific prior written permission

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE

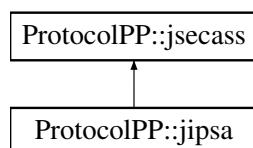
The documentation for this class was generated from the following file:

- [include/jipsa.h](#)

7.60 ProtocolPP::jipsa Class Reference

```
#include <jipsa.h>
```

Inheritance diagram for ProtocolPP::jipsa:



Public Member Functions

- [jipsa \(\)](#)
- [jipsa \(direction_t dir, iana_t version, iana_t nh, jarray< uint8_t > src, jarray< uint8_t > dst, jarray< uint8_t > exthdr, uint8_t dsecn, uint8_t ttl, uint8_t flags, uint16_t fragoff, uint16_t id, uint32_t label, bool jumbogram, unsigned int mtu\)](#)
- [jipsa \(jipsa &rhs\)](#)
 - standard copy constructor*
- [jipsa \(std::shared_ptr< jipsa > &rhs\)](#)
 - standard copy constructor from shared pointer*
- [virtual ~jipsa \(\)](#)

Standard deconstructor.

- template<typename T >
void **set_field** (**field_t** field, T value)
- template<typename T >
T **get_field** (**field_t** field)
- void **to_xml** (tinyxml2::XMLPrinter &myxml, **direction_t** direction)

7.60.1 Constructor & Destructor Documentation

7.60.1.1 ProtocolPP::jipsa::jipsa ()

Standard constructor with defaults

```
*   jipsa snd;
*
*   snd.set_field<iana_t>(field_t::NH, iana_t::UDP);
*   snd.set_field<uint32_t>(field_t::MTU, 390);
*   snd.set_field<bool>(field_t::JUMBOGRAM, false);
*
```

7.60.1.2 ProtocolPP::jipsa::jipsa (**direction_t dir**, **iana_t version**, **iana_t nh**, **jarray< uint8_t > src**, **jarray< uint8_t > dst**, **jarray< uint8_t > exthdr**, **uint8_t dsecn**, **uint8_t ttl**, **uint8_t flags**, **uint16_t fragoff**, **uint16_t id**, **uint32_t label**, **bool jumbogram**, **unsigned int mtu**)

Security Association for IP

Required fields for IP

Parameters

dir	- Direction of data flow
version	- Version of IP to use (IPv4, IPv6)
nh	- Next header of the extension header or payload
src	- Source address (4 bytes for IPv4, 16 bytes for IPv6)
dst	- Destination address (4 bytes for IPv4, 16 bytes for IPv6)
exthdr	- Extension header to add to standard header
dsecn	- Differentiated services and Congestion bits
ttl	- Time-To-Live / Hop Limit
jumbogram	- Nodal support for IPv6 JUMBOGRAMs
mtu	- if using a file for input data, size of the payload

Required IPv4 fields

Parameters

flags	- Flags for the IPv4 header
fragoff	- Fragment offset
id	- ID field for IPv4 header

Required IPv6 fields

Parameters

label	- Label field
--------------	---------------

7.60.1.3 ProtocolPP::jipsa::jipsa (**jipsa & rhs**)

standard copy constructor

7.60.1.4 `ProtocolPP::jipsa::jipsa (std::shared_ptr<jipsa> &rhs)`

standard copy constructor from shared pointer

7.60.1.5 `virtual ProtocolPP::jipsa::~jipsa () [inline], [virtual]`

Standard deconstructor.

7.60.2 Member Function Documentation

7.60.2.1 `template<typename T> T ProtocolPP::jipsa::get_field (field_t field)`

Returns the version field of the IP security association

Due to their dynamic nature, some fields are only available in jip which include the following fields

- LENGTH
- CHECKSUM

Parameters

<code>field</code>	- field to retrieve from the IP security association
--------------------	--

Returns

field of the IP security association

7.60.2.2 `template<typename T> void ProtocolPP::jipsa::set_field (field_t field, T value)`

Allows the user to update the field of the IP security association

Due to their dynamic nature, some fields are only available in jip which include the following fields

- LENGTH
- CHECKSUM

Parameters

<code>field</code>	- field to update the IP security association
<code>value</code>	- value to update the IP security association

7.60.2.3 `void ProtocolPP::jipsa::to_xml (tinyxml2::XMLPrinter &myxml, direction_t direction) [virtual]`

Prints the protocol object in XML

Parameters

<code>myxml</code>	- XMLPrinter object to print to
<code>direction</code>	- randomzation

Implements [ProtocolPP::jsecass](#).

The documentation for this class was generated from the following file:

- [include/jipsa.h](#)

7.61 jipsec Class Reference

```
#include "include/jipsec.h"
```

7.61.1 Detailed Description

7.61.2 Encapsulating Security Payload (ESP)

(See RFC4303 for complete details)

The (outer) protocol header (IPv4, IPv6, or Extension) that immediately precedes the ESP header SHALL contain the value 50 in its Protocol (IPv4) or Next Header (IPv6, Extension) field (see IANA web page at <http://www.iana.org/assignments/protocol-numbers>). Figure 1 illustrates the top-level format of an ESP packet. The packet begins with two 4-byte fields (Security Parameters Index (SPI) and Sequence Number). Following these fields is the Payload Data, which has substructure that depends on the choice of encryption algorithm and mode, and on the use of TFC padding, which is examined in more detail later. Following the Payload Data are Padding and Pad Length fields, and the Next Header field. The optional Integrity Check Value (ICV) field completes the packet.

```
*
* 0           1           2           3
* 0 1 2 3 4 5 6 7 0 1 2 3 4 5 6 7 0 1 2 3 4 5 6 7
* ++++++-----+-----+-----+-----+-----+-----+-----+
* |           Security Parameters Index (SPI)          | ^Int.
* ++++++-----+-----+-----+-----+-----+-----+-----+
* |           Sequence Number                   | ered
* ++++++-----+-----+-----+-----+-----+-----+-----+
* |           Payload Data* (variable)           |   ^
* ~
* |
* |           |Conf.
* +           +-+-----+-----+-----+-----+-----+-----+
* |           | Padding (0-255 bytes)           | ered*
* +-----+-----+-----+-----+-----+-----+-----+
* |           | Pad Length      | Next Header | v   v
* +-----+-----+-----+-----+-----+-----+-----+
* |           Integrity Check Value-ICV (variable)   |
* ~
* |
* +-----+-----+-----+-----+-----+-----+-----+
*
*           Figure 1. Top-Level Format of an ESP Packet
*
* * If included in the Payload field, cryptographic synchronization
*   data, e.g., an Initialization Vector (IV, see Section 2.3),
*   usually is not encrypted per se, although it often is referred
*   to as being part of the ciphertext.
*
```

The (transmitted) ESP trailer consists of the Padding, Pad Length, and Next Header fields. Additional, implicit ESP trailer data (which is not transmitted) is included in the integrity computation, as described below.

If the integrity service is selected, the integrity computation encompasses the SPI, Sequence Number, Payload Data, and the ESP trailer (explicit and implicit).

If the confidentiality service is selected, the ciphertext consists of the Payload Data (except for any cryptographic synchronization data that may be included) and the (explicit) ESP trailer.

As noted above, the Payload Data may have substructure. An encryption algorithm that requires an explicit Initialization Vector (IV), e.g., Cipher Block Chaining (CBC) mode, often prefixes the Payload Data to be protected with that value. Some algorithm modes combine encryption and integrity into a single operation; this document refers to such algorithm modes as "combined mode algorithms". Accommodation of combined mode algorithms requires that the algorithm explicitly describe the payload substructure used to convey the integrity data.

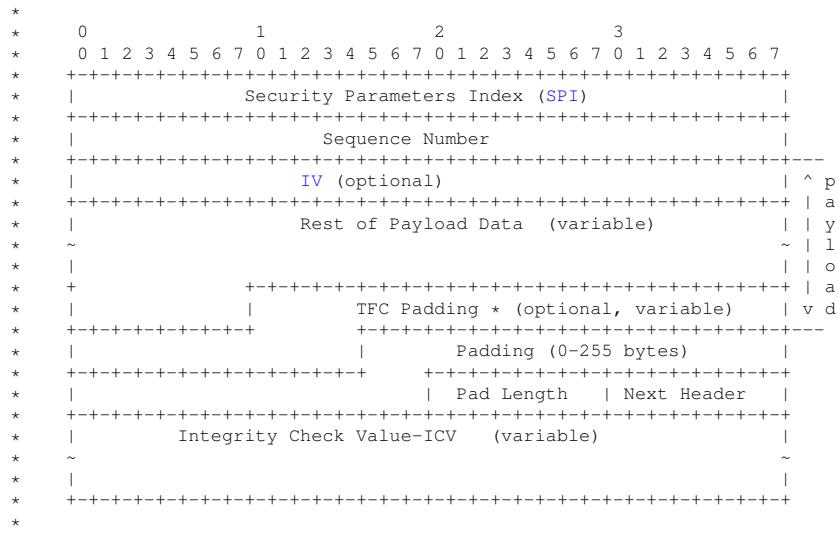
Some combined mode algorithms provide integrity only for data that is

encrypted, whereas others can provide integrity for some additional data that is not encrypted for transmission. Because the SPI and Sequence Number fields require integrity as part of the integrity service, and they are not encrypted, it is necessary to ensure that they are afforded integrity whenever the service is selected, regardless of the style of combined algorithm mode employed.

When any combined mode algorithm is employed, the algorithm itself is expected to return both decrypted plaintext and a pass/fail indication for the integrity check. For combined mode algorithms, the ICV that would normally appear at the end of the ESP packet (when integrity is selected) may be omitted. When the ICV is omitted and integrity is selected, it is the responsibility of the combined mode algorithm to encode within the Payload Data an ICV-equivalent means of verifying the integrity of the packet.

If a combined mode algorithm offers integrity only to data that is encrypted, it will be necessary to replicate the SPI and Sequence Number as part of the Payload Data.

Finally, a new provision is made to insert padding for traffic flow confidentiality after the Payload Data and before the ESP trailer. Figure 2 illustrates this substructure for Payload Data. (Note: This diagram shows bits-on-the-wire. So even if extended sequence numbers are being used, only 32 bits of the Sequence Number will be transmitted (see Section 2.2.1).)



If a combined algorithm mode is employed, the explicit ICV shown in Figures 1 and 2 may be omitted (see Section 3.3.2.2 below). Because algorithms and modes are fixed when an SA is established, the detailed format of ESP packets for a given SA (including the Payload Data substructure) is fixed, for all traffic on the SA.

The tables below refer to the fields in the preceding figures and illustrate how several categories of algorithmic options, each with a different processing model, affect the fields noted above. The processing details are described in later sections.

* Table 1. Separate Encryption and Integrity Algorithms

	# of bytes	Requ'd [1]	What Encrypt Covers	What Integ Covers	What Xmt'd
SPI	4	M		Y plain	
Seq# (low-order bits)	4	M		Y plain	p

```

*          ----- a
*  IV      variable   O      Y      plain    | y
*  IP datagram [2] variable   M or D  Y      cipher[3] |-l
*  TFC padding [4] variable   O      Y      cipher[3] | o
*          ----- a
*  Padding      0-255     M      Y      cipher[3] d
*  Pad Length    1       M      Y      cipher[3]
*  Next Header    1       M      Y      cipher[3]
*  Seq# (high-order bits) 4      if ESN [5] Y      not xmtd
*  ICV Padding   variable if need  Y      not xmtd
*  ICV         variable   M [6]      plain
*
*          [1] M = mandatory; O = optional; D = dummy
*          [2] If tunnel mode -> IP datagram
*                 If transport mode -> next header and data
*          [3] ciphertext if encryption has been selected
*          [4] Can be used only if payload specifies its "real" length
*          [5] See section 2.2.1
*          [6] mandatory if a separate integrity algorithm is used
*
*
*
*          Table 2. Combined Mode Algorithms
*
*          What      What      What
*          # of      Requ'd   Encrypt  Integ  is
*          bytes    [1]      Covers  Covers Xmtd
*          -----  -----  -----  -----  -----
*  SPI        4       M      plain
*  Seq# (low-order bits) 4       M      plain p
*          --- a
*  IV         variable   O      plain | y
*  IP datagram [2] variable   M or D  Y      cipher |-l
*  TFC padding [3] variable   O      Y      cipher | o
*          --- a
*  Padding      0-255     M      cipher d
*  Pad Length    1       M      cipher
*  Next Header    1       M      cipher
*  Seq# (high-order bits) 4      if ESN [4] Y      [5]
*  ICV Padding   variable if need  Y      [5]
*  ICV         variable   O [6]      plain
*
*          [1] M = mandatory; O = optional; D = dummy
*          [2] If tunnel mode -> IP datagram
*                 If transport mode -> next header and data
*          [3] Can be used only if payload specifies its "real" length
*          [4] See Section 2.2.1
*          [5] The algorithm choices determines whether these are
*               transmitted, but in either case, the result is invisible
*               to ESP
*          [6] The algorithm spec determines whether this field is
*               present
*

```

The following subsections describe the fields in the header format. "Optional" means that the field is omitted if the option is not selected, i.e., it is present in neither the packet as transmitted nor as formatted for computation of an ICV (see Section 2.7). Whether or not an option is selected is determined as part of Security Association (SA) establishment. Thus, the format of ESP packets for a given SA is fixed, for the duration of the SA. In contrast, "mandatory" fields are always present in the ESP packet format, for all SAs.

Note: All of the cryptographic algorithms used in IPsec expect their input in canonical network byte order (see Appendix of RFC 791 [Pos81]) and generate their output in canonical network byte order. IP packets are also transmitted in network byte order.

ESP does not contain a version number, therefore if there are concerns about backward compatibility, they MUST be addressed by using a signaling mechanism between the two IPsec peers to ensure compatible versions of ESP (e.g., Internet Key Exchange (IKEv2) [Kau05]) or an out-of-band configuration mechanism.

Security Parameters Index (SPI)

The SPI is an arbitrary 32-bit value that is used by a receiver to identify the SA to which an incoming packet is bound. The SPI field is mandatory.

For a unicast SA, the SPI can be used by itself to specify an SA, or it may be used in conjunction with the IPsec protocol type (in this case ESP). Because the SPI value is generated by the receiver for a unicast SA, whether the value is sufficient to identify an SA by itself or whether it must be used in conjunction with the IPsec protocol

value is a local matter. This mechanism for mapping inbound traffic to unicast SAs MUST be supported by all ESP implementations.

If an IPsec implementation supports multicast, then it MUST support multicast SAs using the algorithm below for mapping inbound IPsec datagrams to SAs. Implementations that support only unicast traffic need not implement this de-multiplexing algorithm.

In many secure multicast architectures (e.g., [RFC3740]), a central Group Controller/Key Server unilaterally assigns the group security association's SPI. This SPI assignment is not negotiated or coordinated with the key management (e.g., IKE) subsystems that reside in the individual end systems that comprise the group. Consequently, it is possible that a group security association and a unicast security association can simultaneously use the same SPI. A multicast-capable IPsec implementation MUST correctly de-multiplex inbound traffic even in the context of SPI collisions.

Each entry in the Security Association Database (SAD) [Ken-Arch] must indicate whether the SA lookup makes use of the destination, or destination and source, IP addresses, in addition to the SPI. For multicast SAs, the protocol field is not employed for SA lookups. For each inbound, IPsec-protected packet, an implementation must conduct its search of the SAD such that it finds the entry that matches the "longest" SA identifier. In this context, if two or more SAD entries match based on the SPI value, then the entry that also matches based on destination, or destination and source, address comparison (as indicated in the SAD entry) is the "longest" match. This implies a logical ordering of the SAD search as follows:

1. Search the SAD for a match on {SPI, destination address, source address}. If an SAD entry matches, then process the inbound ESP packet with that matching SAD entry. Otherwise, proceed to step 2.
2. Search the SAD for a match on {SPI, destination address}. If the SAD entry matches, then process the inbound ESP packet with that matching SAD entry. Otherwise, proceed to step 3.
3. Search the SAD for a match on only {SPI} if the receiver has chosen to maintain a single SPI space for AH and ESP, or on {SPI, protocol} otherwise. If an SAD entry matches, then process the inbound ESP packet with that matching SAD entry. Otherwise, discard the packet and log an auditable event.

In practice, an implementation MAY choose any method to accelerate this search, although its externally visible behavior MUST be functionally equivalent to having searched the SAD in the above order. For example, a software-based implementation could index into a hash table by the SPI. The SAD entries in each hash table bucket's linked list are kept sorted to have those SAD entries with the longest SA identifiers first in that linked list. Those SAD entries having the shortest SA identifiers are sorted so that they are the last entries in the linked list. A hardware-based implementation may be able to effect the longest match search intrinsically, using commonly available Ternary Content-Addressable Memory (TCAM) features.

The indication of whether source and destination address matching is required to map inbound IPsec traffic to SAs MUST be set either as a side effect of manual SA configuration or via negotiation using an SA management protocol, e.g., IKE or Group Domain of Interpretation (GDOI) [RFC3547]. Typically, Source-Specific Multicast (SSM) [HC03] groups use a 3-tuple SA identifier composed of an SPI, a destination multicast address, and source address. An Any-Source Multicast group SA requires only an SPI and a destination multicast address as an identifier.

The set of SPI values in the range 1 through 255 are reserved by the Internet Assigned Numbers Authority (IANA) for future use; a reserved SPI value will not normally be assigned by IANA unless the use of the assigned SPI value is specified in an RFC. The SPI value of zero (0) is reserved for local, implementation-specific use and MUST NOT be sent on the wire. (For example, a key management implementation might use the zero SPI value to mean "No Security Association Exists" during the period when the IPsec implementation has requested that its key management entity establish a new SA, but the SA has not yet been established.)

Sequence Number

This unsigned 32-bit field contains a counter value that increases by one for each packet sent, i.e., a per-SA packet sequence number. For a unicast SA or a single-sender multicast SA, the sender MUST increment this field for every transmitted packet. Sharing an SA among multiple senders is permitted, though generally not recommended. ESP provides no means of synchronizing packet counters among multiple senders or meaningfully managing a receiver packet counter and window in the context of multiple senders. Thus, for a multi-sender SA, the anti-replay features of ESP are not available (see Sections 3.3.3 and 3.4.3.)

The field is mandatory and MUST always be present even if the receiver does not elect to enable the anti-replay service for a specific SA. Processing of the Sequence Number field is at the discretion of the receiver, but all ESP

implementations MUST be capable of performing the processing described in Sections 3.3.3 and 3.4.3. Thus, the sender MUST always transmit this field, but the receiver need not act upon it (see the discussion of Sequence Number Verification in the "Inbound Packet Processing" section (3.4.3) below).

The sender's counter and the receiver's counter are initialized to 0 when an SA is established. (The first packet sent using a given SA will have a sequence number of 1; see Section 3.3.3 for more details on how the sequence number is generated.) If anti-replay is enabled (the default), the transmitted sequence number must never be allowed to cycle. Thus, the sender's counter and the receiver's counter MUST be reset (by establishing a new SA and thus a new key) prior to the transmission of the 2^{32} nd packet on an SA.

Extended (64-bit) Sequence Number

To support high-speed IPsec implementations, Extended Sequence Numbers (ESNs) SHOULD be implemented, as an extension to the current, 32-bit sequence number field. Use of an ESN MUST be negotiated by an SA management protocol. Note that in IKEv2, this negotiation is implicit; the default is ESN unless 32-bit sequence numbers are explicitly negotiated. (The ESN feature is applicable to multicast as well as unicast SAs.)

The ESN facility allows use of a 64-bit sequence number for an SA. (See Appendix A, "Extended (64-bit) Sequence Numbers", for details.) Only the low-order 32 bits of the sequence number are transmitted in the plaintext ESP header of each packet, thus minimizing packet overhead. The high-order 32 bits are maintained as part of the sequence number counter by both transmitter and receiver and are included in the computation of the ICV (if the integrity service is selected). If a separate integrity algorithm is employed, the high order bits are included in the implicit ESP trailer, but are not transmitted, analogous to integrity algorithm padding bits. If a combined mode algorithm is employed, the algorithm choice determines whether the high-order ESN bits are transmitted or are included implicitly in the computation. See Section 3.3.2.2 for processing details.

Payload Data

Payload Data is a variable-length field containing data (from the original IP packet) described by the Next Header field. The Payload Data field is mandatory and is an integral number of bytes in length. If the algorithm used to encrypt the payload requires cryptographic synchronization data, e.g., an Initialization Vector (IV), then this data is carried explicitly in the Payload field, but it is not called out as a separate field in ESP, i.e., the transmission of an explicit IV is invisible to ESP. (See Figure 2.) Any encryption algorithm that requires such explicit, per-packet synchronization data MUST indicate the length, any structure for such data, and the location of this data as part of an RFC specifying how the algorithm is used with ESP. (Typically, the IV immediately precedes the ciphertext. See Figure 2.) If such synchronization data is implicit, the algorithm for deriving the data MUST be part of the algorithm definition RFC. (If included in the Payload field, cryptographic synchronization data, e.g., an Initialization Vector (IV), usually is not encrypted per se (see Tables 1 and 2), although it sometimes is referred to as being part of the ciphertext.)

Note that the beginning of the next layer protocol header MUST be aligned relative to the beginning of the ESP header as follows. For IPv4, this alignment is a multiple of 4 bytes. For IPv6, the alignment is a multiple of 8 bytes.

With regard to ensuring the alignment of the (real) ciphertext in the presence of an IV, note the following:

- For some IV-based modes of operation, the receiver treats the IV as the start of the ciphertext, feeding it into the algorithm directly. In these modes, alignment of the start of the (real) ciphertext is not an issue at the receiver.
- In some cases, the receiver reads the IV in separately from the ciphertext. In these cases, the algorithm specification MUST address how alignment of the (real) ciphertext is to be achieved.

Padding (for Encryption)

Two primary factors require or motivate use of the Padding field.

- If an encryption algorithm is employed that requires the plaintext to be a multiple of some number of bytes, e.g., the block size of a block cipher, the Padding field is used to fill the plaintext (consisting of the Payload Data, Padding, Pad Length, and Next Header fields) to the size required by the algorithm.
- Padding also may be required, irrespective of encryption algorithm requirements, to ensure that the resulting ciphertext terminates on a 4-byte boundary. Specifically, the Pad Length and Next Header fields must be right aligned within a 4-byte word, as illustrated in the ESP packet format figures above, to ensure that the ICV field (if present) is aligned on a 4-byte boundary.

Padding beyond that required for the algorithm or alignment reasons cited above could be used to conceal the actual length of the payload, in support of TFC. However, the Padding field described is too limited to be effective for TFC and thus should not be used for that purpose. Instead, the separate mechanism described below (see Section 2.7) should be used when TFC is required.

The sender MAY add 0 to 255 bytes of padding. Inclusion of the Padding field in an ESP packet is optional, subject to the requirements noted above, but all implementations MUST support generation and consumption of padding.

- For the purpose of ensuring that the bits to be encrypted are a multiple of the algorithm's block size (first bullet above), the padding computation applies to the Payload Data exclusive of any IV, but including the ESP trailer fields. If a combined algorithm mode requires transmission of the SPI and Sequence Number to effect integrity, e.g., replication of the SPI and Sequence Number in the Payload Data, then the replicated versions of these data items, and any associated, ICV-equivalent data, are included in the computation of the pad length. (If the ESN option is selected, the high-order 32 bits of the ESN also would enter into the computation, if the combined mode algorithm requires their transmission for integrity.)
- For the purposes of ensuring that the ICV is aligned on a 4-byte boundary (second bullet above), the padding computation applies to the Payload Data inclusive of the IV, the Pad Length, and Next Header fields. If a combined mode algorithm is used, any replicated data and ICV-equivalent data are included in the Payload Data covered by the padding computation.

If Padding bytes are needed but the encryption algorithm does not specify the padding contents, then the following default processing MUST be used. The Padding bytes are initialized with a series of (unsigned, 1-byte) integer values. The first padding byte appended to the plaintext is numbered 1, with subsequent padding bytes making up a monotonically increasing sequence: 1, 2, 3, When this padding scheme is employed, the receiver SHOULD inspect the Padding field. (This scheme was selected because of its relative simplicity, ease of implementation in hardware, and because it offers limited protection against certain forms of "cut and paste" attacks in the absence of other integrity measures, if the receiver checks the padding values upon decryption.)

If an encryption or combined mode algorithm imposes constraints on the values of the bytes used for padding, they MUST be specified by the RFC defining how the algorithm is employed with ESP. If the algorithm requires checking of the values of the bytes used for padding, this too MUST be specified in that RFC.

Pad Length

The Pad Length field indicates the number of pad bytes immediately preceding it in the Padding field. The range of valid values is 0 to 255, where a value of zero indicates that no Padding bytes are present. As noted above, this does not include any TFC padding bytes. The Pad Length field is mandatory.

Next Header

The Next Header is a mandatory, 8-bit field that identifies the type of data contained in the Payload Data field, e.g., an IPv4 or IPv6 packet, or a next layer header and data. The value of this field is chosen from the set of IP Protocol Numbers defined on the web page of the IANA, e.g., a value of 4 indicates IPv4, a value of 41 indicates IPv6, and a value of 6 indicates TCP.

To facilitate the rapid generation and discarding of the padding traffic in support of traffic flow confidentiality (see Section 2.4), the protocol value 59 (which means "no next header") MUST be used to designate a "dummy" packet. A transmitter MUST be capable of generating dummy packets marked with this value in the next protocol field, and a receiver MUST be prepared to discard such packets, without indicating an error. All other ESP header and trailer fields (SPI, Sequence Number, Padding, Pad Length, Next Header, and ICV) MUST be present in dummy packets, but the plaintext portion of the payload, other than this Next Header field, need not be well-formed, e.g., the rest of the Payload Data may consist of only random bytes. Dummy packets are discarded without prejudice.

Implementations SHOULD provide local management controls to enable the use of this capability on a per-SA basis. The controls should allow the user to specify if this feature is to be used and also provide parametric controls; for example, the controls might allow an administrator to generate random-length or fixed-length dummy packets.

DISCUSSION: Dummy packets can be inserted at random intervals to mask the absence of actual traffic. One can also "shape" the actual traffic to match some distribution to which dummy traffic is added as dictated by the distribution parameters. As with the packet length padding facility for Traffic Flow Security (TFS), the most secure approach would be to generate dummy packets at whatever rate is needed to maintain a constant rate on an SA. If packets are all the same size, then the SA presents the appearance of a constant bit rate data stream, analogous

to what a link crypto would offer at layer 1 or 2. However, this is unlikely to be practical in many contexts, e.g., when there are multiple SAs active, because it would imply reducing the allowed bandwidth for a site, based on the number of SAs, and that would undermine the benefits of packet switching. Implementations SHOULD provide controls to enable local administrators to manage the generation of dummy packets for TFC purposes.

Traffic Flow Confidentiality (TFC) Padding

As noted above, the Padding field is limited to 255 bytes in length. This generally will not be adequate to hide traffic characteristics relative to traffic flow confidentiality requirements. An optional field, within the payload data, is provided specifically to address the TFC requirement.

An IPsec implementation SHOULD be capable of padding traffic by adding bytes after the end of the Payload Data, prior to the beginning of the Padding field. However, this padding (hereafter referred to as TFC padding) can be added only if the Payload Data field contains a specification of the length of the IP datagram. This is always true in tunnel mode, and may be true in transport mode depending on whether the next layer protocol (e.g., IP, UDP, ICMP) contains explicit length information. This length information will enable the receiver to discard the TFC padding, because the true length of the Payload Data will be known. (ESP trailer fields are located by counting back from the end of the ESP packet.) Accordingly, if TFC padding is added, the field containing the specification of the length of the IP datagram MUST NOT be modified to reflect this padding. No requirements for the value of this padding are established by this standard.

In principle, existing IPsec implementations could have made use of this capability previously, in a transparent fashion. However, because receivers may not have been prepared to deal with this padding, the SA management protocol MUST negotiate this service prior to a transmitter employing it, to ensure backward compatibility. Combined with the convention described in Section 2.6 above, about the use of protocol ID 59, an ESP implementation is capable of generating dummy and real packets that exhibit much greater length variability, in support of TFC.

Implementations SHOULD provide local management controls to enable the use of this capability on a per-SA basis. The controls should allow the user to specify if this feature is to be used and also provide parametric controls for the feature.

Integrity Check Value (ICV)

The Integrity Check Value is a variable-length field computed over the ESP header, Payload, and ESP trailer fields. Implicit ESP trailer fields (integrity padding and high-order ESN bits, if applicable) are included in the ICV computation. The ICV field is optional. It is present only if the integrity service is selected and is provided by either a separate integrity algorithm or a combined mode algorithm that uses an ICV. The length of the field is specified by the integrity algorithm selected and associated with the SA. The integrity algorithm specification MUST specify the length of the ICV and the comparison rules and processing steps for validation.

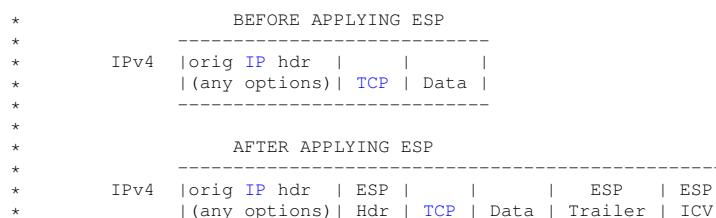
7.61.2.1 Encapsulating Security Protocol Processing

ESP Header Location

ESP may be employed in two ways: transport mode or tunnel mode.

Transport Mode Processing

In transport mode, ESP is inserted after the IP header and before a next layer protocol, e.g., TCP, UDP, ICMP, etc. In the context of IPv4, this translates to placing ESP after the IP header (and any options that it contains), but before the next layer protocol. (If AH is also applied to a packet, it is applied to the ESP header, Payload, ESP trailer, and ICV, if present.) (Note that the term "transport" mode should not be misconstrued as restricting its use to TCP and UDP.) The following diagram illustrates ESP transport mode positioning for a typical IPv4 packet, on a "before and after" basis. (This and subsequent diagrams in this section show the ICV field, the presence of which is a function of the security services and the algorithm/mode selected.)



```

*
-----|<---- encryption ---->|
*      |<----- integrity ----->|
*

```

In the IPv6 context, ESP is viewed as an end-to-end payload, and thus should appear after hop-by-hop, routing, and fragmentation extension headers. Destination options extension header(s) could appear before, after, or both before and after the ESP header depending on the semantics desired. However, because ESP protects only fields after the ESP header, it generally will be desirable to place the destination options header(s) after the ESP header. The following diagram illustrates ESP transport mode positioning for a typical IPv6 packet.

```

*          BEFORE APPLYING ESP
*          -----
*          IPv6 |           | ext hdrs |           |           |
*          | orig IP hdr |if present| TCP | Data |
*          -----
*
*          AFTER APPLYING ESP
*          -----
*          IPv6 | orig |hop-by-hop,dest*,|   |dest|   |   |ESP|   |ESP|
*          |IP|hdr|routing,fragment.|ESP|opt*|TCP|Data|Trailer|ICV|
*          -----
*          |<---- encryption ---->|
*          |<----- integrity ----->|
*
*          * = if present, could be before ESP, after ESP, or both
*
```

Note that in transport mode, for "bump-in-the-stack" or "bump-in- the-wire" implementations, as defined in the Security Architecture document, inbound and outbound IP fragments may require an IPsec implementation to perform extra IP reassembly/fragmentation in order to both conform to this specification and provide transparent IPsec support. Special care is required to perform such operations within these implementations when multiple interfaces are in use.

Tunnel Mode Processing

In tunnel mode, the "inner" IP header carries the ultimate (IP) source and destination addresses, while an "outer" IP header contains the addresses of the IPsec "peers", e.g., addresses of security gateways. Mixed inner and outer IP versions are allowed, i.e., IPv6 over IPv4 and IPv4 over IPv6. In tunnel mode, ESP protects the entire inner IP packet, including the entire inner IP header. The position of ESP in tunnel mode, relative to the outer IP header, is the same as for ESP in transport mode. The following diagram illustrates ESP tunnel mode positioning for typical IPv4 and IPv6 packets.

```

*          BEFORE APPLYING ESP
*          -----
*          IPv4 |orig IP hdr|   |   |   |
*          |(any options)| TCP | Data |
*          -----
*
*          AFTER APPLYING ESP
*          -----
*          IPv4 | new IP hdr*|   | orig IP hdr*|   |   |ESP|   |ESP|
*          |(any options)| ESP | (any options)|TCP|Data|Trailer|ICV|
*          -----
*          |<----- encryption ----->|
*          |<----- integrity ----->|
*
*          BEFORE APPLYING ESP
*          -----
*          IPv6 |           | ext hdrs |           |           |
*          | orig IP hdr |if present| TCP | Data |
*          -----
*
*          AFTER APPLYING ESP
*          -----
*          IPv6 | new* |new ext|   | orig*|orig ext|   |   |ESP|   |ESP|
*          |IP|hdr|hdrs*|ESP|IP|hdr|hdrs*|TCP|Data|Trailer|ICV|
*          -----
*          |<----- encryption ----->|
*          |<----- integrity ----->|
*
*          * = if present, construction of outer IP hdr/extensions and
*          modification of inner IP hdr/extensions is discussed in
*          the Security Architecture document.
*
```

7.61.2.2 Network Address Translation Transversal (NAT)

See https://en.wikipedia.org/wiki/Network_address_translation

See <https://www.rfc-editor.org/rfc/pdfrfc/rfc3948.txt.pdf>

Network address translation (NAT) is a method of remapping one IP address space into another by modifying network address information in Internet Protocol (IP) datagram packet headers while they are in transit across a traffic routing device.[1] The technique was originally used for ease of rerouting traffic in IP networks without renumbering every host. It has become a popular and essential tool in conserving global address space allocations in face of IPv4 address exhaustion by sharing one Internet-routable IP address of a NAT gateway for an entire private network

Methodology

The original use of network address translation consisted of mapping every address of one address space to a corresponding address in another space, such as when an enterprise changed Internet service providers without having a facility to announce a public route to the network. In the face of the foreseeable global IP address space exhaustion, NAT was increasingly used since the late 1990s in conjunction with IP masquerading, which is a technique that hides an entire IP address space, usually consisting of private network IP addresses (RFC 1918), behind a single IP address in another, usually public address space. This mechanism is implemented in a routing device that uses stateful translation tables to map the "hidden" addresses into a single IP address and readdresses the outgoing IP packets on exit so they appear to originate from the routing device. In the reverse communications path, responses are mapped back to the originating IP addresses using the rules ("state") stored in the translation tables. The translation table rules established in this fashion are flushed after a short period unless new traffic refreshes their state to prevent port exhaustion and free state table resources

The method enables communication through the router only when the conversation originates in the masqueraded network since this establishes the translation tables. For example, a web browser in the masqueraded network can browse a website outside, but a web browser outside could not browse a website hosted within the masqueraded network. However, most NAT devices today allow the network administrator to configure translation table entries for permanent use. This feature is often referred to as "static NAT" or port forwarding and allows traffic originating in the "outside" network to reach designated hosts in the masqueraded network

Because of the popularity of this technique to conserve IPv4 address space, the term NAT has become virtually synonymous with the method of IP masquerading

As network address translation modifies the IP address information in packets, it has serious consequences on the quality of Internet connectivity and requires careful attention to the details of its implementation. NAT implementations vary widely in their specific behavior in various addressing cases and their effect on network traffic. The specifics of NAT behavior is not commonly documented by vendors of equipment containing implementations.[2]

Basic NAT

The simplest type of NAT provides a one-to-one translation of IP addresses. RFC 2663 refers to this type of NAT as basic NAT; it is often also called a one-to-one NAT. In this type of NAT, only the IP addresses, IP header checksum and any higher level checksums that include the IP address are changed. Basic NATs can be used to interconnect two IP networks that have incompatible addressing

One-to-many NAT

The majority of NATs map multiple private hosts to one publicly exposed IP address. In a typical configuration, a local network uses one of the designated "private" IP address subnets (RFC 1918). A router on that network has a private address in that address space. The router is also connected to the Internet with a "public" address assigned by an Internet service provider. As traffic passes from the local network to the Internet, the source address in each packet is translated on the fly from a private address to the public address. The router tracks basic data about each active connection (particularly the destination address and port). When a reply returns to the router, it uses the connection tracking data it stored during the outbound phase to determine the private address on the internal network to which to forward the reply

One of the benefits of this is that it is a practical solution to the impending exhaustion of the IPv4 address space. Even large networks can be connected to the Internet with a single IP address. The more common arrangement is having computers that require end-to-end connectivity supplied with a routable IP address, while having others that do not provide services to outside users behind NAT with only a few IP addresses used to enable Internet access

All datagram packets on IP networks have a source IP address and a destination IP address. Typically packets passing from the private network to the public network will have their source address modified while packets passing from the public network back to the private network will have their destination address modified. More complex configurations are also possible

To avoid ambiguity in how to translate returned packets, further modifications to the packets are required. The vast bulk of Internet traffic is TCP and UDP packets, and for these protocols the port numbers are changed so that the combination of IP address and port information on the returned packet can be unambiguously mapped to the corresponding private address and port information. RFC 2663 uses the term network address and port translation (NAPT) for this type of NAT. Other names include port address translation (PAT), IP masquerading, NAT overload and many-to-one NAT. This is the most common type of NAT and has become synonymous with the term NAT in common usage. This method enables communication through the router only when the conversation originates in the masqueraded network since this establishes the translation tables. For example, a web browser in the masqueraded network can browse a website outside, but a web browser outside could not browse a website hosted within the masqueraded network. However, most NAT devices today allow the network administrator to configure static translation table entries for connections from the external network to the internal masqueraded network. This feature is often referred to as static NAT. It may be implemented in two types: port forwarding which forwards traffic from a specific external port to an internal host on a specified port, and designation of a DMZ host which passes all traffic received on the external interface (on any port number) to an internal IP address while preserving the destination port. Both types may be available in the same NAT device

Protocols not based on TCP and UDP require other translation techniques

<p>Full-cone NAT, also known as <i>one-to-one NAT</i></p> <ul style="list-style-type: none"> Once an internal address (iAddr:iPort) is mapped to an external address (eAddr:ePort), any packets from iAddr:iPort are sent through eAddr:ePort. Any external host can send packets to iAddr:iPort by sending packets to eAddr:ePort. 	<p>"Full Cone" NAT</p>
<p>(Address)-restricted-cone NAT</p> <ul style="list-style-type: none"> Once an internal address (iAddr:iPort) is mapped to an external address (eAddr:ePort), any packets from iAddr:iPort are sent through eAddr:ePort. An external host (<i>hAddr:any</i>) can send packets to iAddr:iPort by sending packets to eAddr:ePort only if iAddr:iPort has previously sent a packet to hAddr:any. "Any" means the port number doesn't matter. 	<p>"Restricted Cone" NAT</p>
<p>Port-restricted cone NAT</p> <p>Like an address restricted cone NAT, but the restriction includes port numbers.</p> <ul style="list-style-type: none"> Once an internal address (iAddr:iPort) is mapped to an external address (eAddr:ePort), any packets from iAddr:iPort are sent through eAddr:ePort. An external host (<i>hAddr:hPort</i>) can send packets to iAddr:iPort by sending packets to eAddr:ePort only if iAddr:iPort has previously sent a packet to hAddr:hPort. 	<p>"Port Restricted Cone" NAT</p>
<p>Symmetric NAT</p> <ul style="list-style-type: none"> Each request from the same internal IP address and port to a specific destination IP address and port is mapped to a unique external source IP address and port; if the same internal host sends a packet even with the same source address and port but to a different destination, a different mapping is used. Only an external host that receives a packet from an internal host can send a packet back. 	<p>"Symmetric" NAT</p>

Figure 7.74: Network Address Translation (NAT) Types

NAT and TCP/UDP

"Pure NAT", operating on IP alone, may or may not correctly parse protocols that are totally concerned with IP information, such as ICMP, depending on whether the payload is interpreted by a host on the "inside" or "outside" of translation. As soon as the protocol stack is traversed, even with such basic protocols as TCP and UDP, the

protocols will break unless NAT takes action beyond the network layer

IP packets have a checksum in each packet header, which provides error detection only for the header. IP datagrams may become fragmented and it is necessary for a NAT to reassemble these fragments to allow correct recalculation of higher-level checksums and correct tracking of which packets belong to which connection

The major transport layer protocols, TCP and UDP, have a checksum that covers all the data they carry, as well as the TCP/UDP header, plus a "pseudo-header" that contains the source and destination IP addresses of the packet carrying the TCP/UDP header. For an originating NAT to pass TCP or UDP successfully, it must recompute the TCP/UDP header checksum based on the translated IP addresses, not the original ones, and put that checksum into the TCP/UDP header of the first packet of the fragmented set of packets. The receiving NAT must recompute the IP checksum on every packet it passes to the destination host, and also recognize and recompute the TCP/UDP header using the retranslated addresses and pseudo-header. This is not a completely solved problem. One solution is for the receiving NAT to reassemble the entire segment and then recompute a checksum calculated across all packets

The originating host may perform Maximum transmission unit (MTU) path discovery to determine the packet size that can be transmitted without fragmentation, and then set the don't fragment (DF) bit in the appropriate packet header field. Of course, this is only a one-way solution, because the responding host can send packets of any size, which may be fragmented before reaching the NAT

7.61.2.3 Using Advanced Encryption Standard (AES) Counter Mode With IPsec Encapsulating Security Payload (ESP)

See <https://tools.ietf.org/pdf/rfc3686.pdf>

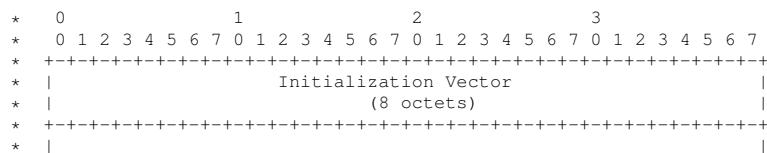
AES-CTR has many properties that make it an attractive encryption algorithm for in high-speed networking. AES-CTR uses the AES block cipher to create a stream cipher. Data is encrypted and decrypted by XORing with the key stream produced by AES encrypting sequential counter block values. AES-CTR is easy to implement, and AES-CTR can be pipelined and parallelized. AES-CTR also supports key stream precomputation

Pipelining is possible because AES has multiple rounds (see section 2.2). A hardware implementation (and some software implementations) can create a pipeline by unwinding the loop implied by this round structure. For example, after a 16-octet block has been input, one round later another 16-octet block can be input, and so on. In AES-CTR, these inputs are the sequential counter block values used to generate the key stream. Multiple independent AES encrypt implementations can also be used to improve performance. For example, one could use two AES encrypt implementations in parallel, to process a sequence of counter block values, doubling the effective throughput. The sender can precompute the key stream. Since the key stream does not depend on any data in the packet, the key stream can be precomputed once the nonce and IV are assigned. This precomputation can reduce packet latency. The receiver cannot perform similar precomputation because the IV will not be known before the packet arrives

AES-CTR uses the only AES encrypt operation (for both encryption and decryption), making AES-CTR implementations smaller than implementations of many other AES modes. When used correctly, AES-CTR provides a high level of confidentiality. Unfortunately, AES-CTR is easy to use incorrectly. Being a stream cipher, any reuse of the per-packet value, called the IV, with the same nonce and key is catastrophic. An IV collision immediately leaks information about the plaintext in both packets. For this reason, it is inappropriate to use this mode of operation with static keys. Extraordinary measures would be needed to prevent reuse of an IV value with the static key across power cycles. To be safe, implementations MUST use fresh keys with AES-CTR. The Internet Key Exchange (IKE) protocol can be used to establish fresh keys. IKE can also provide the nonce value. With AES-CTR, it is trivial to use a valid ciphertext to forge other (valid to the decryptor) ciphertexts. Thus, it is equally catastrophic to use AES-CTR without a companion authentication function. Implementations MUST use AES-CTR in conjunction with an authentication function, such as HMAC-SHA-1-96

ESP Payload

The ESP payload is comprised of the IV followed by the ciphertext. The payload field, as defined in [ESP], is structured as shown in Figure 1



```

* ~ Encrypted Payload (variable) ~
* | |
* +-----+-----+-----+-----+-----+-----+-----+
* | |
* ~ Authentication Data (variable) ~
* | |
* +-----+-----+-----+-----+-----+-----+-----+
* Figure 1. ESP Payload Encrypted with AES-CTR
*

```

Initialization Vector

The AES-CTR IV field MUST be eight octets. The IV MUST be chosen by the encryptor in a manner that ensures that the same IV value is used only once for a given key. The encryptor can generate the IV in any manner that ensures uniqueness. Common approaches to IV generation include incrementing a counter for each packet and linear feedback shift registers (LFSRs). Including the IV in each packet ensures that the decryptor can generate the key stream needed for decryption, even when some packets are lost or reordered

Encrypted Payload

The encrypted payload contains the ciphertext. AES-CTR mode does not require plaintext padding. However, ESP does require padding to 32-bit word-align the authentication data. The padding, Pad Length, and the Next Header MUST be concatenated with the plaintext before performing encryption, as described in [ESP]

Authentication Data

Since it is trivial to construct a forgery AES-CTR ciphertext from a valid AES-CTR ciphertext, AES-CTR implementations MUST employ a non-NUL ESP authentication method. HMAC-SHA-1-96 [HMAC-SHA] is a likely choice

Counter Block Format

Each packet conveys the IV that is necessary to construct the sequence of counter blocks used to generate the key stream necessary to decrypt the payload. The AES counter block cipher block is 128 bits. Figure 2 shows the format of the counter block

```

* 0 1 2 3 4 5 6 7 0 1 2 3 4 5 6 7 0 1 2 3 4 5 6 7
* +-----+-----+-----+-----+-----+-----+-----+
* | | Nonce | |
* +-----+-----+-----+-----+-----+-----+-----+
* | | Initialization Vector (IV) | |
* | | | |
* +-----+-----+-----+-----+-----+-----+-----+
* | | Block Counter | |
* +-----+-----+-----+-----+-----+-----+-----+
* Figure 2. Counter Block Format
*

```

The components of the counter block are as follows:

- Nonce

The Nonce field is 32 bits. As the name implies, the nonce is a single use value. That is, a fresh nonce value MUST be assigned for each security association. It MUST be assigned at the beginning of the security association. The nonce value need not be secret, but it MUST be unpredictable prior to the beginning of the security association

- Initialization Vector

The IV field is 64 bits. As described in section 3.1, the IV MUST be chosen by the encryptor in a manner that ensures that the same IV value is used only once for a given key.

- Block Counter

The block counter field is the least significant 32 bits of the counter block. The block counter begins with the value of one, and it is incremented to generate subsequent portions of the key stream. The block counter is a 32-bit big-endian integer value. Using the encryption process described in section 2.1, this construction permits each packet to consist of up to:

- $2^{32} - 1 \text{ blocks} = 4,294,967,295 \text{ blocks} = 68,719,476,720 \text{ octets}$

This construction can produce enough key stream for each packet sufficient to handle any IPv6 jumbogram

Keying Material and Nonces

As described in section 2.1, implementations MUST use fresh keys with AES-CTR. IKE can be used to establish fresh keys. This section describes the conventions for obtaining the unpredictable nonce value from IKE. Note that this convention provides a nonce value that is secret as well as unpredictable.

IKE makes use of a pseudo-random function (PRF) to derive keying material. The PRF is used iteratively to derive keying material of arbitrary size, called KEYMAT. Keying material is extracted from the output string without regard to boundaries. The size of the requested KEYMAT MUST be four octets longer than is needed for the associated AES key. The keying material is used as follows:

- AES-CTR with a 128 bit key

The KEYMAT requested for each AES-CTR key is 20 octets. The first 16 octets are the 128-bit AES key, and the remaining four octets are used as the nonce value in the counter block.

- AES-CTR with a 192 bit key

The KEYMAT requested for each AES-CTR key is 28 octets. The first 24 octets are the 192-bit AES key, and the remaining four octets are used as the nonce value in the counter block.

- AES-CTR with a 256 bit key

The KEYMAT requested for each AES-CTR key is 36 octets. The first 32 octets are the 256-bit AES key, and the remaining four octets are used as the nonce value in the counter block

7.61.2.4 The Use of Galois/Counter Mode (GCM) in IPsec Encapsulating Security Payload (ESP)

See <https://tools.ietf.org/pdf/rfc4106.pdf>

GCM is a block cipher mode of operation providing both confidentiality and data origin authentication. The GCM authenticated encryption operation has four inputs: a secret key, an initialization vector (IV), a plaintext, and an input for additional authenticated data (AAD). It has two outputs, a ciphertext whose length is identical to the plaintext, and an authentication tag. In the following, we describe how the IV, plaintext, and AAD are formed from the ESP fields, and how the ESP packet is formed from the ciphertext and authentication tag.

ESP also defines an IV. For clarity, we refer to the AES-GCM IV as a nonce in the context of AES-GCM-ESP. The same nonce and key combination MUST NOT be used more than once.

Because reusing an nonce/key combination destroys the security guarantees of AES-GCM mode, it can be difficult to use this mode securely when using statically configured keys. For safety's sake, implementations MUST use an automated key management system, such as the Internet Key Exchange (IKE) [RFC2409], to ensure that this requirement is met.

ESP Payload Data

The ESP Payload Data is comprised of an eight-octet initialization vector (IV), followed by the ciphertext. The payload field, as defined in [RFC2406], is structured as shown in Figure 1, along with the ICV associated with the payload.

```
*   0           1           2           3
*   0 1 2 3 4 5 6 7 0 1 2 3 4 5 6 7 0 1 2 3 4 5 6 7
*   +-----+-----+-----+
*   |           Initialization Vector           |
*   |           (8 octets)                      |
*   +-----+-----+-----+-----+-----+-----+-----+
*   ~           Ciphertext (variable)          ~
*   |                           |
*   +-----+-----+-----+-----+-----+-----+-----+
*
*   Figure 1: ESP Payload Encrypted with AES-GCM.
```

Initialization Vector (IV)

The AES-GCM-ESP IV field MUST be eight octets. For a given key, the IV MUST NOT repeat. The most natural way to implement this is with a counter, but anything that guarantees uniqueness can be used, such as a linear feedback shift register (LFSR). Note that the encrypter can use any IV generation method that meets the uniqueness requirement, without coordinating with the decrypter.

Ciphertext

The plaintext input to AES-GCM is formed by concatenating the plaintext data described by the Next Header field with the Padding, the Pad Length, and the Next Header field. The Ciphertext field consists of the ciphertext output from the AES-GCM algorithm. The length of the ciphertext is identical to that of the plaintext.

Implementations that do not seek to hide the length of the plaintext SHOULD use the minimum amount of padding required, which will be less than four octets.

Nonce Format

The nonce passed to the GCM-AES encryption algorithm has the following layout:

```
*   0           1           2           3
*   0 1 2 3 4 5 6 7 0 1 2 3 4 5 6 7 0 1 2 3 4 5 6 7
*   +-----+-----+-----+-----+
*   |           Salt          |
*   +-----+-----+-----+-----+-----+-----+-----+-----+
*   |           Initialization Vector    |
*   |           |
*   +-----+-----+-----+-----+-----+-----+-----+-----+
*
*   Figure 2: Nonce Format
*
```

The components of the nonce are as follows:

- Salt

The salt field is a four-octet value that is assigned at the beginning of the security association, and then remains constant for the life of the security association. The salt SHOULD be unpredictable (i.e., chosen at random) before it is selected, but need not be secret. We describe how to set the salt for a Security Association established via the Internet Key Exchange in Section 8.1.

- Initialization Vector

The IV field is described in Section 3.1.

AAD Construction

The authentication of data integrity and data origin for the SPI and (Extended) Sequence Number fields is provided without encryption. This is done by including those fields in the AES-GCM Additional Authenticated Data (AAD) field. Two formats of the AAD are defined: one for 32-bit sequence numbers, and one for 64-bit extended sequence numbers. The format with 32-bit sequence numbers is shown in Figure 3, and the format with 64-bit extended sequence numbers is shown in Figure 4.

```
*   0           1           2           3
*   0 1 2 3 4 5 6 7 0 1 2 3 4 5 6 7 0 1 2 3 4 5 6 7
*   +-----+-----+-----+-----+
*   |           SPI          |
*   +-----+-----+-----+-----+-----+-----+-----+-----+
*   |           32-bit Sequence Number    |
*   +-----+-----+-----+-----+-----+-----+-----+-----+
*
*   Figure 3: AAD Format with 32-bit Sequence Number
*
```

```
*   0           1           2           3
*   0 1 2 3 4 5 6 7 0 1 2 3 4 5 6 7 0 1 2 3 4 5 6 7
*   +-----+-----+-----+-----+
*   |           SPI          |
*   +-----+-----+-----+-----+-----+-----+-----+-----+
*   |           64-bit Extended Sequence Number    |
*   +-----+-----+-----+-----+-----+-----+-----+-----+
*
*   Figure 4: AAD Format with 64-bit Extended Sequence Number
*
```

Integrity Check Value (ICV)

The ICV consists solely of the AES-GCM Authentication Tag. Implementations MUST support a full-length 16-octet ICV, and MAY support 8 or 12 octet ICVs, and MUST NOT support other ICV lengths. Although ESP does not require that an ICV be present, AES-GCM-ESP intentionally does not allow a zero-length ICV. This is because GCM provides no integrity protection whatsoever when used with a zero length Authentication Tag.

Packet Expansion

The IV adds an additional eight octets to the packet, and the ICV adds an additional 8, 12, or 16 octets. These are the only sources of packet expansion, other than the 10-13 octets taken up by the ESP SPI, Sequence Number, Padding, Pad Length, and Next Header fields (if the minimal amount of padding is used).

IKE Conventions

This section describes the conventions used to generate keying material and salt values, for use with AES-GCM-ESP, using the Internet Key Exchange (IKE) [RFC2409] protocol. The identifiers and attributes needed to negotiate a security association using AES-GCM ESP are also defined.

Keying Material and Salt Values

IKE makes use of a pseudo-random function (PRF) to derive keying material. The PRF is used iteratively to derive keying material of arbitrary size, called KEYMAT. Keying material is extracted from the output string without regard to boundaries.

The size of the KEYMAT for the AES-GCM-ESP MUST be four octets longer than is needed for the associated AES key. The keying material is used as follows:

- AES-GCM-ESP with a 128 bit key

The KEYMAT requested for each AES-GCM key is 20 octets. The first 16 octets are the 128-bit AES key, and the remaining four octets are used as the salt value in the nonce.

- AES-GCM-ESP with a 192 bit key

The KEYMAT requested for each AES-GCM key is 28 octets. The first 24 octets are the 192-bit AES key, and the remaining four octets are used as the salt value in the nonce.

- AES-GCM-ESP with a 256 bit key

The KEYMAT requested for each AES GCM key is 36 octets. The first 32 octets are the 256-bit AES key, and the remaining four octets are used as the salt value in the nonce.

7.61.2.5 Using Advanced Encryption Standard (AES) CCM Mode with IPsec Encapsulating Security Payload (ESP)

See <https://www.rfc-editor.org/rfc/pdfrfc/rfc4309.txt.pdf>

AES CCM Mode

CCM is a generic authenticate-and-encrypt block cipher mode [CCM]. In this specification, CCM is used with the AES [AES] block cipher. AES CCM has two parameters:

- M

M indicates the size of the integrity check value (ICV). CCM defines values of 4, 6, 8, 10, 12, 14, and 16 octets; However, to maintain alignment and provide adequate security, only the values that are a multiple of four and are at least eight are permitted. Implementations MUST support M values of 8 octets and 16 octets, and implementations MAY support an M value of 12 octets.

- L

L indicates the size of the length field in octets. CCM defines values of L between 2 octets and 8 octets. This specification only supports L = 4. Implementations MUST support an L value of 4 octets, which accommodates a full Jumbogram [JUMBO]; however, the length includes all of the encrypted data, which also includes the ESP Padding, Pad Length, and Next Header fields.

There are four inputs to CCM originator processing:

- key

A single key is used to calculate the ICV using CBC-MAC and to perform payload encryption using counter mode. AES supports key sizes of 128 bits, 192 bits, and 256 bits. The default key size is 128 bits, and implementations MUST support this key size. Implementations MAY also support key sizes of 192 bits and 256 bits.

- nonce

The size of the nonce depends on the value selected for the parameter L. It is 15-L octets. Implementations MUST support a nonce of 11 octets. The construction of the nonce is described in Section 4.

- payload

The payload of the ESP packet. The payload MUST NOT be longer than 4,294,967,295 octets, which is the maximum size of a Jumbogram [JUMBO]; however, the ESP Padding, Pad Length, and Next Header fields are also part of the payload.

- AAD

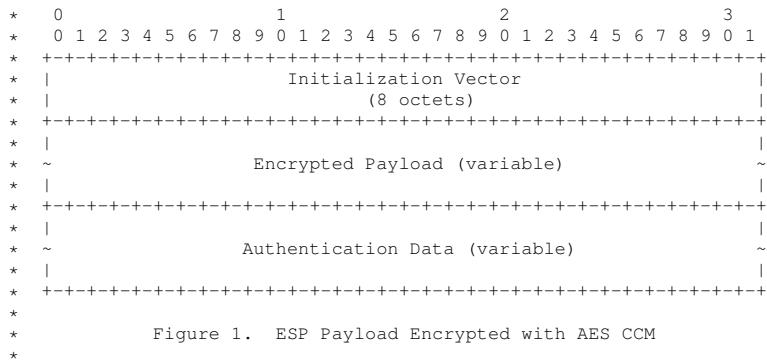
CCM provides data integrity and data origin authentication for some data outside the payload. CCM does not allow additional authenticated data (AAD) to be longer than 18,446,744,073,709,551,615 octets. The ICV is computed from the ESP header, Payload, and ESP trailer fields, which is significantly smaller than the CCM-imposed limit. The construction of the AAD described in Section 5.

AES CCM requires the encryptor to generate a unique per-packet value and to communicate this value to the decryptor. This per-packet value is one of the component parts of the nonce, and it is referred to as the initialization vector (IV). The same IV and key combination MUST NOT be used more than once. The encryptor can generate the IV in any manner that ensures uniqueness. Common approaches to IV generation include incrementing a counter for each packet and linear feedback shift registers (LFSRs).

AES CCM employs counter mode for encryption. As with any stream cipher, reuse of the same IV value with the same key is catastrophic. An IV collision immediately leaks information about the plaintext in both packets. For this reason, it is inappropriate to use this CCM with statically configured keys. Extraordinary measures would be needed to prevent reuse of an IV value with the static key across power cycles. To be safe, implementations MUST use fresh keys with AES CCM. The Internet Key Exchange (IKE) [IKE] protocol or IKEv2 [IKEv2] can be used to establish fresh keys.

ESP Payload

The ESP payload is composed of the IV followed by the ciphertext. The payload field, as defined in [ESP], is structured as shown in Figure 1.



Initialization Vector (IV)

The AES CCM IV field MUST be eight octets. The IV MUST be chosen by the encryptor in a manner that ensures that the same IV value is used only once for a given key. The encryptor can generate the IV in any manner that ensures uniqueness. Common/ approaches to IV generation include incrementing a counter for each packet and linear feedback shift registers (LFSRs).

Including the IV in each packet ensures that the decryptor can generate the key stream needed for decryption, even when some datagrams are lost or reordered.

Encrypted Payload

The encrypted payload contains the ciphertext.

AES CCM mode does not require plaintext padding. However, ESP does require padding to 32-bit word-align the authentication data. The Padding, Pad Length, and Next Header fields MUST be concatenated with the plaintext before performing encryption, as described in [ESP]. When padding is required, it MUST be generated and checked in accordance with the conventions specified in [ESP].

Authentication Data

AES CCM provides an encrypted ICV. The ICV provided by CCM is carried in the Authentication Data fields without further encryption. Implementations MUST support ICV sizes of 8 octets and 16 octets. Implementations MAY also support ICV 12 octets.

Nonce Format

Each packet conveys the IV that is necessary to construct the sequence of counter blocks used by counter mode to generate the key stream. The AES counter block is 16 octets. One octet is used for the CCM Flags, and 4 octets are used for the block counter, as specified by the CCM L parameter. The remaining octets are the nonce. These octets occupy the second through the twelfth octets in the counter block. Figure 2 shows the format of the nonce.

```

*   0           1           2           3
*   0 1 2 3 4 5 6 7 0 1 2 3 4 5 6 7 0 1 2 3 4 5 6 7
*           +-----+-----+-----+
*           |                   Salt
*           Initialization Vector
*           |
*           +-----+-----+-----+-----+-----+-----+-----+
*           |
*           Figure 2. Nonce Format
*

```

Figure 2. Nonce Format

The components of the nonce are as follows:

- Salt

The salt field is 24 bits. As the name implies, it contains an unpredictable value. It MUST be assigned at the beginning of the security association. The salt value need not be secret, but it MUST NOT be predictable prior to the beginning of the security association.

- Initialization Vector

The IV field is 64 bits. As described in Section 3.1, the IV MUST be chosen by the encryptor in a manner that ensures that the same IV value is used only once for a given key.

This construction permits each packet to consist of up to:

2^{32} blocks = 4 294 967 296 blocks = 68 719 476 736 octets

This construction provides more key stream for each packet than is needed to handle any IPv6 Jumbogram [JUMBO].

AAD Construction

The data integrity and data origin authentication for the Security Parameters Index (SPI) and (Extended) Sequence Number fields is provided without encrypting them. Two formats are defined: one for 32-bit sequence numbers and one for 64-bit extended sequence numbers. The format with 32-bit sequence numbers is shown in Figure 3, and the format with 64-bit extended sequence numbers is shown in Figure 4.

Sequence Numbers are conveyed canonical network byte order. Extended Sequence Numbers are conveyed canonical network byte order, placing the high-order 32 bits first and the low-order 32 bits second. Canonical network byte order is fully described in RFC 791, Appendix B.

```

*   0           1           2           3
*   0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
*   +-----+-----+-----+-----+-----+-----+-----+
*   |          SPI          |

```

```

* | 32-bit Sequence Number |
* +-----+-----+-----+-----+-----+-----+-----+
*
*      Figure 3. AAD Format with 32-bit Sequence Number
*
*
* 0          1          2          3
* 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
* +-----+-----+-----+-----+-----+-----+-----+
* |           SPI           |
* +-----+-----+-----+-----+-----+-----+-----+
* |           64-bit Extended Sequence Number           |
* |           |           |
* +-----+-----+-----+-----+-----+-----+-----+
*
*      Figure 4. AAD Format with 64-bit Extended Sequence Number
*

```

Packet Expansion

The initialization vector (IV) and the integrity check value (ICV) are the only sources of packet expansion. The IV always adds 8 octets to the front of the payload. The ICV is added at the end of the payload, and the CCM parameter M determines the size of the ICV. Implementations MUST support M values of 8 octets and 16 octets, and implementations MAY also support an M value of 12 octets.

IKE Conventions

This section describes the conventions used to generate keying material and salt values for use with AES CCM using the Internet Key Exchange (IKE) [IKE] protocol. The identifiers and attributes needed to negotiate a security association that uses AES CCM are also defined.

Keying Material and Salt Values

As previously described, implementations MUST use fresh keys with AES CCM. IKE can be used to establish fresh keys. This section describes the conventions for obtaining the unpredictable salt value for use in the nonce from IKE. Note that this convention provides a salt value that is secret as well as unpredictable.

IKE makes use of a pseudo-random function (PRF) to derive keying material. The PRF is used iteratively to derive keying material of arbitrary size, called KEYMAT. Keying material is extracted from the output string without regard to boundaries.

The size of KEYMAT MUST be three octets longer than is needed for the associated AES key. The keying material is used as follows:

- AES CCM with a 128-bit key

The KEYMAT requested for each AES CCM key is 19 octets. The first 16 octets are the 128-bit AES key, and the remaining three octets are used as the salt value in the counter block.

- AES CCM with a 192-bit key

The KEYMAT requested for each AES CCM key is 27 octets. The first 24 octets are the 192-bit AES key, and the remaining three octets are used as the salt value in the counter block.

- AES CCM with a 256-bit key

The KEYMAT requested for each AES CCM key is 35 octets. The first 32 octets are the 256-bit AES key, and the remaining three octets are used as the salt value in the counter block.

7.61.2.6 ChaCha20, Poly1305, and Their Use in the Internet Key Exchange Protocol (IKE) and IPsec

The Advanced Encryption Standard (AES) [FIPS-197] has become the go to algorithm for encryption. It is now the most commonly used algorithm in many areas, including IPsec Virtual Private Networks (VPNs). On most modern platforms, AES is anywhere from four to ten times as fast as the previously popular cipher, Triple Data Encryption Standard (3DES) [SP800-67]. 3DES also uses a 64-bit block; this means that the amount of data that can be encrypted before rekeying is required is limited. These reasons make AES not only the best choice, but the only viable choice for IPsec.

The problem is that if future advances in cryptanalysis reveal a weakness in AES, VPN users will be in an unenviable position. With the only other widely supported cipher for IPsec implementations being the much slower 3DES, it is

not feasible to reconfigure IPsec installations away from AES. [Standby-Cipher] describes this issue and the need for a standby cipher in greater detail.

This document proposes the fast and secure ChaCha20 stream cipher as such a standby cipher in an Authenticated Encryption with Associated Data (AEAD) construction with the Poly1305 authenticator for use with the Encapsulated Security Protocol (ESP) [RFC4303] and the Internet Key Exchange Protocol version 2 (IKEv2) [RFC7296]. The algorithms are described in a separate document ([RFC7539]). This document only describes the IPsec-specific things.

ChaCha20 and Poly1305 for ESP

AEAD_CHACHA20_POLY1305 ([RFC7539]) is a combined mode algorithm, or AEAD. Usage follows the AEAD construction in Section 2.8 of RFC 7539:

- The Initialization Vector (IV) is 64 bits and is used as part of the nonce. The IV MUST be unique for each invocation for a particular security association (SA) but does not need to be unpredictable. The use of a counter or a linear feedback shift register (LFSR) is RECOMMENDED.
- A 32-bit Salt is prepended to the 64-bit IV to form the 96-bit nonce. The salt is fixed per SA, and it is not transmitted as part of the ESP packet.
- The encryption key is 256 bits.
- The Internet Key Exchange Protocol generates a bitstring called KEYMAT using a pseudorandom function (PRF). That KEYMAT is divided into keys for encryption, message authentication, and whatever else is needed. The KEYMAT requested for each ChaCha20-Poly1305 key is 36 octets. The first 32 octets are the 256-bit ChaCha20 key, and the remaining 4 octets are used as the Salt value in the nonce.

The ChaCha20 encryption algorithm requires the following parameters:

- a 256-bit key
- a 96-bit nonce
- a 32-bit Initial Block Counter

For ESP, we set these as follows:

- The key is set as mentioned above.
- The 96-bit nonce is formed from a concatenation of the 32-bit Salt and the 64-bit IV, as described above.
- The Initial Block Counter is set to one (1). The reason that one is used for the initial counter rather than zero is that zero is reserved for generating the one-time Poly1305 key (see below).

As the ChaCha20 block function is not applied directly to the plaintext, no padding should be necessary. However, in keeping with the specification in RFC 4303, the plaintext always has a pad length octet and a Next Header octet, and it may require padding octets so as to align the buffer to an integral multiple of 4 octets.

The same key and nonce, along with a block counter of zero, are passed to the ChaCha20 block function, and the top 256 bits of the result are used as the Poly1305 key.

Finally, the Poly1305 function is run on the data to be authenticated, which is, as specified in Section 2.8 of [RFC7539], a concatenation of the following in the order below:

- The Authenticated Additional Data (AAD); see Section 2.1.
- Zero-octet padding that rounds the length up to 16 octets. This is 4 or 8 octets depending on the length of the AAD.
- The ciphertext.
- Zero-octet padding that rounds the total length up to an integral multiple of 16 octets.

- The length of the AAD in octets (as a 64-bit integer encoded in little-endian byte order).
- The length of the ciphertext in octets (as a 64-bit integer encoded in little-endian byte order).

The 128-bit output of Poly1305 is used as the tag. All 16 octets are included in the packet.

```
* The figure below is a copy of Figure 2 in RFC 4303:
*
* 0          1          2          3
* 0 1 2 3 4 5 6 7 0 1 2 3 4 5 6 7 0 1 2 3 4 5 6 7
* +-----+-----+-----+
* |           Security Parameters Index (SPI)           |
* +-----+-----+-----+
* |           Sequence Number                         |
* +-----+-----+-----+
* |           IV (optional)                          | ^ p
* +-----+-----+-----+-----+-----+-----+-----+-----+
* |           Rest of Payload Data (variable)       | | y
* ~          ~          ~          ~          ~          ~          ~          ~
* |           |           |           |           |           |           |           |
* |           |           |           |           |           |           |           |
* |           |           |           |           |           |           |           |
* |           +-----+-----+-----+-----+-----+-----+-----+-----+-----+
* |           |           TFC Padding * (optional, variable) | v d
* +-----+-----+-----+-----+-----+-----+-----+-----+-----+
* |           |           Padding (0-255 bytes)           |
* +-----+-----+-----+-----+-----+-----+-----+-----+-----+
* |           |           Pad Length   | Next Header |
* +-----+-----+-----+-----+-----+-----+-----+-----+-----+
* |           Integrity Check Value-ICV (variable)      |
* ~          ~          ~          ~          ~          ~          ~
* +-----+-----+-----+-----+-----+-----+-----+-----+-----+
```

- The IV field is 64 bits. It is the final 64 bits of the 96-bit nonce. If the counter method is used for generating unique IVs, then the final 32 bits of the IV will be equal to the Sequence Number field
- The length of the Padding field need not exceed 4 octets. However, neither RFC 4303 nor this specification require using the minimal padding length
- The Integrity Check Value field contains the 16-octet tag

AAD Construction

The construction of the Additional Authenticated Data (AAD) is similar to the one in [RFC4106]. For security associations (SAs) with 32-bit sequence numbers, the AAD is 8 octets: a 4-octet SPI followed by a 4-octet sequence number ordered exactly as it is in the packet. For SAs with an Extended Sequence Number (ESN), the AAD is 12 octets: a 4-octet SPI followed by an 8-octet sequence number as a 64-bit integer in big-endian byte order.

Use in IKEv2

AEAD algorithms can be used in IKE, as described in [RFC5282]. More specifically:

- The Encrypted Payload is as described in Section 3 of RFC 5282.
- The ChaCha20-Poly1305 keying material is derived similarly to ESP: 36 octets are requested for each of SK_ei and SK_er, of which the first 32 form the key and the last 4 form the salt. No octets are requested for SK_ai and SK_ar.
- The IV is 64 bits, as described in Section 2, and is included explicitly in the Encrypted payload.
- The sender SHOULD include no padding and set the Pad Length field to zero. The receiver MUST accept any length of padding
- The AAD is as described in Section 5.1 of RFC 5282, so it is 32 octets (28 for the IKEv2 header plus 4 octets for the encrypted payload header), assuming no unencrypted payloads

Negotiation in IKEv2

When negotiating the ChaCha20-Poly1305 algorithm for use in IKE or IPsec, the value ENCR_CHACHA20_POLY1305 (28) should be used in the transform substructure of the SA payload as the ENCR (type 1) transform ID. As

with other AEAD algorithms, INTEG (type 3) transform substructures MUST NOT be specified, or just one INTEG transform MAY be included with value NONE (0).

Security Considerations

The ChaCha20 cipher is designed to provide 256-bit security.

The Poly1305 authenticator is designed to ensure that forged messages are rejected with a probability of $1/(n(2^{102}))$ for a $16n$ -octet message, even after sending 2^{64} legitimate messages, so it is SUF-CMA (strong unforgeability against chosen-message attacks) in the terminology of [AE].

The most important security consideration in implementing this document is the uniqueness of the nonce used in ChaCha20. The nonce should be selected uniquely for a particular key, but unpredictability of the nonce is not required. Counters and LFSRs are both acceptable ways of generating unique nonces.

Another issue with implementing these algorithms is avoiding side channels. This is trivial for ChaCha20, but requires some care for Poly1305. Considerations for implementations of these algorithms are in [RFC7539].

The Salt value in used nonce construction in ESP and IKEv2 is derived from the keystream, same as the encryption key. It is never transmitted on the wire, but the security of the algorithm does not depend on its secrecy. Thus, implementations that keep keys and other secret material within some security boundary MAY export the Salt from the security boundary. This may be useful if the API provided by the library accepts the nonce as a parameter rather than the IV.

IANA Considerations

IANA has assigned the value 28 as a transform identifier for the algorithm described in this document in the "Transform Type 1 Encryption Algorithm Transform IDs" registry with name ENCR_CHACHA20_POLY1305 and this document as reference for both ESP and IKEv2.

7.61.2.7 IPsec Cipher and Authentication Synopsis

- NOTE - CHACHA20 and POLY1305 always go together as an AEAD mode and are identified as CHACHA20_POLY1305
- GCM and CCM are also AEAD modes and require no additional authentication mode

For API Documentation:

See Also

[ProtocolPP::jipsec](#)
[ProtocolPP::jipsecsa](#)
[ProtocolPP::jikey2](#)
[ProtocolPP::jikey2sa](#)
[ProtocolPP::jprotocol](#)
[ProtocolPP::jicmp](#)
[ProtocolPP::jmodes](#)
[ProtocolPP::jreplay](#)

For Additional Documentation:

See Also

[jipsec](#)
[jipsecsa](#)
[jikey2](#)
[jikey2sa](#)
[jprotocol](#)
[jicmp](#)
[jmodes](#)
[jreplay](#)

Protocol++ modified by : John Peter Greninger © John Peter Greninger 2015-2019 All Rights Reserved

All copyrights and trademarks are the property of their respective owners

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer
- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution
- Any and all modifications must be returned to John Peter Greninger at GitHub.com <https://github.com/jpgreninger/protocolpp> for evaluation. Inclusion of modifications in the source code shall be determined solely by John Peter Greninger. Failure to provide modifications shall render this license NULL and VOID and revoke any rights to use of Protocol++
- Commercial use requires a fee-based license obtainable at www.protocolpp.com
- Academic use requires written and notarized permission from John Peter and Sheila Rocha Greninger
- US Copyrights at <https://www.copyright.gov/>
 - TXu002059872 (Version 1.0.0)
 - TXu002066632 (Version 1.2.7)
 - TXu002082674 (Version 1.4.0)
 - TXu002097880 (Version 2.0.0)

The names of its contributors may not be used to endorse or promote products derived from this software without specific prior written permission

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE

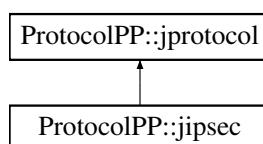
The documentation for this class was generated from the following file:

- [include/jipsec.h](#)

7.62 ProtocolPP::jipsec Class Reference

```
#include <jipsec.h>
```

Inheritance diagram for ProtocolPP::jipsec:



Public Types

- enum `audit_t` {
 INVALIDSA, **FRAGMENT**, **ROLLOVER**, **REPLAY**,
ICV, **FORMAT**, **DUMMY** }

Public Member Functions

- `jipsec` (`std::shared_ptr< jrand >` `&rand`, `std::shared_ptr< jipsecsa >` `&security`)
- virtual `~jipsec` ()
- `void encaps_packet` (`std::shared_ptr< jarray< uint8_t >>` `&input`, `std::shared_ptr< jarray< uint8_t >>` `&output`)
- `void decap_packet` (`std::shared_ptr< jarray< uint8_t >>` `&input`, `std::shared_ptr< jarray< uint8_t >>` `&output`)
- `void set_hdr` (`jarray< uint8_t >` `&hdr`)
- `void set_field` (`field_t` field, `uint64_t` value)
- `jarray< uint8_t >` `get_exthdr` (`iana_t` extension, `iana_t` nh, `jarray< uint8_t >` `&data`, `uint16_t` type_offset=0, `uint8_t` segments=0)
- `jarray< uint8_t >` `get_hdr` ()
- `uint64_t` `get_field` (`field_t` field)
- `uint64_t` `get_field` (`field_t` field, `jarray< uint8_t >` `&header`)
- `void to_xml` (`tinyxml2::XMLPrinter` &myxml, `direction_t` direction)
- `void audit` (`audit_t` type, `jarray< uint8_t >` `&header`)

Additional Inherited Members

7.62.1 Member Enumeration Documentation

7.62.1.1 enum ProtocolPP::jipsec::audit_t

Audit types for IPsec

Parameters

INVALIDSA	- Invalid Security Association
FRAGMENT	- Fragment received when not allowed
ROLLOVER	- Sequence number rolled over
REPLAY	- Anti-Replay error
ICV	- Received ICV was incorrect
FORMAT	- IPv6 JUMBOGRAM formatting errors
DUMMY	- Dummy packet received

Enumerator

INVALIDSA Audit for invalid security association.

FRAGMENT Audit for received fragment when no fragments allowed.

ROLLOVER Audit for sequence number overflow.

REPLAY Audit for anti-replay error.

ICV Audit for incorrect received ICV.

FORMAT Audit for IPv6 JUMBOGRAM formatting errors.

DUMMY Audit for received DUMMY packets.

7.62.2 Constructor & Destructor Documentation

7.62.2.1 ProtocolPP::jipsec::jipsec (std::shared_ptr<jrand> & rand, std::shared_ptr<jipsecsa> & security)

Constructor for IPsec

Parameters

<i>rand</i>	- Random data generation for IVs and padding
<i>security</i>	- Security association (SA) for this IPsec flow

7.62.2.2 virtual ProtocolPP::jipsec::~jipsec() [inline], [virtual]

Standard deconstructor flush and close the auditlog if present

7.62.3 Member Function Documentation

7.62.3.1 void ProtocolPP::jipsec::audit(audit_t *type*, jarray< uint8_t >& *header*)

Audit function

Parameters

<i>type</i>	- Type of Audit
<i>header</i>	- IPsec header to extract audit information

7.62.3.2 void ProtocolPP::jipsec::decap_packet(std::shared_ptr< jarray< uint8_t >> & *input*, std::shared_ptr< jarray< uint8_t >> & *output*) [virtual]

Decap will produce a payload from the packet passed

Parameters

<i>input</i>	- packet to decapsulate with IPsec
<i>output</i>	- packet encapsulated with IPsec

Implements [ProtocolPP::jprotocol](#).

7.62.3.3 void ProtocolPP::jipsec::encap_packet(std::shared_ptr< jarray< uint8_t >> & *input*, std::shared_ptr< jarray< uint8_t >> & *output*) [virtual]

Encap will produce a packet from the payload passed

Parameters

<i>input</i>	- payload to protect with IPsec
<i>output</i>	- packet encapsulated with IPsec

Implements [ProtocolPP::jprotocol](#).

7.62.3.4 jarray<uint8_t> ProtocolPP::jipsec::get_exthdr(iana_t *extension*, iana_t *nh*, jarray< uint8_t >& *data*, uint16_t *type_offset* = 0, uint8_t *segments* = 0)

Add the extension header

Parameters

<i>extension</i>	- extension to add to the IPsec header
<i>nh</i>	- Next header after the current one
<i>data</i>	- Data necessary for the extension header (32-bit length for JUMBOGRAM, Route header, Identification for Fragment, etc.)
<i>type_offset</i>	- Type (8-bits) for Routing header, offset (13-bits) for Fragment header
<i>segments</i>	- Number of segments left for Routing header (8-bits), "More" bit for Fragment header (1-bit)

Returns

extension header(s)

7.62.3.5 uint64_t ProtocolPP::jipsec::get_field (field_t *field*) [virtual]

Retrieve the field from the IP header

Parameters

<i>field</i>	- field to return from the header
--------------	-----------------------------------

Returns

field

Reimplemented from [ProtocolPP::jprotocol](#).

7.62.3.6 uint64_t ProtocolPP::jipsec::get_field (field_t *field*, jarray<uint8_t> & *header*) [virtual]

Retrieve the field from the IP header

Parameters

<i>field</i>	- field to return from the header
<i>header</i>	- header to retrieve the field from

Returns

field

Implements [ProtocolPP::jprotocol](#).

7.62.3.7 jarray<uint8_t> ProtocolPP::jipsec::get_hdr () [virtual]

Retrieve the IP header

Returns

current IP header

Implements [ProtocolPP::jprotocol](#).

7.62.3.8 void ProtocolPP::jipsec::set_field (field_t *field*, uint64_t *value*) [virtual]

Update IP field with the new value

Parameters

<i>field</i>	- field to update
<i>value</i>	- new value for the field

Implements [ProtocolPP::jprotocol](#).

7.62.3.9 void ProtocolPP::jipsec::set_hdr (jarray<uint8_t> & *hdr*) [virtual]

Update the current IP header with a new header

Parameters

hdr - new IP header for this flow

Implements [ProtocolPP::jprotocol](#).

7.62.3.10 void ProtocolIPP::ipsec::to_xml (tinyxml2::XMLPrinter & myxml, direction_t direction) [virtual]

Return the protocol and security fields as XML

Parameters

<i>myxml</i>	- XMLPrinter object
<i>direction</i>	- randomization

Implements [ProtocolPP::jprotocol](#).

The documentation for this class was generated from the following file:

- include/jipsec.h

7.63 jipsecsa Class Reference

```
#include "include/jipsecsa.h"
```

7.63.1 Detailed Description

7.63.2 Encapsulating Security Payload (ESP) Security Association

(See RFC4303 for complete details)

The (outer) protocol header (IPv4, IPv6, or Extension) that immediately precedes the ESP header SHALL contain the value 50 in its Protocol (IPv4) or Next Header (IPv6, Extension) field (see IANA web page at <http://www.iana.org/assignments/protocol-numbers>). Figure 1 illustrates the top-level format of an ESP packet. The packet begins with two 4-byte fields (Security Parameters Index (SPI) and Sequence Number). Following these fields is the Payload Data, which has substructure that depends on the choice of encryption algorithm and mode, and on the use of TFC padding, which is examined in more detail later. Following the Payload Data are Padding and Pad Length fields, and the Next Header field. The optional Integrity Check Value (ICV) field completes the packet.

```

*          1           2           3
* 0 1 2 3 4 5 6 7 0 1 2 3 4 5 6 7 0 1 2 3 4 5 6 7
* +---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
* |                         Security Parameters Index (SPI)           | ^Int.
* +---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
* |                         Sequence Number                      | Iced
* +---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
* |                         Payload Data* (variable)            | | |
* ~ |                                         |                   | ~ | |
* |                                         |                   | Conf.
* +                         +---+---+---+---+---+---+---+---+---+---+---+ | Cov-
* | |                         Padding (0-255 bytes)           | Iced*
* +---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
* | |                         Pad Length | Next Header | v   v
* +---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
* |                         Integrity Check Value-ICV (variable) | |
* ~ |                                         |                   | ~
* |                                         |                   | |
* +---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
*          Figure 1. Top-Level Format of an ESP Packet
*
* * If included in the Payload field, cryptographic synchronization
*   data, e.g., an Initialization Vector (IV, see Section 2.3),
*   usually is not encrypted per se, although it often is referred
*   to as being part of the ciphertext.

```

*

The (transmitted) ESP trailer consists of the Padding, Pad Length, and Next Header fields. Additional, implicit ESP trailer data (which is not transmitted) is included in the integrity computation, as described below.

If the integrity service is selected, the integrity computation encompasses the SPI, Sequence Number, Payload Data, and the ESP trailer (explicit and implicit).

If the confidentiality service is selected, the ciphertext consists of the Payload Data (except for any cryptographic synchronization data that may be included) and the (explicit) ESP trailer.

As noted above, the Payload Data may have substructure. An encryption algorithm that requires an explicit Initialization Vector (IV), e.g., Cipher Block Chaining (CBC) mode, often prefixes the Payload Data to be protected with that value. Some algorithm modes combine encryption and integrity into a single operation; this document refers to such algorithm modes as "combined mode algorithms". Accommodation of combined mode algorithms requires that the algorithm explicitly describe the payload substructure used to convey the integrity data.

Some combined mode algorithms provide integrity only for data that is encrypted, whereas others can provide integrity for some additional data that is not encrypted for transmission. Because the SPI and Sequence Number fields require integrity as part of the integrity service, and they are not encrypted, it is necessary to ensure that they are afforded integrity whenever the service is selected, regardless of the style of combined algorithm mode employed.

When any combined mode algorithm is employed, the algorithm itself is expected to return both decrypted plaintext and a pass/fail indication for the integrity check. For combined mode algorithms, the ICV that would normally appear at the end of the ESP packet (when integrity is selected) may be omitted. When the ICV is omitted and integrity is selected, it is the responsibility of the combined mode algorithm to encode within the Payload Data an ICV-equivalent means of verifying the integrity of the packet.

If a combined mode algorithm offers integrity only to data that is encrypted, it will be necessary to replicate the SPI and Sequence Number as part of the Payload Data.

Finally, a new provision is made to insert padding for traffic flow confidentiality after the Payload Data and before the ESP trailer. Figure 2 illustrates this substructure for Payload Data. (Note: This diagram shows bits-on-the-wire. So even if extended sequence numbers are being used, only 32 bits of the Sequence Number will be transmitted (see Section 2.2.1).)

```

*
*      0          1          2          3
*      0 1 2 3 4 5 6 7 0 1 2 3 4 5 6 7 0 1 2 3 4 5 6 7
*      +-----+-----+-----+-----+
*      |           Security Parameters Index (SPI)           |
*      +-----+-----+-----+-----+
*      |           Sequence Number                         |
*      +-----+-----+-----+-----+
*      |           IV (optional)                         | ^ p
*      +-----+-----+-----+-----+-----+-----+-----+-----+
*      |           Rest of Payload Data (variable)       | | y
*      ~           |           TFC Padding * (optional, variable) | v d
*      +-----+-----+-----+-----+-----+-----+-----+-----+
*      |           |           Padding (0-255 bytes)        |
*      +-----+-----+-----+-----+-----+-----+-----+-----+
*      |           |           Pad Length | Next Header   |
*      +-----+-----+-----+-----+-----+-----+-----+-----+
*      |           Integrity Check Value-ICV (variable)    |
*      ~           |                                           ~
*      |                                           |

```

```

*
*      +-----+
*      | Figure 2. Substructure of Payload Data
*
*      * If tunnel mode is being used, then the IPsec implementation
*      can add Traffic Flow Confidentiality (TFC) padding (see
*      Section 2.4) after the Payload Data and before the Padding
*      (0-255 bytes) field.
*
*
```

If a combined algorithm mode is employed, the explicit ICV shown in Figures 1 and 2 may be omitted (see Section 3.3.2.2 below). Because algorithms and modes are fixed when an SA is established, the detailed format of ESP packets for a given SA (including the Payload Data substructure) is fixed, for all traffic on the SA.

The tables below refer to the fields in the preceding figures and illustrate how several categories of algorithmic options, each with a different processing model, affect the fields noted above. The processing details are described in later sections.

* Table 1. Separate Encryption and Integrity Algorithms

	# of bytes	Requ'd [1]	What Encrypt Covers	What Integ Covers	What is Xmted
SPI	4	M		Y	plain
Seq# (low-order bits)	4	M		Y	plain p
IV	variable	O		Y	plain y
IP datagram [2]	variable	M or D	Y	Y	cipher[3] -l
TFC padding [4]	variable	O	Y	Y	cipher[3] o
Padding	0-255	M	Y	Y	cipher[3] d
Pad Length	1	M	Y	Y	cipher[3]
Next Header	1	M	Y	Y	cipher[3]
Seq# (high-order bits)	4 if ESN [5]			Y	not xmtd
ICV Padding	variable if need			Y	not xmtd
ICV	variable M [6]				plain

[1] M = mandatory; O = optional; D = dummy
[2] If tunnel mode -> IP datagram
[3] If transport mode -> next header and data
[4] ciphertext if encryption has been selected
[5] Can be used only if payload specifies its "real" length
[6] See section 2.2.1
[7] mandatory if a separate integrity algorithm is used

* Table 2. Combined Mode Algorithms

	# of bytes	Requ'd [1]	What Encrypt Covers	What Integ Covers	What is Xmted
SPI	4	M			plain
Seq# (low-order bits)	4	M			plain p
IV	variable	O		Y	plain y
IP datagram [2]	variable	M or D	Y	Y	cipher -l
TFC padding [3]	variable	O	Y	Y	cipher o
Padding	0-255	M	Y	Y	cipher d
Pad Length	1	M	Y	Y	cipher
Next Header	1	M	Y	Y	cipher
Seq# (high-order bits)	4 if ESN [4]			Y	[5]
ICV Padding	variable if need			Y	[5]
ICV	variable O [6]				plain

[1] M = mandatory; O = optional; D = dummy
[2] If tunnel mode -> IP datagram
[3] If transport mode -> next header and data
[4] Can be used only if payload specifies its "real" length
[5] See Section 2.2.1
[6] The algorithm choices determines whether these are transmitted, but in either case, the result is invisible to ESP
[7] The algorithm spec determines whether this field is present

The following subsections describe the fields in the header format. "Optional" means that the field is omitted if the

option is not selected, i.e., it is present in neither the packet as transmitted nor as formatted for computation of an ICV (see Section 2.7). Whether or not an option is selected is determined as part of Security Association (SA) establishment. Thus, the format of ESP packets for a given SA is fixed, for the duration of the SA. In contrast, "mandatory" fields are always present in the ESP packet format, for all SAs.

Note: All of the cryptographic algorithms used in IPsec expect their input in canonical network byte order (see Appendix of RFC 791 [Pos81]) and generate their output in canonical network byte order. IP packets are also transmitted in network byte order.

ESP does not contain a version number, therefore if there are concerns about backward compatibility, they MUST be addressed by using a signaling mechanism between the two IPsec peers to ensure compatible versions of ESP (e.g., Internet Key Exchange (IKEv2) [Kau05]) or an out-of-band configuration mechanism.

Security Parameters Index (SPI)

The SPI is an arbitrary 32-bit value that is used by a receiver to identify the SA to which an incoming packet is bound. The SPI field is mandatory.

For a unicast SA, the SPI can be used by itself to specify an SA, or it may be used in conjunction with the IPsec protocol type (in this case ESP). Because the SPI value is generated by the receiver for a unicast SA, whether the value is sufficient to identify an SA by itself or whether it must be used in conjunction with the IPsec protocol value is a local matter. This mechanism for mapping inbound traffic to unicast SAs MUST be supported by all ESP implementations.

If an IPsec implementation supports multicast, then it MUST support multicast SAs using the algorithm below for mapping inbound IPsec datagrams to SAs. Implementations that support only unicast traffic need not implement this de-multiplexing algorithm.

In many secure multicast architectures (e.g., [RFC3740]), a central Group Controller/Key Server unilaterally assigns the group security association's SPI. This SPI assignment is not negotiated or coordinated with the key management (e.g., IKE) subsystems that reside in the individual end systems that comprise the group. Consequently, it is possible that a group security association and a unicast security association can simultaneously use the same SPI. A multicast-capable IPsec implementation MUST correctly de-multiplex inbound traffic even in the context of SPI collisions.

Each entry in the Security Association Database (SAD) [Ken-Arch] must indicate whether the SA lookup makes use of the destination, or destination and source, IP addresses, in addition to the SPI. For multicast SAs, the protocol field is not employed for SA lookups. For each inbound, IPsec-protected packet, an implementation must conduct its search of the SAD such that it finds the entry that matches the "longest" SA identifier. In this context, if two or more SAD entries match based on the SPI value, then the entry that also matches based on destination, or destination and source, address comparison (as indicated in the SAD entry) is the "longest" match. This implies a logical ordering of the SAD search as follows:

1. Search the SAD for a match on {SPI, destination address, source address}. If an SAD entry matches, then process the inbound ESP packet with that matching SAD entry. Otherwise, proceed to step 2.
2. Search the SAD for a match on {SPI, destination address}. If the SAD entry matches, then process the inbound ESP packet with that matching SAD entry. Otherwise, proceed to step 3.
3. Search the SAD for a match on only {SPI} if the receiver has chosen to maintain a single SPI space for AH and ESP, or on {SPI, protocol} otherwise. If an SAD entry matches, then process the inbound ESP packet with that matching SAD entry. Otherwise, discard the packet and log an auditable event.

In practice, an implementation MAY choose any method to accelerate this search, although its externally visible behavior MUST be functionally equivalent to having searched the SAD in the above order. For example, a software-based implementation could index into a hash table by the SPI. The SAD entries in each hash table bucket's linked list are kept sorted to have those SAD entries with the longest SA identifiers first in that linked list. Those SAD entries having the shortest SA identifiers are sorted so that they are the last entries in the linked list. A hardware-based implementation may be able to effect the longest match search intrinsically, using commonly available Ternary Content-Addressable Memory (TCAM) features.

The indication of whether source and destination address matching is required to map inbound IPsec traffic to SAs MUST be set either as a side effect of manual SA configuration or via negotiation using an SA management protocol, e.g., IKE or Group Domain of Interpretation (GDOI) [RFC3547]. Typically, Source-Specific Multicast (SSM) [HC03] groups use a 3-tuple SA identifier composed of an SPI, a destination multicast address, and source address. An Any-Source Multicast group SA requires only an SPI and a destination multicast address as an identifier.

The set of SPI values in the range 1 through 255 are reserved by the Internet Assigned Numbers Authority (IANA) for future use; a reserved SPI value will not normally be assigned by IANA unless the use of the assigned SPI value is specified in an RFC. The SPI value of zero (0) is reserved for local, implementation-specific use and MUST NOT be sent on the wire. (For example, a key management implementation might use the zero SPI value to mean "No Security Association Exists" during the period when the IPsec implementation has requested that its key management entity establish a new SA, but the SA has not yet been established.)

Sequence Number

This unsigned 32-bit field contains a counter value that increases by one for each packet sent, i.e., a per-SA packet sequence number. For a unicast SA or a single-sender multicast SA, the sender MUST increment this field for every transmitted packet. Sharing an SA among multiple senders is permitted, though generally not recommended. ESP provides no means of synchronizing packet counters among multiple senders or meaningfully managing a receiver packet counter and window in the context of multiple senders. Thus, for a multi-sender SA, the anti-replay features of ESP are not available (see Sections 3.3.3 and 3.4.3.)

The field is mandatory and MUST always be present even if the receiver does not elect to enable the anti-replay service for a specific SA. Processing of the Sequence Number field is at the discretion of the receiver, but all ESP implementations MUST be capable of performing the processing described in Sections 3.3.3 and 3.4.3. Thus, the sender MUST always transmit this field, but the receiver need not act upon it (see the discussion of Sequence Number Verification in the "Inbound Packet Processing" section (3.4.3) below).

The sender's counter and the receiver's counter are initialized to 0 when an SA is established. (The first packet sent using a given SA will have a sequence number of 1; see Section 3.3.3 for more details on how the sequence number is generated.) If anti-replay is enabled (the default), the transmitted sequence number must never be allowed to cycle. Thus, the sender's counter and the receiver's counter MUST be reset (by establishing a new SA and thus a new key) prior to the transmission of the 2^{32} nd packet on an SA.

Extended (64-bit) Sequence Number

To support high-speed IPsec implementations, Extended Sequence Numbers (ESNs) SHOULD be implemented, as an extension to the current, 32-bit sequence number field. Use of an ESN MUST be negotiated by an SA management protocol. Note that in IKEv2, this negotiation is implicit; the default is ESN unless 32-bit sequence numbers are explicitly negotiated. (The ESN feature is applicable to multicast as well as unicast SAs.)

The ESN facility allows use of a 64-bit sequence number for an SA. (See Appendix A, "Extended (64-bit) Sequence Numbers", for details.) Only the low-order 32 bits of the sequence number are transmitted in the plaintext ESP header of each packet, thus minimizing packet overhead. The high-order 32 bits are maintained as part of the sequence number counter by both transmitter and receiver and are included in the computation of the ICV (if the integrity service is selected). If a separate integrity algorithm is employed, the high order bits are included in the implicit ESP trailer, but are not transmitted, analogous to integrity algorithm padding bits. If a combined mode algorithm is employed, the algorithm choice determines whether the high-order ESN bits are transmitted or are included implicitly in the computation. See Section 3.3.2.2 for processing details.

Payload Data

Payload Data is a variable-length field containing data (from the original IP packet) described by the Next Header field. The Payload Data field is mandatory and is an integral number of bytes in length. If the algorithm used to encrypt the payload requires cryptographic synchronization data, e.g., an Initialization Vector (IV), then this data is carried explicitly in the Payload field, but it is not called out as a separate field in ESP, i.e., the transmission of an explicit IV is invisible to ESP. (See Figure 2.) Any encryption algorithm that requires such explicit, per-packet synchronization data MUST indicate the length, any structure for such data, and the location of this data as part of an RFC specifying how the algorithm is used with ESP. (Typically, the IV immediately precedes the ciphertext. See Figure 2.) If such synchronization data is implicit, the algorithm for deriving the data MUST be part of the algorithm definition RFC. (If included in the Payload field, cryptographic synchronization data, e.g., an Initialization Vector (IV), usually is not encrypted per se (see Tables 1 and 2), although it sometimes is referred to as being part of the ciphertext.)

Note that the beginning of the next layer protocol header MUST be aligned relative to the beginning of the ESP header as follows. For IPv4, this alignment is a multiple of 4 bytes. For IPv6, the alignment is a multiple of 8 bytes.

With regard to ensuring the alignment of the (real) ciphertext in the presence of an IV, note the following:

- For some IV-based modes of operation, the receiver treats the IV as the start of the ciphertext, feeding it into the algorithm directly. In these modes, alignment of the start of the (real) ciphertext is not an issue at the receiver.
- In some cases, the receiver reads the IV in separately from the ciphertext. In these cases, the algorithm specification MUST address how alignment of the (real) ciphertext is to be achieved.

Padding (for Encryption)

Two primary factors require or motivate use of the Padding field.

- If an encryption algorithm is employed that requires the plaintext to be a multiple of some number of bytes, e.g., the block size of a block cipher, the Padding field is used to fill the plaintext (consisting of the Payload Data, Padding, Pad Length, and Next Header fields) to the size required by the algorithm.
- Padding also may be required, irrespective of encryption algorithm requirements, to ensure that the resulting ciphertext terminates on a 4-byte boundary. Specifically, the Pad Length and Next Header fields must be right aligned within a 4-byte word, as illustrated in the ESP packet format figures above, to ensure that the ICV field (if present) is aligned on a 4-byte boundary.

Padding beyond that required for the algorithm or alignment reasons cited above could be used to conceal the actual length of the payload, in support of TFC. However, the Padding field described is too limited to be effective for TFC and thus should not be used for that purpose. Instead, the separate mechanism described below (see Section 2.7) should be used when TFC is required.

The sender MAY add 0 to 255 bytes of padding. Inclusion of the Padding field in an ESP packet is optional, subject to the requirements noted above, but all implementations MUST support generation and consumption of padding.

- For the purpose of ensuring that the bits to be encrypted are a multiple of the algorithm's block size (first bullet above), the padding computation applies to the Payload Data exclusive of any IV, but including the ESP trailer fields. If a combined algorithm mode requires transmission of the SPI and Sequence Number to effect integrity, e.g., replication of the SPI and Sequence Number in the Payload Data, then the replicated versions of these data items, and any associated, ICV-equivalent data, are included in the computation of the pad length. (If the ESN option is selected, the high-order 32 bits of the ESN also would enter into the computation, if the combined mode algorithm requires their transmission for integrity.)
- For the purposes of ensuring that the ICV is aligned on a 4-byte boundary (second bullet above), the padding computation applies to the Payload Data inclusive of the IV, the Pad Length, and Next Header fields. If a combined mode algorithm is used, any replicated data and ICV-equivalent data are included in the Payload Data covered by the padding computation.

If Padding bytes are needed but the encryption algorithm does not specify the padding contents, then the following default processing MUST be used. The Padding bytes are initialized with a series of (unsigned, 1-byte) integer values. The first padding byte appended to the plaintext is numbered 1, with subsequent padding bytes making up a monotonically increasing sequence: 1, 2, 3, When this padding scheme is employed, the receiver SHOULD inspect the Padding field. (This scheme was selected because of its relative simplicity, ease of implementation in hardware, and because it offers limited protection against certain forms of "cut and paste" attacks in the absence of other integrity measures, if the receiver checks the padding values upon decryption.)

If an encryption or combined mode algorithm imposes constraints on the values of the bytes used for padding, they MUST be specified by the RFC defining how the algorithm is employed with ESP. If the algorithm requires checking of the values of the bytes used for padding, this too MUST be specified in that RFC.

Pad Length

The Pad Length field indicates the number of pad bytes immediately preceding it in the Padding field. The range of valid values is 0 to 255, where a value of zero indicates that no Padding bytes are present. As noted above, this does not include any TFC padding bytes. The Pad Length field is mandatory.

Next Header

The Next Header is a mandatory, 8-bit field that identifies the type of data contained in the Payload Data field, e.g., an IPv4 or IPv6 packet, or a next layer header and data. The value of this field is chosen from the set of IP Protocol Numbers defined on the web page of the IANA, e.g., a value of 4 indicates IPv4, a value of 41 indicates IPv6, and a value of 6 indicates TCP.

To facilitate the rapid generation and discarding of the padding traffic in support of traffic flow confidentiality (see Section 2.4), the protocol value 59 (which means "no next header") MUST be used to designate a "dummy" packet. A transmitter MUST be capable of generating dummy packets marked with this value in the next protocol field, and a receiver MUST be prepared to discard such packets, without indicating an error. All other ESP header and trailer fields (SPI, Sequence Number, Padding, Pad Length, Next Header, and ICV) MUST be present in dummy packets, but the plaintext portion of the payload, other than this Next Header field, need not be well-formed, e.g., the rest of the Payload Data may consist of only random bytes. Dummy packets are discarded without prejudice.

Implementations SHOULD provide local management controls to enable the use of this capability on a per-SA basis. The controls should allow the user to specify if this feature is to be used and also provide parametric controls; for example, the controls might allow an administrator to generate random-length or fixed-length dummy packets.

DISCUSSION: Dummy packets can be inserted at random intervals to mask the absence of actual traffic. One can also "shape" the actual traffic to match some distribution to which dummy traffic is added as dictated by the distribution parameters. As with the packet length padding facility for Traffic Flow Security (TFS), the most secure approach would be to generate dummy packets at whatever rate is needed to maintain a constant rate on an SA. If packets are all the same size, then the SA presents the appearance of a constant bit rate data stream, analogous to what a link crypto would offer at layer 1 or 2. However, this is unlikely to be practical in many contexts, e.g., when there are multiple SAs active, because it would imply reducing the allowed bandwidth for a site, based on the number of SAs, and that would undermine the benefits of packet switching. Implementations SHOULD provide controls to enable local administrators to manage the generation of dummy packets for TFC purposes.

Traffic Flow Confidentiality (TFC) Padding

As noted above, the Padding field is limited to 255 bytes in length. This generally will not be adequate to hide traffic characteristics relative to traffic flow confidentiality requirements. An optional field, within the payload data, is provided specifically to address the TFC requirement.

An IPsec implementation SHOULD be capable of padding traffic by adding bytes after the end of the Payload Data, prior to the beginning of the Padding field. However, this padding (hereafter referred to as TFC padding) can be added only if the Payload Data field contains a specification of the length of the IP datagram. This is always true in tunnel mode, and may be true in transport mode depending on whether the next layer protocol (e.g., IP, UDP, ICMP) contains explicit length information. This length information will enable the receiver to discard the TFC padding, because the true length of the Payload Data will be known. (ESP trailer fields are located by counting back from the end of the ESP packet.) Accordingly, if TFC padding is added, the field containing the specification of the length of the IP datagram MUST NOT be modified to reflect this padding. No requirements for the value of this padding are established by this standard.

In principle, existing IPsec implementations could have made use of this capability previously, in a transparent fashion. However, because receivers may not have been prepared to deal with this padding, the SA management protocol MUST negotiate this service prior to a transmitter employing it, to ensure backward compatibility. Combined with the convention described in Section 2.6 above, about the use of protocol ID 59, an ESP implementation is capable of generating dummy and real packets that exhibit much greater length variability, in support of TFC.

Implementations SHOULD provide local management controls to enable the use of this capability on a per-SA basis. The controls should allow the user to specify if this feature is to be used and also provide parametric controls for the feature.

Integrity Check Value (ICV)

The Integrity Check Value is a variable-length field computed over the ESP header, Payload, and ESP trailer fields. Implicit ESP trailer fields (integrity padding and high-order ESN bits, if applicable) are included in the ICV computation. The ICV field is optional. It is present only if the integrity service is selected and is provided by either a separate integrity algorithm or a combined mode algorithm that uses an ICV. The length of the field is specified by

the integrity algorithm selected and associated with the SA. The integrity algorithm specification MUST specify the length of the ICV and the comparison rules and processing steps for validation.

For API Documentation:

See Also

[ProtocolPP::jsecass](#)
[ProtocolPP::jprotocol](#)
[ProtocolPP::jicmp](#)
[ProtocolPP::jmodes](#)
[ProtocolPP::sm4](#)

For Additional Documentation:

See Also

[jsecass](#)
[jprotocol](#)
[jicmp](#)
[jmodes](#)
[sm4](#)

Protocol++ modified by : John Peter Greninger © John Peter Greninger 2015-2019 All Rights Reserved

All copyrights and trademarks are the property of their respective owners

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer
- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution
- Any and all modifications must be returned to John Peter Greninger at GitHub.com <https://github.com/jpgreninger/protocolpp> for evaluation. Inclusion of modifications in the source code shall be determined solely by John Peter Greninger. Failure to provide modifications shall render this license NULL and VOID and revoke any rights to use of Protocol++
- Commercial use requires a fee-based license obtainable at www.protocolpp.com
- Academic use requires written and notarized permission from John Peter and Sheila Rocha Greninger
- US Copyrights at <https://www.copyright.gov/>
 - TXU002059872 (Version 1.0.0)
 - TXU002066632 (Version 1.2.7)
 - TXU002082674 (Version 1.4.0)
 - TXU002097880 (Version 2.0.0)

The names of its contributors may not be used to endorse or promote products derived from this software without specific prior written permission

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT

OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE

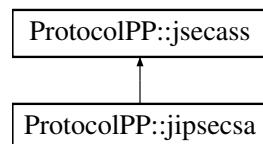
The documentation for this class was generated from the following file:

- [include/jipsecsa.h](#)

7.64 ProtocolPP::jipsecsa Class Reference

```
#include <jipsecsa.h>
```

Inheritance diagram for ProtocolPP::jipsecsa:



Public Member Functions

- [jipsecsa \(\)](#)
- [jipsecsa \(direction_t dir, iana_t ver, ipmode_t mode, uint32_t spi, uint32_t seqnum, uint32_t extseqnum, unsigned int arlen, jarray< uint8_t > arwin, cipher_t cipher, unsigned int ckeylen, std::shared_ptr< jarray< uint8_t >> cipherkey, auth_t auth, unsigned int akeylen, std::shared_ptr< jarray< uint8_t >> authkey, unsigned int ivlen, std::shared_ptr< jarray< uint8_t >> iv, unsigned int saltlen, std::shared_ptr< jarray< uint8_t >> salt, uint64_t bytecnt, uint64_t lifetime, bool seqnumovrflw, bool statefulfrag, bool bypassdf, bool bypassdscp, bool nat, bool nchk, uint8_t dsecn, uint8_t ttl, uint8_t flags, uint16_t natsrc, uint16_t natdst, uint16_t id, uint32_t label, uint16_t fragoff, bool morefrag, uint32_t fragid, unsigned int mtu, jarray< uint8_t > src, jarray< uint8_t > dst, jarray< uint8_t > exthdr, iana_t nh, unsigned int icvlen, unsigned int hdrhlen, unsigned int tfclen, bool usext, bool randiv, bool jumbogram, bool audit, std::string auditlog\)](#)
- [jipsecsa \(jipsecsa &rhs\)](#)
- [jipsecsa \(std::shared_ptr< jipsecsa > &rhs\)](#)
- virtual [~jipsecsa \(\)](#)
- template<typename T >
void [set_field \(field_t field, T value\)](#)
- template<typename T >
T [get_field \(field_t field\)](#)
- void [to_xml \(tinyxml2::XMLPrinter &myxml, direction_t direction\)](#)

7.64.1 Constructor & Destructor Documentation

7.64.1.1 ProtocolPP::jipsecsa::jipsecsa()

Standard constructor with defaults

7.64.1.2 ProtocolPP::jipsecsa::jipsecsa(direction_t dir, iana_t ver, ipmode_t mode, uint32_t spi, uint32_t seqnum, uint32_t extseqnum, unsigned int arlen, jarray< uint8_t > arwin, cipher_t cipher, unsigned int ckeylen, std::shared_ptr< jarray< uint8_t >> cipherkey, auth_t auth, unsigned int akeylen, std::shared_ptr< jarray< uint8_t >> authkey, unsigned int ivlen, std::shared_ptr< jarray< uint8_t >> iv, unsigned int saltlen, std::shared_ptr< jarray< uint8_t >> salt, uint64_t bytecnt, uint64_t lifetime, bool seqnumovrflw, bool statefulfrag, bool bypassdf, bool bypassdscp, bool nat, bool nchk, uint8_t dsecn, uint8_t ttl, uint8_t flags, uint16_t natsrc, uint16_t natdst, uint16_t id, uint32_t label, uint16_t fragoff, bool morefrag, uint32_t fragid, unsigned int mtu, jarray< uint8_t > src, jarray< uint8_t > dst, jarray< uint8_t > exthdr, iana_t nh, unsigned int icvlen, unsigned int hdrhlen, unsigned int tfclen, bool usext, bool randiv, bool jumbogram, bool audit, std::string auditlog)

Copyright 2017-2019 John Peter Greninger : Generated on Sat Aug 31 2019 22:28:11 for Protocol++ (ProtocolPP) 3.0.1 by Doxygen

See RFC4301 for required fields and their meanings (Section 4.4.2.1)

Security Association for IPsec. Initialization Vectors are generated randomly using the Mersenne Twister algorithm or are passed to the SA during key negotiation depending on the randiv setting.

Parameters

<i>dir</i>	- Direction of processing (ENCAP or DECAP)
<i>ver</i>	- Version of IPsec (IPV4 or IPV6)
<i>spi</i>	- Security parameters index
<i>seqnum</i>	- Initial sequence number
<i>extseqnum</i>	- Initial value of the extended sequence number
<i>arlen</i>	- Number of packets to track in replay window
<i>arwin</i>	- Anti-replay window for tracking packets
<i>cipher</i>	- Encryption algorithm to use with IPsec
<i>ckeylen</i>	- Length of the cipher key
<i>cipherkey</i>	- Key for the encryption algorithm
<i>auth</i>	- Authentication algorithm to use with IPsec
<i>akeylen</i>	- Length of the authentication key
<i>authkey</i>	- Key for the authentication algorithm
<i>ivlen</i>	- Length of the initialization vector (IV)
<i>iv</i>	- Passed in initializaton vector (IV)
<i>saltlen</i>	- Length of the IPsec salt material
<i>salt</i>	- IPsec salt material
<i>bytecnt</i>	- Lifetime implementation of encrypted bytes
<i>lifetime</i>	- Time for this SA to live before it expires
<i>mode</i>	- Mode of operation, either TUNNEL or TRANSPORT
<i>seqnumovrlw</i>	- Allow sequence number overflow in ENCAP
<i>statefulfrag</i>	- Indicates whether stateful fragment checking applies to this SA
<i>bypassdf</i>	- Do not copy the DF bit from inner to outer header (both IPv4)
<i>bypassdscp</i>	- Do not copy the DSCP field from inner to outer header
<i>nat</i>	- Perform NAT-T on packets
<i>nchk</i>	- Update the UDP checksum if NAT-T is enabled
<i>natsrc</i>	- Source port for NAT-T
<i>natdst</i>	- Destination port for NAT-T
<i>dsecn</i>	- Traffic class bits (DS and ECN)
<i>ttl</i>	- Time-To-Live (or HOP limit)
<i>flags</i>	- Flags for IPv4
<i>id</i>	- Identification field for IPv4
<i>label</i>	- Flow label for IPv6
<i>fragoff</i>	- Fragment offset for IPv4
<i>morefrag</i>	- More fragments flag in IPV6_FRAG extension header 1-more fragments after this one, 0-last fragment
<i>fragid</i>	- ID for the fragmentation segment
<i>mtu</i>	- Maximum Transmission Unit (MTU) for the path
<i>src</i>	- Source address
<i>dst</i>	- Destination address
<i>exthdr</i>	- Extension headers for IPv6
<i>audit</i>	- Enable auditing
<i>auditlog</i>	- path to the audit log

Fields required for this implementation:

Parameters

<i>nh</i>	- Next Header (NH) value for the payload
<i>icvlen</i>	- Length of the ICV tag
<i>hdrlen</i>	- Length of the IP header
<i>tfclen</i>	- Length of the TFC padding, if any

		field type
<i>usext</i>	- Use extended sequence number (default=TRUE)	direction_t
<i>randiv</i>	- Use a random IV instead of the passed in one	iana_t
<i>jumbogram</i>	- Nodal support for IPv6 JUMBOGRAMs	

7.64.1.3 ProtocolPP::jipsecsa::jipsecsa (*jipsecsa* & *rhs*)

Constructor for IPsec

Parameters

<i>rhs</i>	- Security association (SA) for this IPsec flow	ipmode_t
------------	---	----------

7.64.1.4 ProtocolPP::jipsecsa::jipsecsa (std::shared_ptr<*jipsecsa*> & *rhs*)

Constructor for IPsec

Parameters

<i>rhs</i>	- Security association (SA) for this IPsec flow	cipher_t
------------	---	----------

7.64.1.5 virtual ProtocolPP::jipsecsa::~jipsecsa () [inline], [virtual]

Standard deconstructor flush and close the auditlog if present

7.64.2 Member Function Documentation

7.64.2.1 template<typename T> T ProtocolPP::jipsecsa::get_field (field_t *field*)

Retrieve the field from the IPsec security association

Due to their dynamic nature, some fields are only available in jip which include the following fields

- LENGTH
- CHECKSUM

Parameters

<i>field</i>	- field to return from the security association	std::shared_ptr<jarray<uint8_t>>
--------------	---	----------------------------------

Returns

field

7.64.2.2 template<typename T> void ProtocolPP::jipsecsa::set_field (field_t *field*, T *value*)

Update IPsec field with the new value

Due to their dynamic nature, some fields are only available in jip which include the following fields

- LENGTH
- CHECKSUM

Parameters

<i>field</i>	- field to update
<i>value</i>	- new value for the field

7.64.2.3 void ProtocolPP::jipsecrsa::to_xml (tinyxml2::XMLPrinter & *myxml*, direction_t *direction*) [virtual]

Return the protocol and security fields as XML

Parameters

<i>myxml</i>	- XMLPrinter object
<i>direction</i>	- randomization

Implements [ProtocolPP::jsecass](#).

The documentation for this class was generated from the following file:

- [include/jipsecrsa.h](#)

7.65 InterfacePP::jlogger Class Reference

```
#include "include/jlogger.h"
```

Public Types

- enum **asciicolor** {
 BLACK =30, **RED** =31, **GREEN** =32, **YELLOW** =33,
BLUE =34, **MAGENTA** =35, **CYAN** =36, **WHITE** =37,
BRIGHTBLACK =90, **BRIGHTRED** =91, **BRIGHTGREEN** =92, **BRIGHTYELLOW** =93,
BRIGHTBLUE =94, **BRIGHTMAGENTA** =95, **BRIGHTCYAN** =96, **BRIGHTWHITE** =97 }
- enum **severity_type** {
 info =1, **debug**, **warning**, **error**,
fatal, **pass** }

Public Member Functions

- **jlogger** (const std::string &*name*, bool *logstdout*=false, int *loglvl*=3, **jlogger::asciicolor** *debugclr*=**jlogger::asciicolor::MAGENTA**, **jlogger::asciicolor** *infoclr*=**jlogger::asciicolor::BLACK**, **jlogger::asciicolor** *warningclr*=**jlogger::asciicolor::YELLOW**, **jlogger::asciicolor** *errorclr*=**jlogger::asciicolor::RED**, **jlogger::asciicolor** *fatalclr*=**jlogger::asciicolor::RED**, **jlogger::asciicolor** *passclr*=**jlogger::asciicolor::GREEN**)
- template<typename *jlogger::severity_type* , typename... *Args*>
void print (*Args...**args*)
- virtual ~**jlogger** ()
standard deconstructor
- **void set_color** (std::string *color*, **jlogger::asciicolor** *jcolor*)
- **jlogger::asciicolor get_color** (std::string *color*)

Static Public Member Functions

- static std::string **toStr** (**jlogger::asciicolor** *jcolor*)
- static **jlogger::asciicolor toEnum** (std::string *color*)

7.65.1 Detailed Description

```
#ifdef DARK_THEME line.insert(0, "\033[1;34m "); #else line.insert(0, "\033[1;30m "); #endif } else if (line.find("<DEBUG>") != std::string::npos) { #ifdef DARK_THEME line.insert(0, "\033[1;35m "); #else line.insert(0, "\033[1;34m "); #endif } else if (line.find("<WARN>") != std::string::npos) { #ifdef LIGHT_THEME line.insert(0, "\033[0;34m "); #else line.insert(0, "\033[1;33m "); #endif } else if (line.find("<ERROR>") != std::string::npos) { #ifdef LIGHT_THEME line.insert(0, "\033[1;31m "); #else line.insert(0, "\033[1;31m "); #endif } else if (line.find("<FATAL>") != std::string::npos) { #ifdef DARK_THEME line.insert(0, "\033[1;35m "); #else line.insert(0, "\033[1;35m "); #endif } else if (line.find("<PASS>") != std::string::npos) { #ifdef DARK_THEME line.insert(0, "\033[1;32m "); #else line.insert(0, "\033[1;32m "); #endif } else { #ifdef DARK_THEME line.insert(0, "\033[1;34m "); #else line.insert(0, "\033[1;34m "); #endif }
```

7.65.2 Member Enumeration Documentation

7.65.2.1 enum InterfacePP::jlogger::asciicolor

Color enums

Parameters

BLACK	- Standard black
RED	- Standard red
GREEN	- Standard green
YELLOW	- Standard yellow
BLUE	- Standard blue
MAGENTA	- Standard magenta
CYAN	- Standard cyan
WHITE	- Standard white
BRIGHTBLACK	- Bold black
BRIGHTRED	- Bold red
BRIGHTGREEN	- Bold green
BRIGHTYELLOW	- Bold yellow
BRIGHTBLUE	- Bold blue
BRIGHTMAGENTA	- Bold magenta
BRIGHTCYAN	- Bold cyan
BRIGHTWHITE	- Bold white

Enumerator

BLACK Standard black.

RED Standard red.

GREEN Standard green.

YELLOW Standard yellow.

BLUE Standard blue.

MAGENTA Standard magenta.

CYAN Standard cyan.

WHITE Standard white.

BRIGHTBLACK Bold black.

BRIGHTRED Bold red.

BRIGHTGREEN Bold green.

BRIGHTYELLOW Bold yellow.

BRIGHTBLUE Bold blue.

BRIGHTMAGENTA Bold magenta.

BRIGHTCYAN Bold cyan.

BRIGHTWHITE Bold white.

7.65.2.2 enum InterfacePP::jlogger::severity_type

severity type

Parameters

info	- information level of logging
debug	- debug level of logging
warning	- warning level of logging
error	- error level of logging
fatal	- fatal level of logging
pass	- PASS banner level of logging

Enumerator

info information level of logging

debug debug level of logging

warning warning level of logging

error error level of logging

fatal fatal level of logging

pass PASS banner.

7.65.3 Constructor & Destructor Documentation

```
7.65.3.1 InterfacePP::jlogger ( const std::string & name, bool logstdout = false, int loglvl = 3,
                                jlogger::asciicolor debugclr = jlogger::asciicolor::MAGENTA, jlogger::asciicolor
                                infoclr = jlogger::asciicolor::BLACK, jlogger::asciicolor warningclr = jlogger::-
                                :asciicolor::YELLOW, jlogger::asciicolor errorclr = jlogger::asciicolor::RED,
                                jlogger::asciicolor fatalclr = jlogger::asciicolor::RED, jlogger::asciicolor passclr =
                                jlogger::asciicolor::GREEN )
```

logger class

Parameters

name	- name of the output log file
logstdout	- print logging to stdout as well as file
loglvl	- Level of logging (4-<DEBUG, INFO, WARN, ERROR, FATAL, PASS>, 3[default]-<INFO, WARN, ERROR, FATAL, PASS>, 2-<WARN, ERROR, FATAL, PASS>, 1-<ERROR, FATAL, PASS>, 0-NO LOGGING)
debugclr	- Color for the DEBUG messages (default: MAGENTA)
infoclr	- Color for the INFO messages (default: BLACK)
warningclr	- Color for the WARN messages (default: YELLOW)
errorclr	- Color for the ERROR messages (default:RED)
fatalclr	- Color for the FATAL messages (default:RED)
passclr	- Color for the PASS messages (default:GREEN)

7.65.3.2 virtual InterfacePP::jlogger::~jlogger() [virtual]

standard deconstructor

7.65.4 Member Function Documentation

7.65.4.1 `jlogger::asciicolor InterfacePP::jlogger::get_color (std::string color)`

Obtain current color for logging field

Parameters

<code>color</code>	- Color to set values are DEBUGCLR, INFOCLR, WARNCLR, ERRCLR, FATALCLR, PASS-CLR
--------------------	--

Returns

Returns the `jlogger::asciicolor` Enum value of the color

7.65.4.2 `template<jlogger::severity_type severity, typename... Args> void InterfacePP::jlogger::print (Args... args)`

print function for logger

Parameters

<code>Args...args</code>	- variable number of arguments to print
--------------------------	---

7.65.4.3 `void InterfacePP::jlogger::set_color (std::string color, jlogger::asciicolor jcolor)`

Change colors for the logging output

Parameters

<code>color</code>	- Color to set values are DEBUGCLR, INFOCLR, WARNCLR, ERRCLR, FATALCLR, PASS-CLR
<code>jcolor</code>	- Enum value of new color

7.65.4.4 `static jlogger::asciicolor InterfacePP::jlogger::toEnum (std::string color) [static]`

Convert string to `jlogger::asciicolor` Enum

Parameters

<code>color</code>	- Color to convert
--------------------	--------------------

Returns

Returns the `jlogger::asciicolor` Enum value of the color

7.65.4.5 `static std::string InterfacePP::jlogger::toStr (jlogger::asciicolor jcolor) [static]`

Convert `jlogger::asciicolor` Enum to a string representation

Parameters

<code>jcolor</code>	- Enum to convert
---------------------	-------------------

The documentation for this class was generated from the following file:

- `jlogger/include/jlogger.h`

7.66 ilto Class Reference

	Description
	SRBs
#include "include.h"	DRBs, if configured by upper layers (pdcp-SN-Size)
	DRBs, if configured by upper layers (pdcp-SN-Size)
	DRBs, if configured by upper layers (pdcp-SN-Size)
7.66.1 Detailed Description	DRBs
	DRBs, if configured by upper layers (pdcp-SN-Size)

7.66.2 Long Term Evolution (LTE)

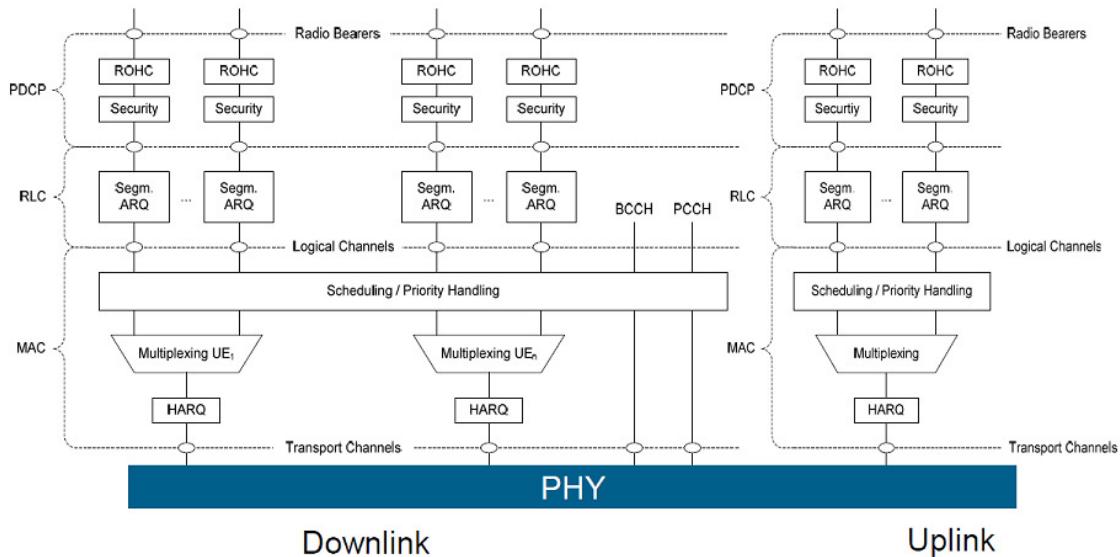
Table 7.40: *PDCPSNLength***Figure 3.11. LTE Protocol Stack**

Figure 7.75: LTE Protocol Stack at the Data-Link Layer [1]

See PDCP_LTE_v14_1_0 specification on www.3gpp.org

If not otherwise mentioned in the definition of each field then the bits in the parameters shall be interpreted as follows: the left most bit string is the first and most significant and the right most bit is the last and least significant bit

Unless otherwise mentioned, integers are encoded in standard binary encoding for unsigned integers. In all cases the bits appear ordered from MSB to LSB when read in the PDU

7.66.2.1 PDCP SN

Length: 5, 7, 12, 15, 16, or 18 bits as indicated in the following table except for NB-IoT which uses 7 bit PDCP SN for DRB.

7.66.2.2 Data

Length: Variable

The Data field may include either one of the following:

- Uncompressed PDCP SDU (user plane data, or control plane data); or
- Compressed PDCP SDU (user plane data only)

7.66.2.3 MAC-I

Length: 32 bits

The MAC-I field carries a message authentication code calculated as specified in subclause 5.7 For control plane data that are not integrity protected, the MAC-I field is still present and should be padded with padding bits set to 0

Bit
0
1

7.66.2.4 COUNT

Length: 32 bits

For ciphering and integrity a COUNT value is maintained. The COUNT value is composed of a HFN and the PDCP SN. The length of the PDCP SN is configured by upper layers

The size of the HFN part in bits is equal to 32 minus the length of the PDCP SN.

NOTE: When performing comparison of values related to COUNT, the UE takes into account that COUNT is a 32-bit value, which may wrap around (e.g., COUNT value of 232 - 1 is less than COUNT value of 0).

Bit
0
1

7.66.2.5 R

Length: 1 bit

Reserved. In this version of the specification reserved bits shall be set to 0. Reserved bits shall be ignored by the receiver

7.66.2.6 D/C

Length: 1 bit

7.66.2.7 PDU type

Length: 3 bits

7.66.2.8 FMS

Length: 12 bits when a 12 bit SN length is used, 15 bits when a 15 bit SN length is used, and 18 bits when an 18 bit SN length is used

PDCP SN of the first missing PDCP SDU

7.66.2.9 Bitmap

Length: Variable

The length of the bitmap field can be 0

The MSB of the first octet of the type "Bitmap" indicates whether or not the PDCP SDU with the SN (FMS + 1) modulo (Maximum_PDCP_SN + 1) has been received and, optionally decompressed correctly. The LSB of the first octet of the type "Bitmap" indicates whether or not the PDCP SDU with the SN (FMS + 8) modulo (Maximum_PDCP_SN + 1) has been received and, optionally decompressed correctly

The UE fills the bitmap indicating which SDUs are missing (unset bit - '0'), i.e. whether an SDU has not been received or optionally has been received but has not been decompressed correctly, and which SDUs do not need retransmission (set bit - '1'), i.e. whether an SDU has been received correctly and may or may not have been decompressed correctly

7.66.2.10 Interspersed ROHC feedback packet

Length: Variable

Contains one ROHC packet with only feedback, i.e. a ROHC packet that is not associated with a PDCP SDU as defined in subclause 5.5.4.

7.66.2.11 PGK Index

Length: 5 bits

5 LSBs of PGK Identity as specified in [13]

7.66.2.12 PTK Identity

Length: 16 bits

PTK Identity as specified in [13].

7.66.2.13 SDU Type

Length: 3 bits

PDCP SDU type, i.e. Layer-3 Protocol Data Unit type as specified in [14]. PDCP entity may handle the SDU differently per SDU Type, e.g. header compression is applicable to IP SDU but not ARP SDU and Non-IP SDU

7.66.2.14 KD-sess ID

Length: 16 bits

KD-sess Identity as specified in [13].

7.66.2.15 NMP

Length: 12 bits when a 12 bit SN length is used, 15 bits when a 15 bit SN length is used, and 18 bits when an 18 bit SN length is used. Number of missing PDCP SDU(s) with associated COUNT value below the associated COUNT value corresponding to HRW, starting from and including the associated COUNT value corresponding to FMS

7.66.2.16 HRW

Length: 12 bits when a 12 bit SN length is used, 15 bits when a 15 bit SN length is used and 18 bits when an 18 bit SN length is used

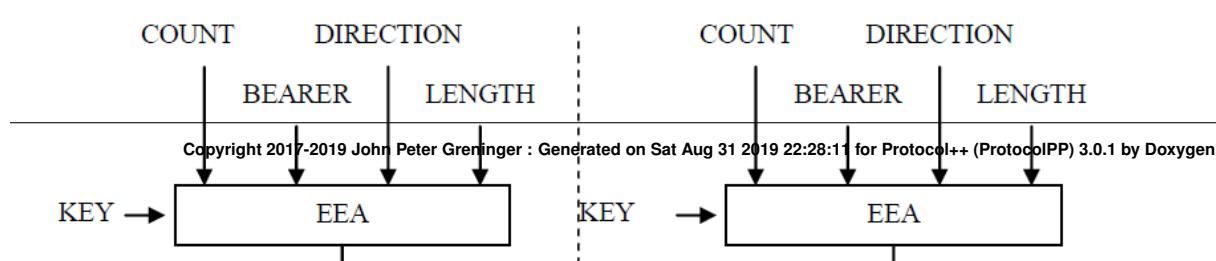
PDCP SN of the PDCP SDU received on WLAN with highest associated PDCP COUNT value

7.66.2.17 P

Length: 1 bit

Polling indication. Set to 1 when eNB triggers a PDCP status report or LWA status report for LWA

7.66.2.18 Encryption



Parameters

<i>security</i>	- Parameters to setup a LTE flow
-----------------	----------------------------------

7.67.1.2 virtual ProtocolPP::jlte::~jlte() [inline], [virtual]

Standard deconstructor.

7.67.2 Member Function Documentation

7.67.2.1 void ProtocolPP::jlte::decap_packet (std::shared_ptr<jarray<uint8_t>> &*input*, std::shared_ptr<jarray<uint8_t>> &*output*) [virtual]

This function is for use with the constructor without a file handle. Decap will produce a payload from the packet passed in check the flags, and update the window

Parameters

<i>input</i>	- LTE packet to decapsulate
<i>output</i>	- decapsulated payload

Implements [ProtocolPP::jprotocol](#).

7.67.2.2 void ProtocolPP::jlte::encap_packet (std::shared_ptr<jarray<uint8_t>> &*input*, std::shared_ptr<jarray<uint8_t>> &*output*) [virtual]

This function is for use with the constructor without a file handle. Encap will produce a packet from the payload passed in

Parameters

<i>input</i>	- payload to encapsulate with LTE
<i>output</i>	- encapsulated LTE packet

Implements [ProtocolPP::jprotocol](#).

7.67.2.3 uint64_t ProtocolPP::jlte::get_field (field_t *field*) [virtual]

Retrieves the field from the LTE security association

Parameters

<i>field</i>	- field from LTE header to retrieve
--------------	-------------------------------------

Returns

field from the LTE security association

Reimplemented from [ProtocolPP::jprotocol](#).

7.67.2.4 uint64_t ProtocolPP::jlte::get_field (field_t *field*, jarray<uint8_t> &*header*) [virtual]

Retrieves the field from the LTE header

Parameters

<i>field</i>	- field from LTE header to retrieve
<i>header</i>	- LTE header to retrieve field from

Returns

field from the LTE header

Implements [ProtocolPP::jprotocol](#).

7.67.2.5 `jarray<uint8_t> ProtocolPP::jlte::get_hdr() [virtual]`

Retrieves the complete LTE header

Returns

LTE header

Implements [ProtocolPP::jprotocol](#).

7.67.2.6 `void ProtocolPP::jlte::set_field(field_t field, uint64_t value) [virtual]`

Allows the user to update a field in the header

Parameters

<i>field</i>	- field to update in the LTE header
<i>value</i>	- new value to add to the LTE header

Implements [ProtocolPP::jprotocol](#).

7.67.2.7 `void ProtocolPP::jlte::set_hdr(jarray< uint8_t > & hdr) [virtual]`

Allows the user to update complete LTE header

Parameters

<i>hdr</i>	- new header to add to the LTE header
------------	---------------------------------------

Implements [ProtocolPP::jprotocol](#).

7.67.2.8 `void ProtocolPP::jlte::to_xml(tinyxml2::XMLPrinter & myxml, direction_t direction) [virtual]`

Writes the XML protocol and security objects to XML

Parameters

<i>myxml</i>	- XMLPrinter object
<i>direction</i>	- randomization

Implements [ProtocolPP::jprotocol](#).

The documentation for this class was generated from the following file:

- [include/jlte.h](#)

7.68 jltesa Class Reference

```
#include "include/jltesa.h"
```

7.68.1 Detailed Description

7.68.2 Long Term Evolution (LTE) Security Association

Figure 3.11. LTE Protocol Stack

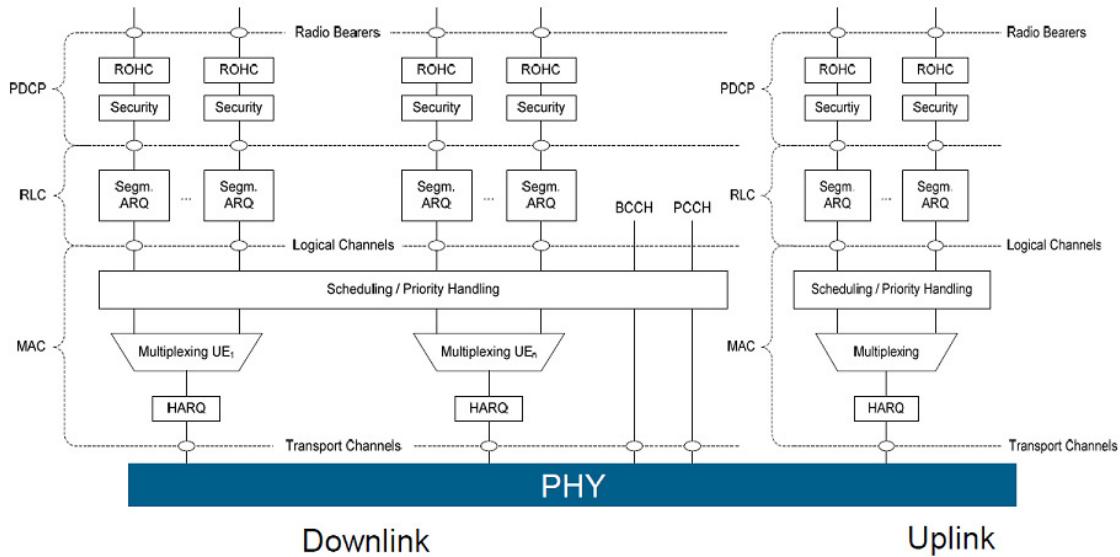


Figure 7.78: LTE Protocol Stack at the Data-Link Layer [1]

See PDCP_LTE_v14_1_0 specification on www.3gpp.org

If not otherwise mentioned in the definition of each field then the bits in the parameters shall be interpreted as follows: the left most bit string is the first and most significant and the right most bit is the last and least significant bit

Unless otherwise mentioned, integers are encoded in standard binary encoding for unsigned integers. In all cases the bits appear ordered from MSB to LSB when read in the PDU

7.68.2.1 PDCP SN

Length: 5, 7, 12, 15, 16, or 18 bits as indicated in the following table except for NB-IoT which uses 7 bit PDCP SN for DRB.

7.68.2.2 Data

Length: Variable

The Data field may include either one of the following:

- Uncompressed PDCP SDU (user plane data, or control plane data); or
- Compressed PDCP SDU (user plane data only)

7.68.2.3 MAC-I

Length: 32 bits

The MAC-I field carries a message authentication code calculated as specified in subclause 5.7 For control plane data that are not integrity protected, the MAC-I field is still present and should be padded with padding bits set to 0

	Description
Length: 32 bits	Control PDU
	Data PDU

For ciphering and integrity a COUNT value is maintained. The COUNT value is composed of a HFN and the PDCP SN. The length of the PDCP SN is configured by upper layers

The size of the HFN part in bits is equal to 32 minus the length of the PDCP SN.

NOTE: When performing comparison of values related to COUNT, the UE takes into account that COUNT is a 32-bit value, which may wrap around (e.g., COUNT value of 232 - 1 is less than COUNT value of 0).

7.68.2.5 R

Length: 1 bit

Reserved. In this version of the specification reserved bits shall be set to 0. Reserved bits shall be ignored by the receiver

7.68.2.6 D/C

Length: 1 bit

7.68.2.7 PDU type

Length: 3 bits

7.68.2.8 FMS

Length: 12 bits when a 12 bit SN length is used, 15 bits when a 15 bit SN length is used, and 18 bits when an 18 bit SN length is used

PDCP SN of the first missing PDCP SDU

7.68.2.9 Bitmap

Length: Variable

The length of the bitmap field can be 0

The MSB of the first octet of the type "Bitmap" indicates whether or not the PDCP SDU with the SN (FMS + 1) modulo (Maximum_PDCP_SN + 1) has been received and, optionally decompressed correctly. The LSB of the first octet of the type "Bitmap" indicates whether or not the PDCP SDU with the SN (FMS + 8) modulo (Maximum_PDCP_SN + 1) has been received and, optionally decompressed correctly

For API Documentation:

See Also

[ProtocolPP::jsecass](#)
[ProtocolPP::jprotocol](#)
[ProtocolPP::jsnow3g](#)
[ProtocolPP::jzuc](#)
[ProtocolPP::jmodes](#)

For Additional Documentation:

See Also

[jsecass](#)
[jprotocol](#)
[jsnow3g](#)
[jzuc](#)
[jmodes](#)

Protocol++ modified by : John Peter Greninger © John Peter Greninger 2015-2019 All Rights Reserved

All copyrights and trademarks are the property of their respective owners

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer
- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution
- Any and all modifications must be returned to John Peter Greninger at GitHub.com <https://github.com/jpgreninger/protocolpp> for evaluation. Inclusion of modifications in the source code shall be determined solely by John Peter Greninger. Failure to provide modifications shall render this license NULL and VOID and revoke any rights to use of Protocol++
- Commercial use requires a fee-based license obtainable at www.protocolpp.com
- Academic use requires written and notarized permission from John Peter and Sheila Rocha Greninger
- US Copyrights at <https://www.copyright.gov/>
 - TXu002059872 (Version 1.0.0)
 - TXu002066632 (Version 1.2.7)
 - TXu002082674 (Version 1.4.0)
 - TXu002097880 (Version 2.0.0)

The names of its contributors may not be used to endorse or promote products derived from this software without specific prior written permission

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE

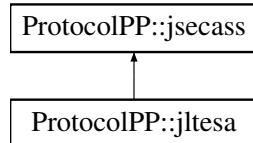
The documentation for this class was generated from the following file:

- [include/jltesa.h](#)

7.69 ProtocolPP::jltesa Class Reference

```
#include <jltesa.h>
```

Inheritance diagram for ProtocolPP::jltesa:



Public Member Functions

- `jltesa ()`
- `jltesa (direction_t dir, protocol_t type, cipher_t cipher, auth_t auth, int snlen, bool datctrl, bool poll, bool ext, bool rsn, uint8_t pdutype, uint8_t hdrext, uint8_t pgkindex, uint8_t sdutype, uint8_t bearer, uint16_t lenind, uint16_t ptkident, uint16_t kdid, uint32_t seqnum, uint32_t hfni, uint32_t fms, uint32_t nmp, uint32_t hrw, uint32_t fresh, unsigned int ckeylen, unsigned int akeylen, unsigned int bitmapplen, jarray< uint8_t > sufi, jarray< uint8_t > bitmap, std::shared_ptr< jarray< uint8_t >> cipherkey, std::shared_ptr< jarray< uint8_t >> authkey)`
- `jltesa (jltesa &rhs)`
Standard copy constructor.
- `jltesa (std::shared_ptr< jltesa > &rhs)`
Standard copy constructor from shared pointer.
- `virtual ~jltesa ()`
Standard deconstructor.
- template<typename T>
 `void set_field (field_t field, T value)`
- template<typename T>
 `T get_field (field_t field)`
- `void to_xml (tinyxml2::XMLPrinter &myxml, direction_t direction)`

7.69.1 Constructor & Destructor Documentation

7.69.1.1 ProtocolPP::jltesa::jltesa ()

Standard Constructor with defaults

```

*   jltesa rcv;
*
*   rcv.set_field<cipher_t>(field_t::CIPHER,
*     cipher_t::ZUCE);
*   rcv.set_field<uint8_t>(field_t::BEARER, 0x1E);
*   rcv.set_field<uint32_t>(field_t::HFNI, 0x000ABCDE);
*   rcv.set_field<uint32_t>(field_t::FRESH, 0x12345678);
*   rcv.set_field<uint32_t>(field_t::CKEYLEN, 16);
*   rcv.set_field<uint32_t>(field_t::AKEYLEN, 16);
*   rcv.set_field<jarray<uint8_t>>(field_t::CIPHERKEY, m_rand->getbyte(16))
*   ;
*   rcv.set_field<jarray<uint8_t>>(field_t::AIPHERKEY, m_rand->getbyte(16));
*

```

7.69.1.2 ProtocolPP::jltesa::jltesa (direction_t dir, protocol_t type, cipher_t cipher, auth_t auth, int snlen, bool datctrl, bool poll, bool ext, bool rsn, uint8_t pdutype, uint8_t hdrext, uint8_t pgkindex, uint8_t sdutype, uint8_t bearer, uint16_t lenind, uint16_t ptkident, uint16_t kdid, uint32_t seqnum, uint32_t hfni, uint32_t fms, uint32_t nmp, uint32_t hrw, uint32_t fresh, unsigned int ckeylen, unsigned int akeylen, unsigned int bitmapplen, jarray< uint8_t > sufi, jarray< uint8_t > bitmap, std::shared_ptr< jarray< uint8_t >> cipherkey, std::shared_ptr< jarray< uint8_t >> authkey)

LTE security association

Parameters

<i>dir</i>	- direction either UPLINK or DOWNLINK
<i>type</i>	- Type of packet (RLC or LTE)
<i>cipher</i>	- Encryption cipher (SNOWE, ZUCE, AES_CCM)
<i>auth</i>	- Authentication method (SNOW3G, ZUCA, AES_CCM)
<i>snlen</i>	- Sequence number length in bits (7 or 12 for RLC, 5, 7, 12, 15, 16, or 18 bits for LTE)
<i>datctrl</i>	- Data or Control flag bit (1-bit)
<i>poll</i>	- Polling flag. Used to request a status report from the receiver (1-bit)
<i>ext</i>	- Extension bit (1-bit)
<i>rsn</i>	- Reset sequence number (RSN)
<i>pductype</i>	- Type of Control PDU (STATUS, RESET, RESET_ACK, RSVD, 3-bits)
<i>hdrext</i>	- Header extension flag (2-bits)
<i>pgkindex</i>	- Five LSB(s) of PGK identity
<i>lenind</i>	- Length indicator (either 7 or 15 bits)
<i>sdtotype</i>	- PDCP SDU type i.e., Layer-3 Protocol Data Unit (IP, ARP, PC5SIG, NONIP, RSVD)
<i>bearer</i>	- BEARER (F8) value for SNOW3G and ZUC
<i>lenind</i>	- Length indicator
<i>ptkident</i>	- PTK identity
<i>kdid</i>	- Kd identity
<i>seqnum</i>	- Sequence Number (5, 7, 12, 15, 16, or 18 bits)
<i>hfni</i>	- Hyper Frame Number Indicator (HFNI, 20-bits)
<i>fms</i>	- PDCP SN of the First Missing PDCP SDU (FMS)
<i>nmp</i>	- Number of missing PDCP SDU(s) with associated COUNT value
<i>hrw</i>	- PDCP SN of the PDCP SDU received on WLAN with highest associated PDCP COUNT value
<i>fresh</i>	- FRESH (F9) value for SNOW3G and ZUC
<i>ckeylen</i>	- Length of the encryption key
<i>akeylen</i>	- Length of the authentication key
<i>bitmaplen</i>	- Length of the packet bit map
<i>sufi</i>	- Super Field (variable number of bits)
<i>bitmap</i>	- Sounds alot like a replay window
<i>cipherkey</i>	- Encryption key
<i>authkey</i>	- Authentication key

7.69.1.3 ProtocolPP::jltesa::jltesa (*jltesa* & *rhs*)

Standard copy constructor.

7.69.1.4 ProtocolPP::jltesa::jltesa (std::shared_ptr<*jltesa*> & *rhs*)

Standard copy constructor from shared pointer.

7.69.1.5 virtual ProtocolPP::jltesa::~jltesa () [inline], [virtual]

Standard deconstructor.

7.69.2 Member Function Documentation

7.69.2.1 template<typename T > T ProtocolPP::jltesa::get_field (*field_t field*)

Returns the version field of the LTE security association

	field name	Example
	DIRECTION <i>field</i> - field to retrieve from	set_field<ProtocolPP::direction_t>(ProtocolPP::field_t::DIRECTION, ProtocolPP::UPLINK)
Returns	TYPE version field of the LTE security association	set_field<ProtocolPP::protocol_t>(ProtocolPP::field_t::TYPE, ProtocolPP::LTE)
7.69.2.2 template<typename T > void ProtocolPP::jtesa::set_field(field_t field, T value)	CIPHER	set_field<ProtocolPP::cipher_t>(ProtocolPP::field_t::CIPHER, ProtocolPP::ZUCE)
Allows the user to update the field of the LTE security association	AUTH	set_field<ProtocolPP::auth_t>(ProtocolPP::field_t::AUTH, ProtocolPP::SNOWA)
Parameters	SNLEN <i>field</i> - field to update the LTE security association <i>value</i> - value to update the LTE security association	set_field<int>(ProtocolPP::field_t::SNLEN, 7)
7.69.2.3 void ProtocolPP::jtesa::to_xml(tinyxml2::XMLPrinter & myxml, direction_t direction) [virtual]	DATCTRL	set_field<bool>(ProtocolPP::field_t::DATCTRL, false)
Prints the protocol object in XML	POLLBIT	set_field<bool>(ProtocolPP::field_t::POLLBIT, false)
Parameters	EXTENSION <i>myxml</i> - XMLPrinter object to print to <i>direction</i> - randomzation	set_field<bool>(ProtocolPP::field_t::EXTENSION, false)
RN Implements ProtocolPP::jsecass .	PDUTYPE • include/jtesa.h	set_field<bool>(ProtocolPP::field_t::RSN, true) set_field<uint8_t>(ProtocolPP::field_t::PDUTYPE, 0x32)
7.70 jmacsec Class Reference	HDREXT	set_field<uint8_t>(ProtocolPP::field_t::HDREXT, 0xFC)
PGKINDEX #include "include/jmacsec.h"	SDUTYPE 7.70.1 Detailed Description	set_field<uint8_t>(ProtocolPP::field_t::PGKINDEX, 0x03)
BEARER Information provided by IEEE https://standards.ieee.org/getieee802/download/802.1AE-2006.pdf	MACsec protocol 7.70.2 MACsec protocol data units	set_field<uint8_t>(ProtocolPP::field_t::SDUTYPE, 0x15) set_field<uint8_t>(ProtocolPP::field_t::BEARER, 0x54) set_field<uint16_t>(ProtocolPP::field_t::LENGTHIND, 0x0101)
PTKINDENT This clause specifies the structure and encoding of the MACsec Protocol Data Units (MPDUs) exchanged between MAC Security Entities (SecYs). It	Encoding of MACsec protocol data units • KID • SEQNUM • HFNI • FMS	set_field<uint16_t>(ProtocolPP::field_t::PTKINDENT, 0x0142) set_field<uint16_t>(ProtocolPP::field_t::KID, 0x3542) set_field<uint32_t>(ProtocolPP::field_t::SEQNUM, 0x00000001) set_field<uint32_t>(ProtocolPP::field_t::HFNI, 0xFFFFFFFF80) set_field<uint32_t>(ProtocolPP::field_t::FMS, 0x89432178)
NMP		set_field<uint32_t>(ProtocolPP::field_t::NMP, 0x00000000)

NOTE—The MPDU validation checks specified in this clause are deliberately limited to ensuring successful decoding, and do not overlap with the specification of SecY operation (Clause 10).

Structure, representation, and encoding

All MPDUs shall contain an integral number of octets. The octets in a MPDU are numbered starting from 1 and increasing in the order they are put into the MAC Service Data Unit (MSDU) that accompanies a request to or indication from the instance of the MAC Internal Sublayer Service (ISS) used by a SecY. The bits in an octet are numbered from 1 to 8 in order of increasing bit significance, where 1 is the least significant bit in the octet. Where octets and bits within a MPDU are represented using a diagram, octets shown higher on the page than subsequent octets and octets shown to the left of subsequent octets at the same height on the page are lower numbered, bits shown to the left of other bits within the same octet are higher numbered. Where two or more consecutive octets are represented as hexadecimal values, lower numbered octet(s) are shown to the left and each octet following the first is preceded by a hyphen, e.g., 01-80-C2-00-00-00.

When consecutive octets are used to encode a binary number, the lower octet number has the more significant value. When consecutive bits within an octet are used to encode a binary number, the higher bit number has the most significant value. When bits within consecutive octets are used to encode a binary number, the lower octet number composes the more significant bits of the number. A flag is encoded as a single bit, and is set (True) if the bit takes the value 1, and clear (False) otherwise. The remaining bits within the octet can be used to encode other protocol fields.

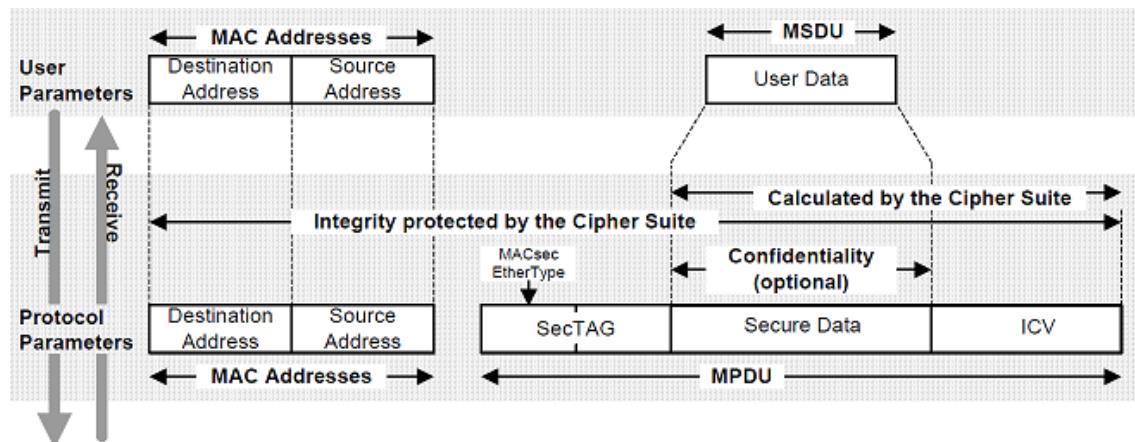


Figure 8-1—MACsec

Figure 7.79: Security Encapsulation of Ethernet Packets (MacSec)

Major components

Each MPDU comprises

- A Security TAG (SecTAG)
- Secure Data (9.10) c) An Integrity Check Value (ICV)

Each of these components comprises an integral number of octets and is encoded in successive octets of the MPDU as illustrated in Figure 9-1

NOTE—The MPDU does not include the source and destination MAC addresses, as these are separate parameters of the service requests and indications to and from the insecure service that supports MACsec

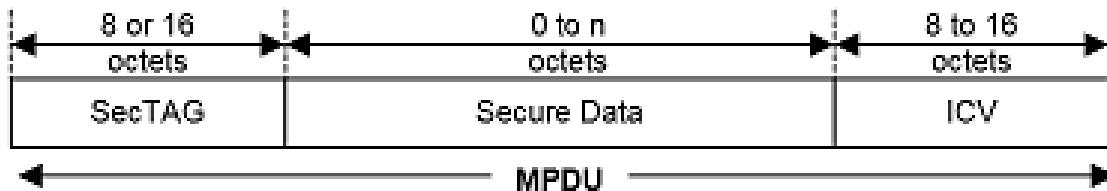


Figure 9-1—MPDU components

Figure 7.80: Packet Data Unit (PDU) for MacSec

Security TAG

The Security TAG (SecTAG) is identified by the MACsec EtherType, and conveys the

- TAG Control Information (TCI, 9.5)
- Association Number (AN, 9.6)
- Short Length (SL, 9.7)
- Packet Number (PN, 9.8)
- Optionally encoded Secure Channel Identifier (SCI, 9.9).

The format of the SecTAG is illustrated in Figure 9-2

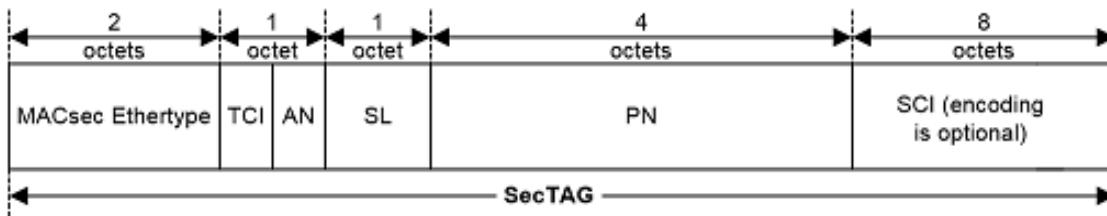


Figure 9-2—SecTAG format

Figure 7.81: Security Tag (SecTag) for MacSec packets

MACsec EtherType

The MACsec EtherType comprises octet 1 and octet 2 of the SecTAG. It is included to allow

- Coexistence of MACsec capable systems in the same environment as other systems
- Incremental deployment of MACsec capable systems
- Peer SecYs to communicate using the same media as other communicating entities
- Concurrent operation of Key Agreement protocols that are independent of the MACsec protocol and the Current Cipher Suite
- Operation of other protocols and entities that make use of the service provided by the SecY's Uncontrolled Port to communicate independently of the Key Agreement state

Table 9-1—MACsec EtherType allocation

Tag Type	Name	Value
IEEE 802.1AE Security TAG	MACsec EtherType	88-E5

Figure 7.82: Ethernet Packet Type

The encoding of the MACsec EtherType in the MPDU is illustrated in the next figure

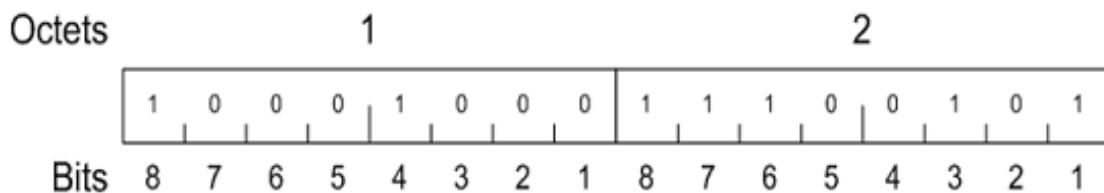
**Figure 9-3—MACsec EtherType encoding**

Figure 7.83: EtherType Field Within the Macsec Packet

TAG Control Information (TCI)

The TCI field comprises bits 8 through 3 of octet 3 (Figure 9-4) of the SecTAG. These bits facilitate

- Version numbering of the MACsec protocol without changing the MACsec EtherType
- Optional use of the MAC Source Address parameter to convey the SCI
- Optional inclusion of an explicitly encoded SCI (7.1.2, Figure 7-7)
- Use of the EPON (Clause 12) Single Copy Broadcast capability, without requiring an explicit SCI to distinguish the SCB Secure Channel
- Extraction of the User Data from MPDUs by systems that do not possess the SAK (8.1.2, 8.1.4) when confidentiality is not being provided
- Determination of whether confidentiality or integrity alone are in use

The encoding of the MACsec TCI in the MPDU is illustrated in Figure 9-4.

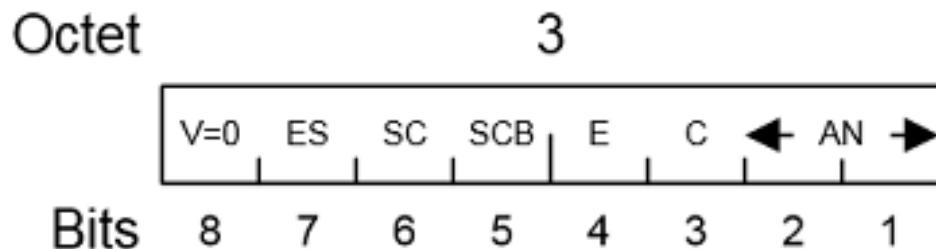


Figure 9-4—MACsec TCI and AN Encoding

Figure 7.84: TAG Control Information (TCI) of the Macsec Packet

TCIAN bit encoding is as follows:

- The version number shall be 0 and is encoded in bit 8. NOTE—Future versions of the MACsec protocol may use additional bits of the TCI to encode the version number. The fields and format of the remainder of the MPDU may change if the version number changes
- If the MPDU is transmitted by an end station and the first 6 octets of the SCI are equal to the value of the octets of MAC Source Address parameter of the ISS request in canonical format order, bit 7 [the End Station (ES) bit] of the TCI may be set
- If the ES bit is set, bit 6 (the SC bit) shall not be set and an SCI shall not be explicitly encoded in the SecTAG. The ES bit is clear if the Source Address is not used to determine the SCI. If an SCI (9.9, 7.1.2) is explicitly encoded in the SecTAG, bit 6 (the SC bit) of the TCI shall be set. The SC bit shall be clear if an SCI is not present in the SecTAG
- If and only if the MPDU is associated with the Secure Channel that supports the EPON Single Copy Broadcast capability, bit 5 (the SCB bit) of the TCI may be set
- If the SCB is set, bit 6 (the SC bit) shall not be set and an SCI shall not be explicitly included in the SecTAG. If the ES bit is set and the SCB is not set, the SCI comprises a Port Identifier (7.1.2) component of 00-01. If the SCB bit is set, the Port Identifier (7.1.2) component has the reserved SCB value of 00-00
- If the Encryption (E) bit is set and the Changed Text (C) bit is clear, the frame is not processed by the SecY (10.6) but is reserved for use by the KaY. Otherwise, the E bit is set if and only if confidentiality is being provided and is clear if integrity only is being provided
- The C bit is clear if and only if the Secure Data is exactly the same as the User Data and the ICV is 16 octets long. When the Default Cipher Suite (14.5) is used for integrity protection only, the Secure Data is the unmodified User Data, and a 16 octet ICV is used. Both the E bit and the C bit are therefore clear, and the data conveyed by MACsec is available to applications, such as network management, that need to see the data but are not trusted with the SAK that would permit its modification. Other Cipher Suites may also integrity protect data without modifying it, and use a 16 octet ICV, enabling read access to the data by other applications. The E and C bits are also clear for such Cipher Suites when integrity only is provided
- Some cryptographic algorithms modify or add to the data even when integrity only is being provided, or use an ICV that is not 16 octets long. The C bit is never clear for such an algorithm, even if the E bit is clear to indicate that confidentiality is not provided. Recovery of the data from a MACsec frame with the E bit clear and the C bit set requires knowledge of the Cipher Suite at a minimum. That information is not provided in the MACsec frame. If both the C bit and E bit are set, confidentiality of the original User Data is being provided.

Association Number (AN)

The AN is encoded as an integer in bits 1 and 2 of octet 3 of the SecTAG (Figure 9-4) and identifies up to four different SAs within the context of an SC. NOTE—Although each receiving SecY only needs to maintain two SAs per SC, the use of a 2-bit AN simplifies the design of protocols that update values associated with each of the SAs.

Short Length (SL)

SL is an integer encoded in bits 1 through 6 of octet 4 of the SecTAG and is set to the number of octets in the Secure Data (9.10) field, i.e., the number of octets between the last octet of the SecTAG and the first octet of the ICV, if that number is less than 48. Otherwise, SL is set to zero. If the number is zero then the frame is deemed not to have been short. The Secure Data field always comprises at least one octet. Bits 7 and 8 of octet 4 shall be zero.

Packet Number (PN)

The PN is encoded in octets 5 through 8 of the SecTAG to

- Provide a unique IV PDU for all MPDUs transmitted using the same SA
- Support replay protection

NOTE—As specified in this clause, the IV used by the default Cipher Suite (GCM-AES-128) comprises the SCI (even if the SCI is not transmitted in the SecTAG) and the PN. Subject to proper unique MAC Address allocation procedures, the SCI is a globally unique identifier for a SecY. To satisfy the IV uniqueness requirements of CTR mode of operation, a fresh key is used before PN values are reused.

Secure Channel Identifier (SCI)

If the SC bit in the TCI is set, the SCI (7.1.2, 8.2.1) is encoded in octets 9 through 16 of the SecTAG, and facilitates

- Identification of the SA where the CA comprises three or more peers
- Network management identification of the SecY that has transmitted the frame Octets 9 through 14 of the SecTAG encode the System Identifier component of the SCI. This comprises the six octets of a globally unique MAC address uniquely associated with the transmitting SecY. The octet values and their sequence conform to the Canonical Format specified by IEEE Std 802. Octets 15 and 16 of the SecTAG encode the Port Identifier component of the SCI, as an integer.

NOTE —The 64-bit value FF-FF-FF-FF-FF-FF is never used as an SCI and is reserved for use by implementations to indicate the absence of an SC or an SCI in contexts where an SC can be present.

An explicitly encoded SCI field in the SecTAG is not required on point-to-point links, which are identified by the operPointToPointMAC status parameter of the service provider. In the point-to-point case, the secure association created by the SecY for the peer SecYs, together with the direction of transmission of the secured MPDU, can be used to identify the transmitting SecY and therefore an explicitly encoded SCI is unnecessary. Although the SCI does not have to be repeated in each frame when only two SecYs participate in a CA (see Clause 8, Clause9, and Clause10), the SCI still forms part of the cryptographic computation.

Secure Data

The Secure Data comprises all the octets that follow the MACsec TAG and precede the ICV. The Secure Data field is never of zero length, since the primitives of the MAC Service require a non-null MSDU (User Data) parameter.

NOTE 1—In practice, if the MSDU composed by the operation of the current Cipher Suite following MPDU reception contains less than two octets, it will be discarded by the user of the SecY's controlled port, since it is too short to contain an EtherType or an LLC length field. Such discard is, however, determined by the user of the Controlled Port and not by the SecY itself.

NOTE 2—Ethernet transports frames of a minimum size, and provides no explicit indication of PDU length if the PDU is composed of fewer octets. The SL field allows the originator of the frame, which is not necessarily aware of the need of an intervening Ethernet component to pad the frame, to specify the number of octets in the MPDU, thus allowing the receiver to unambiguously locate the ICV.

Integrity Check Value (ICV)

The length of the ICV is Cipher Suite dependent, but is not less than 8 octets and not more than 16 octets, depending on the Cipher Suite.

NOTE—The ICV protects the destination and source MAC address parameters, as well as all the fields of the MPDU.

PDU validation

A received MPDU is valid if and only if it comprises a valid SecTAG, one or more octets of Secure Data, and an ICV, i.e.,

- It comprises at least 17 octets
- Octets 1 and 2 compose the MACsec EtherType
- The V bit in the TCI is clear
- If the ES or the SCB bit in the TCI is set, then the SC bit is clear
- Bits 7 and 8 of octet 4 of the SecTAG are clear
- If the C and SC bits in the TCI are clear, the MPDU comprises 24 octets plus the number of octets indicated by the SL field if that is non-zero and at least 72 octets otherwise
- If the C bit is clear and the SC bit set, then the MPDU comprises 32 octets plus the number of octets indicated by the SL field if that is non-zero and at least 80 octets otherwise
- If the C bit is set and the SC bit clear, then the MPDU comprises 8 octets plus the minimum length of the ICV as determined by the Cipher Suite in use at the receiving SecY, plus the number of octets indicated by the SL field if that is non-zero and at least 48 additional octets otherwise
- If the C and SC bits are both set, the frame comprises at least 16 octets plus the minimum length of the ICV as determined by the Cipher Suite in use at the receiving SecY, plus the number of octets indicated by the SL field if that is non-zero and at least 48 additional octets otherwise

7.7.0.3 Packet Processing

Secure frame generation

For each transmit request at the Controlled Port, the Secure Frame Generation process

- a. Assigns the frame to an SA (10.5.1)
- b. Assigns the nextPN variable for that SA to be used as the value of the PN in the SecTAG (10.5.2)
- c. Encodes the octets of the SecTAG (10.5.3)
- d. Provides the protection function (14.1, 10.5.4) of the Current Cipher Suite with
 1. The SA Key (SAK)
 2. The SCI for the SC used by the SecY to transmit
 3. The PN
 4. The SecTAG
 5. The sequence of octets that compose the User Data
- e. Receives the following parameters from the Cipher Suite protection operation
 1. The sequence of octets that compose the Secure Data
 2. The ICV

f. Issues a request to the Transmit Multiplexer with the destination and source MAC addresses, and priority of the frame as received from the Controlled Port, and an MPDU comprising the octets of the SecTAG, Secure Data, and the ICV concatenated in that order (10.5.5)

If the management control protectFrames is False, the preceding steps are omitted, an identical transmit request is made to the Transmit Multiplexer, and the OutPktsUntagged counter incremented

NOTE—This model of operation supports the externally observable behavior that can result when the Cipher Suite implementation calculates the Secure Data and ICV parameters for a number of frames in parallel, and the responses to protection and validation requests are delayed. Transmitted frames are not misordered

Transmit SA assignment

Each SA is identified by its Association Number (AN). Each frame is assigned to the SA identified by the current value of the encodingSA variable. This is updated following an LMI request from the KaY to start transmitting using the SA and can be read but not written by network management. Frames will be protected using the encodingSA immediately after the last frame assigned to the previous SA has been protected.

If the SA identified by the encodingSA is not available for use, and the management control protectFrames is set, MAC_Operational transitions to False for the Controlled Port, and frames are neither accepted or delivered using the port

Transmit PN assignment

The frame's PN is set to the value of nextPN for the SA, and nextPN is incremented. If the nextPN variable for the encodingSA is zero (or 2 32) and the protectFrames control is set, MAC_Operational transitions to False for the Controlled Port and frames are neither accepted or delivered. The initial value of nextPN is set by the KaY via the LMI prior to use of the SA, and its current value can be read both while and after the SA is used to transmit frames. The value of nextPN can be read, but not written, by network management.

SecTAG encoding

The SecTAG is encoded as specified in Clause 9

- If the management control useES is True and alwaysIncludeSCI is False, the ES bit in the SecTAG shall be set. Otherwise, if useES is False or alwaysIncludeSCI is True, the ES bit shall be clear.
- If the management control alwaysIncludeSCI is set, or the number of receive SCs with SAs enabled for reception is greater than one and both useES and useSCB are False, the SC bit in the SecTAG shall be set and the SCI explicitly encoded in the SecTAG; otherwise, the SC bit shall be clear and the SCI not explicitly encoded
- If the management control useSCB is True and alwaysIncludeSCI is False, the SCB bit in the SecTAG shall be set. Otherwise, if useSCB is False or alwaysIncludeSCI is True, the SCB bit shall be clear
- The values of useES, useSCB, and alwaysIncludeSCI can be written and read by management. The number of active receive SCs is controlled by the KaY but can be read by management
- If a frame is to be integrity protected, but not encrypted, with the number and value of the octets of the Secure Data exactly the same as those of the User Data, and an ICV of 16 octets, then the E bit shall be clear and the C bit clear. The E bit shall be clear and the C bit set if the frame is not encrypted but the octets of the Secure Data differ from those of the User Data or the ICV is not 16 octets
- If both confidentiality (through encryption) and integrity protection are applied to a frame then both the E bit and the C bit shall be set. The SecY shall not encode a SecTAG that has both the E bit set and the C bit clear for any frame received from the Controlled Port for transmission
- If the alwaysIncludeSCI control is set or the number of receive SCs with SAs enabled for reception is greater than 1, the SCI is included in the SecTAG; otherwise, it is omitted. The value of alwaysIncludeSCI can be written and read by management. The number of active receive SCs is controlled by the KaY, but can be read by management

Cryptographic protection

If the Cipher Suite is currently protecting frames using the previous SA and its SA Key, as reflected by the value of the encipheringSA, the frame can be queued awaiting protection. The value of the encipheringSA is updated, and

protection of the frame parameters is started within a minimum frame size transmission delay, after the last frame has been protected using the previous key

The use of each of the Cipher Suites specified by this standard is specified in Clause 14, which takes precedence over any explanation in this or other clauses

The appropriate octet counter is incremented by the number of octets in the User Data (OutOctetsEncrypted if confidentiality protection was provided, and OutOctetsProtected otherwise)

Transmit request

If the MPDU composed of the concatenated octets of the SecTAG, Secure Data, and ICV exceeds the size of the MSDU supported by the Common Port, the frame is discarded and a counter incremented. Details of the discarded frame may be recorded to assist network management resolution of the problem. Otherwise, the parameters of the service request are submitted to the Transmit Multiplexer

Receive Secure frame verification

For each receive indication from the Receive Demultiplexer, the Secure Frame Verification process

- a. Examines the user data for a SecTAG
- b. Validates frames with a SecTAG as specified in 9.12
- c. Extracts and decodes the SecTAG as specified in 9.3 through 9.9
- d. Extracts the User Data and ICV as specified in 9.10 and 9.11
- e. Assigns the frame to an SA (10.6.1)
- f. Performs a preliminary replay check against the last validated PN for the SA (10.6.2)
- g. Provides the validation function (14.1, 10.6.3) of the Current Cipher Suite with
 1. The SA Key (SAK)
 2. The SCI for the SC used by the SecY to transmit
 3. The PN
 4. The SecTAG
 5. The sequence of octets that compose the Secure Data
 6. The ICV
- h. Receives the following parameters from the Cipher Suite validation operation
 1. A Valid indication, if the integrity check was valid and the User Data could be recovered
 2. The sequence of octets that compose the User Data
- i. Updates the replay check (10.6.4)
- j. Issues an indication to the Controlled Port with the DA, SA, and priority of the frame as received from the Receive Demultiplexer, and the User Data provided by the validation operation (10.6.5).

If the management control validateFrames is not Strict, frames without a SecTAG are received, counted, and delivered to the Controlled Port; otherwise, they are counted and discarded. If validateFrames is Disabled, cryptographic validation is not applied to tagged frames, but frames whose original service user data can be recovered are delivered. Frames with a SecTAG that has the TCI E bit set but the C bit clear are discarded, as this reserved encoding is used to identify frames with a SecTAG that are not to be delivered to the Controlled Port. Figure 10-5 summarizes the operation of management controls and counters

Receive SA assignment

An SCI is associated with the received frame, and used to locate the receive SC. If an SCI is not explicitly encoded in the SecTAG, the default value established by the KaY for a single peer is used. If the SC is not found, it may be recorded to assist network management resolution of the problem, and:

a. If validateFrames is Strict or the C bit in the SecTAG is set, the InPktsNoSCI counter is incremented and the frame is discarded; otherwise

b. The InPktsUnknownSCI counter is incremented and the frame (with the SecTAG and ICV removed) is delivered to the Controlled Port

If the receive SC has been identified, the frame's AN is used to locate the receive SA received frame and processing continues with the preliminary replay check. If the SA is not in use:

c. If validateFrames is Strict or the C bit is set, the frame is discarded and the InPktsNotUsingSA counter incremented; otherwise

d. The InPktsUnusedSA counter is incremented and the frame delivered to the Controlled Port

NOTE—The short phrase “the frame is discarded” is commonly used to express the more formal notion of not processing a service primitive (an indication or request) further and recovering the resources that embody the parameters of that service primitive. No further processing is applied. However, if a duplicate of the primitive has been submitted to another process, by the Receive Demultiplexer in this case, processing of that duplicate is unaffected

Preliminary replay check

If replayProtect control is enabled and the PN of the received frame is less than the lowest acceptable packet number (see 10.6.5) for the SA, the frame is discarded and the InPktsLate counter incremented.

NOTE—If the SC is supported by a network that includes buffering with priority queueing, such as a provider bridged network, delivered frames can be reordered.

Cryptographic validation

The frame can be queued awaiting validation. If the frame reception rate exceeds the Cipher Suite's validation capabilities, the frame may be discarded and the InPktsOverrun counter incremented

If the validateFrames control is Disabled, the Cipher Suite validation is not used to validate the frame

If validateFrames is not Disabled, and the E bit in the SecTAG is set, the Cipher Suite is used to validate and decrypt the frame. If the Cipher Suite does not provide confidentiality protection, it shall not return VALID

The InOctetsDecrypted counter is incremented by the number of octets in the resulting User Data (or an estimate of that number, if VALID is not returned)

If validateFrames is not Disabled, and the E bit in the SecTAG is clear, the Cipher Suite is used to validate the frame. If the Cipher Suite does not provide integrity protection without confidentiality it shall not return VALID. The InOctetsValidated counter is incremented by the number of octets in the resulting User Data (or an estimate of that number, if VALID is not returned)

The frame is marked valid if the Cipher Suite is used and returns VALID, and is marked invalid otherwise. The use of each of the Cipher Suites specified by this standard is specified in Clause 14, which takes precedence over any explanation in this or other clauses

Replay check update

If the PN of the received frame is less than the lowest acceptable packet number for the SA, and replayProtect is enabled, the frame is discarded and the InPktsLate counter incremented.

NOTE—This model of operation assumes that any queuing within the verification process occurs prior to frame validation, and the check described uses the lowest acceptable PN updated by prior frames as described below (10.6.5). Implementations can process frames as convenient, provided the externally observable result is the same.

Receive indication

If the received frame is marked as invalid, and the validateFrames control is Strict or the C bit in the SecTAG was set, the frame is discarded and the InPktsNotValid counter incremented. Otherwise the frame is delivered to the Controlled Port, and the appropriate counter incremented as follows:

a. If the frame is not valid and validateFrames is set to Check, InPktsInvalid, otherwise

b. If the received PN is less than the lowest acceptable PN (treating a PN value of zero as 2^{32}), InPktsDelayed, otherwise

c. If the frame is not valid, InPktsUnchecked, otherwise

d. InPktsOK

If the PN for the frame was equal to or greater than the nextPN variable for the SA, nextPN is set to the value for the received frame, incremented by one. The lowest acceptable PN variable is set to the greater of its existing value and the value of nextPN minus the replayWindow variable.

NOTE—The lowest acceptable packet number can also be set or incremented by the KaY to ensure timely delivery.

7.70.4 Cipher Suites

A Cipher Suite is an interoperable specification of cryptographic algorithms together with the values of parameters (for example, key size) to be used by those algorithms. Specification of the cryptographic functions required by MAC Security in terms of Cipher Suites increases interoperability by providing a clear default and a limited number of alternatives.

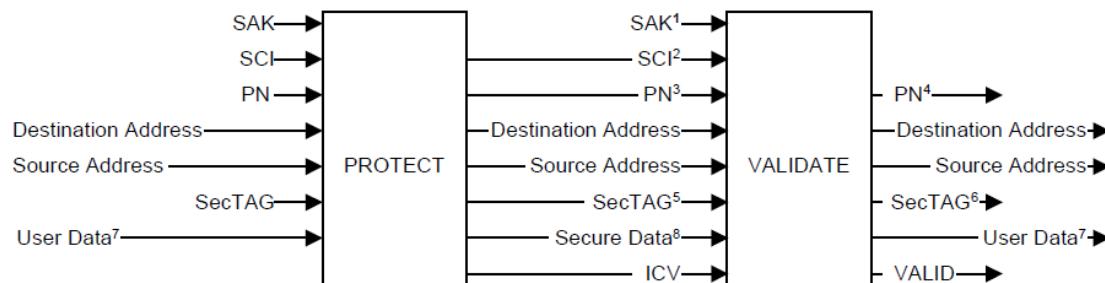
This clause specifies

- Terms that describe the use of each Cipher Suite by the MAC Security Entity (SecY)
- Capabilities required of each Cipher Suite
- Requirements this standard places on Cipher Suite specification
- Mandatory and optional Cipher Suites for use in conjunction with this standard
- Criteria for the use of additional Cipher Suites in conjunction with MAC Security for implementations for which a claim of conformance to this standard is made

NOTE —The choice and combination of cryptographic methods is notorious for the introduction of unexpected security exposures. Each Cipher Suite is an algorithm or combination of algorithms whose interactions have been studied by the professional security community

Cipher Suite use

A Cipher Suite is initialized with one or more Cipher Suite dependent keys, and then used to protect protocol parameters. Any implementation of the same Cipher Suite, initialized with the same key values, can be used to validate and recover the protected parameters. The protect and validate operations are illustrated in Figure 14-1, and their inputs and outputs specified below



¹ The SAK to be used on receipt of the frame is identified by the SCI and the AN.

² The SCI is extracted from the SCI field of the SecTAG if present. A value conveyed by key agreement (point-to-point only) is used otherwise.

³ The PN is conveyed in the SecTAG

⁴ The validated PN can be used for replay protection.

⁵ All the transmitted octets of the SecTAG are protected, including the optional SCI field if present

⁶ The validated received SecTAG contains bits of the TCI, and optionally the SCI, these can be used for service multiplexing (11.7).

⁷ The length, in octets, of the User Data is conveyed by the User Data parameter, and is protected by Cipher Suite operation.

⁸ The length, in octets, of the Secure Data is conveyed by the MACsec frame, unless it is short, when it is conveyed by the SL parameter in the SecTAG TCI

Figure 14-1—Cipher Suite Protect and Validate operations

Protect (SAK, SCI, PN, Destination Address, Source Address, SecTAG, User Data)	Validate (SAK, SCI, PN, Destination Address, Source Address, SecTAG, Secure Data, ICV)
	Secure Data, ICV		User Data, VALID

Figure 7.85: Supported MacSec Ciphersuites

The SAK (Secure Association Key, 3.36, 7.1) is the value of the Cipher Suite dependent key(s)

The SCI (Secure Channel Identifier, 3.36, 7.1.2) is a 64-bit identifier that is globally unique amongst all correctly configured Cipher Suite implementation instances protecting MACsec protocol parameters

The PN (Packet Number, 3.27, 8.3) is a 32-bit number that is never zero, is incremented each time a protect request is made for a given SCI, and is never repeated for an SCI unless the SAK is changed

The Destination Address and Source Address are the MAC addresses of the frame. MAC Addresses are specified as octet strings, using the canonical format specified in IEEE Std 802

The SecTAG is as specified in Clause 9

The ICV (Integrity Check Value, 3.13, 8.3) is a string of octets. VALID is a Boolean parameter. If TRUE the validation was successful

Given the SAK, SCI, PN, Source Address, Destination Address, SecTAG, and the User Data, the Protect operation returns the Secure Data and ICV.

Given the same SAK, SCI, PN, Source Address, Destination Address, and SecTAG, and the Secure Data and ICV, the Verify operation returns the original User Data and VALID. If any of the parameters were modified, VALID is returned False.

Cipher Suite capabilities

Any Cipher Suite used with MACsec shall

- Provide integrity protection for the SCI, PN, Source Address, Destination Address, SecTAG, and from 0 through $2^{16} - 1$ octets of User Data on each invocation
- Provide integrity and confidentiality (if specified) for up to $2^{32} - 1$ invocations, each with a different PN, without requiring a fresh SAK

- Given any specific number of octets of User Data, generate a predictable number of octets of Secure Data and ICV

and may

- Provide confidentiality protection for all the octets of the User Data
- Provide confidentiality protection for all the octets of the User Data following an initial number of octets, as specified in 10.7.24

and shall not

- Generate Secure Data that when added to the number of octets in the ICV contains over 896 octets more than the User Data. NOTE — A Cipher Suite may introduce additional fields into the Secure Data even if confidentiality is not provided
- Modify or constrain the values of the SCI, PN, Source Address, Destination Address, or SecTAG fields, other than as specified in this Clause (14)
- Require an SAK exceeding 1024 bits long (in total for all keys that compose the SAK)
- Require different keys for the protect and validate operations

An implementation of MACsec for which conformance to this standard is claimed includes at least one Cipher Suite that provides integrity without confidentiality, with the Secure Data the same as the User Data, and the ICV comprising 16 octets. This requirement is met by the mandatory Default Cipher Suite.

Cipher Suite specification

Each Cipher Suite specification shall comprise an interoperable specification of the protection and verification procedures in terms of the parameters specified in 14.1 and shall state

- Whether confidentiality of the User Data is provided
- The maximum difference in the lengths of the User Data and Secure Data
- The length of the ICV
- The length and properties of the keys required, including assumptions of the scope of uniqueness

NOTE—While this standard provides definitive specifications of the Cipher Suites that support full conformance, those specifications make the greatest possible use of other public and established standards, and are principally concerned with ensuring unambiguous application of those standards in the context of MAC Security.

Cipher Suite conformance

An implementation of MAC Security that claims full conformance to this standard shall implement the mandatory Cipher Suites in Table 14-1, may implement one or more of the Optional Cipher Suites in the table, and shall not implement any other Cipher Suite. Every conformant implementation shall include at least one Cipher Suite that does not encrypt User Data.

Table 14-1—MACsec Cipher Suites

Cipher Suite #	Cipher Suite Name	Services provided		Mandatory/Optional	Defining Clause
		Integrity without confidentiality	Integrity and confidentiality		
00-80-02-00-01-00-00-01	GCM-AES-128	Yes	Yes	Mandatory	14.5

Figure 7.86: Macsec Ciphersuite Conformance Requirements

NOTE —Currently, Table 14-1 does not include any optional Cipher Suites.

Table 14-1 assigns a Cipher Suite reference number for use in protocol identification within a MACsec context, provides a short name for use in this standard, indicates the type of cryptographic algorithm used and the security services provided, specifies whether the Cipher Suite is mandatory or optional for conformance to this standard, and references the clause of this standard that provides the definitive description of the Cipher Suite.

Conformance with Cipher Suite variance

An implementation of MAC Security that claims conformance to this standard with Cipher Suite variance, shall implement the mandatory Cipher Suites in Table 14-1, may implement one or more of the optional Cipher Suites in Table 14-1, and may implement alternate Cipher Suites that meet the requirements of 14.2 and 14.3, and the following guidelines, and shall not implement any other Cipher Suite, or other combination of cryptographic algorithms and parameters.

The use of additional Cipher Suites shall meet the following guidelines:

- Algorithms chosen have an effective key length of at least 128 bits. In schemes built on block ciphers, the underlying block cipher has a block width of at least 128 bits
- If serviced by separate algorithms, the properties of the authentication and confidentiality mechanisms are combinable in accordance with well-established security results. Either the encryption happens before authentication, or the encryption is performed through keystream generation
- Either of the following holds true:
 - The underlying cryptographic cipher is approved by either a national or international standards body or a government agency; or
 - The following conditions i) through iv) apply:
 - * The Cipher Suite provides message authentication using a message authentication algorithm with a publicly available proof of security against forgery attacks, even in a model where the attacker has the ability to choose messages for the sender
 - * If confidentiality is provided, the confidentiality mechanism has a publicly available proof of security in a model where the attacker has the ability to adaptively choose both plaintext and cipher text
 - * Mechanisms for confidentiality and message authentication are used in a way that is consistent with their proof of security. For example, if using the Cipher Block Chaining (AES-CBC) mode of operation the IV is performed through keystream generation
 - * Mechanisms for confidentiality and message authentication are used in a way that is consistent with their proof of security. For instance, if using the Cipher Block Chaining (AES-CBC) mode of operation, the IV is randomly selected with each message, and not sequentially

Default Cipher Suite (GCM-AES-128)

The Default Cipher Suite uses the Galois/Counter Mode of Operation with the AES-128 symmetric block cipher, as specified in this clause by reference to the terms K, IV, A, P, C, T used in section 2.1 of the GCM specification (GCM) as submitted to NIST.

K is the 128 bit SAK. The 64 most significant bits of the 96-bit IV are the octets of the SCI, encoded as a binary number (9.1). The 32 least significant bits of the 96-bit IV are the octets of the PN, encoded as a binary number (9.1). T is the ICV and is 128 bits long. When the bit-strings A, P, and C are specified in terms of octet strings, earlier octets compose earlier bits, and more significant bits in each octet are earlier.

NOTE—The bit strings obtained by transforming MAC Address and data octets using these rules do not correspond to IEEE 802.3 “wire order” for frame transmission.

When the Default Cipher Suite is used for Integrity Protection

- A is the Destination MAC Address, Source MAC Address, and the octets of the SecTAG and User Data concatenated in that order
- P is null
- The Secure Data is the octets of the User Data, without modification.

When the Default Cipher Suite is used for Confidentiality Protection without a confidentiality offset

- A is the Destination MAC Address, Source MAC Address, and the octets of the SecTAG concatenated in that order
- P is the octets of the User Data
- The Secure Data is C.

When the Default Cipher Suite is used for Confidentiality Protection with a confidentiality offset

- A is the Destination MAC Address, Source MAC Address, and the octets of the SecTAG and the first confidentialityOffset (10.7.24) octets of the User Data concatenated in that order
- P is the remaining octets of the User Data
- The Secure Data is the first confidentialityOffset octets of the User Data concatenated with C, in that order.

7.70.4.1 Media Access Control (MAC) Security—Amendment 2: Extended Packet Numbering

Amendment 2: Extended Packet Numbering

This amendment to IEEE Std 802.1AE-2006 allows more than 2^{32} MACsec protected frames to be sent using a single Secure Association Key (SAK) by enabling the use of a 64-bit Packet Number (PN) and specifying two Cipher Suites (GCM-AES-XPN-128 and GCM-AES-XPN-256) that use that extended packet numbering as part of their Initial Value (IV) construction. MACsec frame formats and principles of MAC Security Entity operation remain unchanged. Changes are applied to the base text of IEEE Std 802.1AE-2006 as amended by IEEE Std 802.1AEbn-2011

7.70.4.2 Security TAG

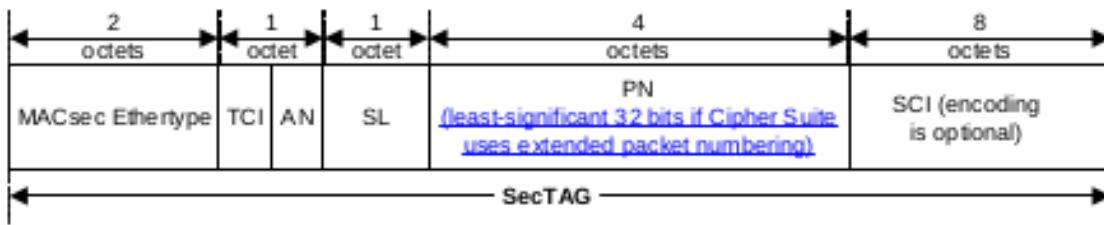


Figure 9-2—SecTAG format

Figure 7.87: Security Tag Format for Extended Packet Numbers

7.70.4.3 Packet Number (PN)

The 32 least significant bits of the PN are encoded in octets 5 through 8 of the SecTAG to

- Provide a unique IV PDU for all MPDUs transmitted using the same SA
- Support replay protection

NOTE 1 — As specified in this clause, the IV used by the default Cipher Suite (GCM-AES-128) comprises the SCI (even if the SCI is not transmitted in the SecTAG) and the a 32-bit PN. Subject to proper unique MAC Address allocation procedures, the SCI is a globally unique identifier for a SecY. To satisfy the IV uniqueness requirements of CTR mode of operation, a fresh key is used before PN values are reused

NOTE 2 — If the Current Cipher Suite provides extended packet numbering, i.e. uses a 64-bit PN, the 32 least significant bits of the PN are conveyed in this SecTAG field and the 32 most significant bits are recovered on receipt as specified in 10.6. The IV used by such a Cipher Suite (e.g. GCM-AES-XPN-128, 14.7) comprises a 32-bit S-SCI distributed by key agreement protocol and unique for each SCI within the scope of the CA (and hence within potential users of the same SAK) and the 64-bit non-repeating PN

7.70.4.4 PN recovery and preliminary replay check

If the Current Cipher Suite does not use extended packet numbering, i.e. the PN comprises 32-bits, the value of the PN is that of the lower 32 bits decoded from the SecTAG of the received frame. If extended packet numbering is used, the 32 most significant bits are recovered for each received frame as specified in by applying the assumption that they have remained unchanged since their use in the frame with the lowest acceptable PN (10.6.2) — unless the most significant of the 32 least significant bits of the lowest acceptable PN is set and the corresponding bit of the received PN is not set, in which case the 32 most significant bits of the PN are those of the lowest acceptable PN incremented by one

NOTE—If a large number of successive frames were to be lost ($2^{30} - 1$, corresponding to approximately 9 seconds of full utilization of a 400 Gb/s link by minimum sized Ethernet frames) subsequent receipt of MACsec frames might fail to establish a correct PN value. MKA, the MACsec Key Agreement protocol specified in IEEE Std 802.1X and its amendments communicates the value of the high order bits periodically to recover from this eventuality

7.70.4.5 verification controls

If the Current Cipher Suite uses extended packet numbering, i.e. a 64-bit PN, then the maximum value of replayWindow used in the Secure Frame Verification process (10.6) is $2^{30} - 1$, but any higher value set by network management is retained for possible subsequent use with a different Cipher Suite and will be reported if read by network management. This provision provides compatibility with prior revisions of this standard, though it is unlikely that such a high value of replayWindow would have been used

7.70.4.6 Receive SA creation

A receive SA is created for an existing SC on request from the KaY, with the following parameters:

- The association number, AN, for the SA
- nextPN (10.6, 10.6.5)
- lowestPN, the lowest acceptable PN value for a received frame (10.6, 10.6.2, 10.6.4, 10.6.5)
- A reference to an SAK that is unchanged for the life of the SA

and, if the Current Cipher Suite uses extended packet numbering (e.g. 14.7, 14.8), the following parameter:

- an SSCI, unique within all the SecY's (each associated with a KaY, and identified by an SCI) using the SAK, and subsequently available for Cipher Suite protection and validation operations

7.70.4.7 Transmit SA creation

An SA is created for the transmit SC on request from the KaY, with the following parameters:

- AN, the association number for the SA
- nextPN, the initial value of Transmit PN (10.5.2) for the SA
- confidentiality, True if the SA is to provide confidentiality as well as integrity for transmitted frames
- A reference to an SAK that is unchanged for the life of the SA

and, if the Current Cipher Suite uses extended packet numbering(e.g. 14.7, 14.8), the following parameter:

- an SSCI, unique within all the SecY's (each associated with a KaY, and identified by an SCI) using the SAK, and subsequently available for Cipher Suite protection and validation operations

7.70.4.8 SAK creation

An SAK record is created on request from the KaY, with the following parameters:

- The SAK value
- A Key Identifier, used by network management to reference the key

and, if the Current Cipher Suite uses extended packet numbering, the following parameter:

- Salt, a 96-bit parameter subsequently available for Cipher Suite protection and validation operations

7.70.4.9 Cipher Suite use

The PN (Packet Number, 3.27, 8.3) is a 32-bit number that is never zero, is incremented each time a protect request is made for a given SCI, and is never repeated for an SCI unless the SAK is changed. The size of the PN depends on the Cipher Suite, and is 32-bits unless otherwise specified. Cipher suites that provide extended packet numbering use a 64-bit PN. Irrespective of the size of the PN, only the least significant 32- bits are conveyed in the SecTAG. If extended packet numbering is used, the most significant 32-bits are recovered for each received frame as specified in 10.6.2.

7.70.4.10 Cipher Suite conformance

Table 14-1—MACsec Cipher Suites

Cipher Suite # Identifier	Cipher Suite Name	Services provided		Mandatory/Optional	Defining Clause
		Integrity without confidentiality	Integrity and confidentiality		
00-80-C2-00-01-00-00-01	GCM-AES-128	Yes	Yes	Mandatory	14.5
00-80-C2-00-01-00-00-02	GCM-AES-256	Yes	Yes	Optional	14.6
00-80-C2-00-01-00-00-03	GCM-AES-XPN-128	Yes	Yes	Optional	14.7
00-80-C2-00-01-00-00-04	GCM-AES-XPN-256	Yes	Yes	Optional	14.8

Figure 7.88: Ciphersuites for Macsec Extended Packet Number Packets

GCM-AES-256

GCM-AES-256 uses the Galois/Counter Mode of operation with the AES-256 symmetric block cipher, as specified in this clause by reference to the terms K, IV, A, P, C, T used in NIST SP 800-38D. K is the 256 bit SAK. The 64 most significant bits of the 96-bit IV are the octets of the SCI, encoded as a binary number (9.1). The 32 least significant bits of the 96-bit IV are the octets of the PN, encoded as a binary number (9.1). T is the ICV, and is 128 bits long. When the bit-strings A, P, and C are specified in terms of octet strings, earlier octets compose earlier bits, and more significant bits in each octet are earlier

NOTE — The bit strings obtained by transforming MAC Address and data octets using these rules do not correspond to 802.3 ‘wire order’ for frame transmission.

When the Default Cipher Suite is used for Integrity Protection

- A is the Destination MAC Address, Source MAC Address, and the octets of the SecTAG and User Data concatenated in that order
- P is null
- The Secure Data is the octets of the User Data, without modification

When the Default this Cipher Suite is used for Confidentiality Protection without a confidentiality offset

- A is the Destination MAC Address, Source MAC Address, and the octets of the SecTAG concatenated in that order
- P is the octets of the User Data
- The Secure Data is C

When the Default this Cipher Suite is used for Confidentiality Protection with a confidentiality offset

- A is the Destination MAC Address, Source MAC Address, and the octets of the SecTAG and the first confidentialityOffset (10.7.24) octets of the User Data concatenated in that order

- P is the remaining octets of the User Data
- The Secure Data is the first confidentialityOffset octets of the User Data concatenated with C, in that order

GCM-AES-XPN-128

Each instance of the GCM-AES-XPN-128 Cipher Suite, i.e. the protection and validation capabilities created for a given SAK at the request of the KaY (10.7.26, Figure 10-6) maintains an instance of the following parameter as specified in 10.7.23:

- Salt, a 96-bit value distributed by key agreement protocol to all members of the CA.

The MACsec Key Agreement (MKA) protocol specified in IEEE 802.1X-2010 does not include explicit parameters for distributing the Salt (applicable to all SAs using a given SAK) or the SSCI (applicable to a given SA, using a given SAK) explicitly. Each KaY computes these parameters from the MKA protocol information as follows. The 64 least-significant bits of the Salt comprise the SCI of the MKA Key Server, and the 32 most significant bits of the Salt comprise the value obtain by the exclusive-or of the 32 most significant and the 32 least significant bits of that SCI. The KaY with numerically greatest SCI uses the SSCI value 0x0001, the KaY with the next to the greatest SCI uses the SSCI value 0x0002, and so on

NOTE 1 — This procedure does not ensure that the Salt is secret. Readers of this standard are encouraged to consult the latest revision of 802.1X and its amendments

NOTE 2 — MKA guarantees that each KaY that receives a given SAK has a unique SCI, and these SCIs are present in every MKPDU that conveys a (key-wrapped) SAK

GCM-AES-XPN-128 uses the Galois/Counter Mode of operation with the AES-128 symmetric block cipher, as specified in this clause by reference to the terms K, IV, A, P, C, T used in NIST SP 800-38D. K is the 128 bit SAK. The 32 most significant bits of the 96-bit IV are the octets of the SSCI, encoded as a binary number (9.1) and exclusive-or'd with the 32 most significant bits of the Salt. The 64 least significant bits of the 96-bit IV are the octets of the PN, encoded as a binary number (9.1) and exclusive-or'd with the 64 least significant bits of the Salt. T is the ICV, and is 128 bits long. When the bit-strings A, P, and C are specified in terms of octet strings, earlier octets compose earlier bits, and more significant bits in each octet are earlier

NOTE 3 — The bit strings obtained by transforming MAC Address and data octets using these rules do not correspond to 802.3 'wire order' for frame transmission

When this Cipher Suite is used for Integrity Protection

- A is the Destination MAC Address, Source MAC Address, and the octets of the SecTAG and User Data concatenated in that order
- P is null
- The Secure Data is the octets of the User Data, without modification

When this Cipher Suite is used for Confidentiality Protection without a confidentiality offset

- A is the Destination MAC Address, Source MAC Address, and the octets of the SecTAG concatenated in that order
- P is the octets of the User Data
- The Secure Data is C

This Cipher Suite does not provide Confidentiality Protection with a confidentiality offset

GCM-AES-XPN-256

Each instance of the GCM-AES-XPN-256 Cipher Suite, i.e. the protection and validation capabilities created for a given SAK at the request of the KaY (10.7.26, Figure 10-6) maintains an instance of the following parameter as specified in 10.7.23:

- Salt, a 96-bit value distributed by key agreement protocol to all members of the CA

The MACsec Key Agreement (MKA) protocol specified in IEEE 802.1X-2010 does not include explicit parameters for distributing the Salt (applicable to all SAs using a given SAK) or the SSCI (applicable to a given SA, using a given SAK) explicitly. Each KaY computes these parameters from the MKA protocol information as follows. The 64 least-significant bits of the Salt comprise the SCI of the MKA Key Server, and the 32 most significant bits of the Salt comprise the value obtain by the exclusive-or of the 32 most significant and the 32 least significant bits of that SCI. The KaY with numerically greatest SCI uses the SSCI value 0x0001, the KaY with the next to the greatest SCI uses the SSCI value 0x0002, and so on

NOTE 1 — This procedure does not ensure that the Salt is secret. Readers of this standard are encouraged to consult the latest revision of 802.1X and its amendments

NOTE 2 — MKA guarantees that each KaY that receives a given SAK has a unique SCI, and these SCIs are present in every MKPDU that conveys a (key-wrapped) SAK

GCM-AES-XPN-256 uses the Galois/Counter Mode of operation with the AES-256 symmetric block cipher, as specified in this clause by reference to the terms K, IV, A, P, C, T used in NIST SP 800-38D

K is the 256 bit SAK. The 32 most significant bits of the 96-bit IV are the octets of the SSCI, encoded as a binary number (9.1) and exclusive-or'd with the 32 most significant bits of the Salt. The 64 least significant bits of the 96-bit IV are the octets of the PN, encoded as a binary number (9.1) and exclusive-or'd with the 64 least significant bits of the Salt. T is the ICV, and is 128 bits long. When the bit-strings A, P, and C are specified in terms of octet strings, earlier octets compose earlier bits, and more significant bits in each octet are earlier

NOTE 3 — The bit strings obtained by transforming MAC Address and data octets using these rules do not correspond to 802.3 'wire order' for frame transmission

When this Cipher Suite is used for Integrity Protection

- A is the Destination MAC Address, Source MAC Address, and the octets of the SecTAG and User Data concatenated in that order.
- P is null
- The Secure Data is the octets of the User Data, without modification

When this Cipher Suite is used for Confidentiality Protection without a confidentiality offset

- A is the Destination MAC Address, Source MAC Address, and the octets of the SecTAG concatenated in that order
- P is the octets of the User Data
- The Secure Data is C

This Cipher Suite does not provide Confidentiality Protection with a confidentiality offset

For API Documentation:

See Also

[ProtocolPP::jmacsec](#)
[ProtocolPP::jmacsecsa](#)
[ProtocolPP::jprotocol](#)
[ProtocolPP::jmodes](#)
[ProtocolPP::jarray](#)
[ProtocolPP::jrand](#)
[ProtocolPP::jenum](#)

For Additional Documentation:

See Also

[jmacsec](#)
[jmacsecsa](#)
[jprotocol](#)
[jmodes](#)
[jarray](#)
[jrand](#)
[jenum](#)

Protocol++ modified by : John Peter Greninger © John Peter Greninger 2015-2019 All Rights Reserved

All copyrights and trademarks are the property of their respective owners

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer
- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution
- Any and all modifications must be returned to John Peter Greninger at GitHub.com <https://github.com/jpgreninger/protocolpp> for evaluation. Inclusion of modifications in the source code shall be determined solely by John Peter Greninger. Failure to provide modifications shall render this license NULL and VOID and revoke any rights to use of Protocol++
- Commercial use requires a fee-based license obtainable at www.protocolpp.com
- Academic use requires written and notarized permission from John Peter and Sheila Rocha Greninger
- US Copyrights at [https://www.copyright.gov/](http://www.copyright.gov/)
 - TXu002059872 (Version 1.0.0)
 - TXu002066632 (Version 1.2.7)
 - TXu002082674 (Version 1.4.0)
 - TXu002097880 (Version 2.0.0)

The names of its contributors may not be used to endorse or promote products derived from this software without specific prior written permission

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE

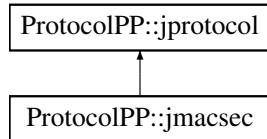
The documentation for this class was generated from the following file:

- [include/jmacsec.h](#)

7.71 ProtocolPP::jmacsec Class Reference

```
#include <jmacsec.h>
```

Inheritance diagram for ProtocolPP::jmacsec:



Public Member Functions

- `jmacsec (std::shared_ptr<jmacsecsa> &sec)`
- `void encaps_packet (std::shared_ptr<jarray<uint8_t>> &input, std::shared_ptr<jarray<uint8_t>> &output)`
- `void decap_packet (std::shared_ptr<jarray<uint8_t>> &input, std::shared_ptr<jarray<uint8_t>> &output)`
- `void set_hdr (jarray<uint8_t> &hdr)`
- `void set_field (field_t field, uint64_t value)`
- `jarray<uint8_t> get_hdr ()`
- `uint64_t get_field (field_t field, jarray<uint8_t> &hdr)`
- `uint64_t get_field (field_t field)`
- `void to_xml (tinyxml2::XMLPrinter &myxml, direction_t direction)`

Additional Inherited Members

7.71.1 Constructor & Destructor Documentation

7.71.1.1 ProtocolPP::jmacsec::jmacsec (std::shared_ptr<jmacsecsa> & sec)

Create an instance of the macsec protocol. Use update_sec to create the security association

Parameters

<code>sec</code>	- security association for this flow
------------------	--------------------------------------

7.71.2 Member Function Documentation

7.71.2.1 void ProtocolPP::jmacsec::decap_packet (std::shared_ptr<jarray<uint8_t>> & input, std::shared_ptr<jarray<uint8_t>> & output) [virtual]

Decapsulate the packet with macsec

Parameters

<code>input</code>	- shared pointer with array of bytes that represent the encapsulated macsec packet
<code>output</code>	- shared pointer with array of bytes that represent the decapsulated payload

Input payload could be TCP, UDP, IP, Ipsec, etc



Encapsulated output packet has format as follows:

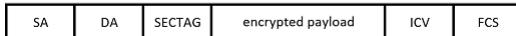


Figure 7.89: Decapsulation of Macsec Packet Into Payload

Implements [ProtocolPP::jprotocol](#).

7.71.2.2 void ProtocolPP::jmacsec::encap_packet (std::shared_ptr<jarray<uint8_t>> &*input*, std::shared_ptr<jarray<uint8_t>> &*output*) [virtual]

Encapsulate the payload with macsec

Parameters

<i>input</i>	- shared pointer with array of bytes that represent the payload
<i>output</i>	- shared pointer with array of bytes that represent the encapsulated macsec packet

Encapsulated input packet has format as follows:



Output payload could be TCP, UDP, IP, Ipsec, etc



Figure 7.90: Encapsulation of Payload Into Macsec Packet

Implements [ProtocolPP::jprotocol](#).

7.71.2.3 uint64_t ProtocolPP::jmacsec::get_field (field_t *field*, jarray<uint8_t> &*hdr*) [virtual]

Return the macsec field

Parameters

<i>field</i>	- MACSEC field to retrieve
<i>hdr</i>	- header to extract field from

Returns

Current macsec field value

Implements [ProtocolPP::jprotocol](#).

7.71.2.4 uint64_t ProtocolPP::jmacsec::get_field (field_t *field*) [virtual]

Return the macsec field from the security association

Parameters

<i>field</i>	- MACSEC field to retrieve
--------------	----------------------------

Returns

Current macsec field value

Reimplemented from [ProtocolPP::jprotocol](#).

7.71.2.5 `jarray<uint8_t> ProtocolPP::jmacsec::get_hdr() [virtual]`

Retrieve MACSEC header

Returns

MacSEC header

Implements [ProtocolPP::jprotocol](#).

7.71.2.6 `void ProtocolPP::jmacsec::set_field(field_t field, uint64_t value) [virtual]`

Update MACSEC field

Parameters

<i>field</i>	- MACSEC field to update
<i>value</i>	- new value for the MACSEC field

Implements [ProtocolPP::jprotocol](#).

7.71.2.7 `void ProtocolPP::jmacsec::set_hdr(jarray< uint8_t > & hdr) [virtual]`

Update MACSEC header

Parameters

<i>hdr</i>	- new MacSEC header
------------	---------------------

Implements [ProtocolPP::jprotocol](#).

7.71.2.8 `void ProtocolPP::jmacsec::to_xml(tinyxml2::XMLPrinter & myxml, direction_t direction) [virtual]`

Write the protocol and security objects as XML

Parameters

<i>myxml</i>	- XMLPrinter object to print with
<i>direction</i>	- randomization

Implements [ProtocolPP::jprotocol](#).

The documentation for this class was generated from the following file:

- [include/jmacsec.h](#)

7.72 jmacsecsa Class Reference

```
#include "include/jmacsecsa.h"
```

7.72.1 Detailed Description

7.72.2 MACsec Security Association

Information provided by IEEE <https://standards.ieee.org/getieee802/download/802.1A-E-2006.pdf>

Encoding of MACsec protocol data units

This clause specifies the structure and encoding of the MACsec Protocol Data Units (MPDUs) exchanged between MAC Security Entities (SecYs). It

- Specifies rules for the representation and encoding of protocol fields
- Specifies the major components of each MPDU, and the fields they comprise
- Reviews the purpose of each field, and the functionality provided
- Specifies validation of the MPDU on reception
- Documents the allocation of an EtherType value, the MACsec EtherType, to identify MPDUs

NOTE—The MPDU validation checks specified in this clause are deliberately limited to ensuring successful decoding, and do not overlap with the specification of SecY operation (Clause 10).

Structure, representation, and encoding

All MPDUs shall contain an integral number of octets. The octets in a MPDU are numbered starting from 1 and increasing in the order they are put into the MAC Service Data Unit (MSDU) that accompanies a request to or indication from the instance of the MAC Internal Sublayer Service (ISS) used by a SecY. The bits in an octet are numbered from 1 to 8 in order of increasing bit significance, where 1 is the least significant bit in the octet. Where octets and bits within a MPDU are represented using a diagram, octets shown higher on the page than subsequent octets and octets shown to the left of subsequent octets at the same height on the page are lower numbered, bits shown to the left of other bits within the same octet are higher numbered. Where two or more consecutive octets are represented as hexadecimal values, lower numbered octet(s) are shown to the left and each octet following the first is preceded by a hyphen, e.g., 01-80-C2-00-00-00.

When consecutive octets are used to encode a binary number, the lower octet number has the more significant value. When consecutive bits within an octet are used to encode a binary number, the higher bit number has the most significant value. When bits within consecutive octets are used to encode a binary number, the lower octet number composes the more significant bits of the number. A flag is encoded as a single bit, and is set (True) if the bit takes the value 1, and clear (False) otherwise. The remaining bits within the octet can be used to encode other protocol fields.

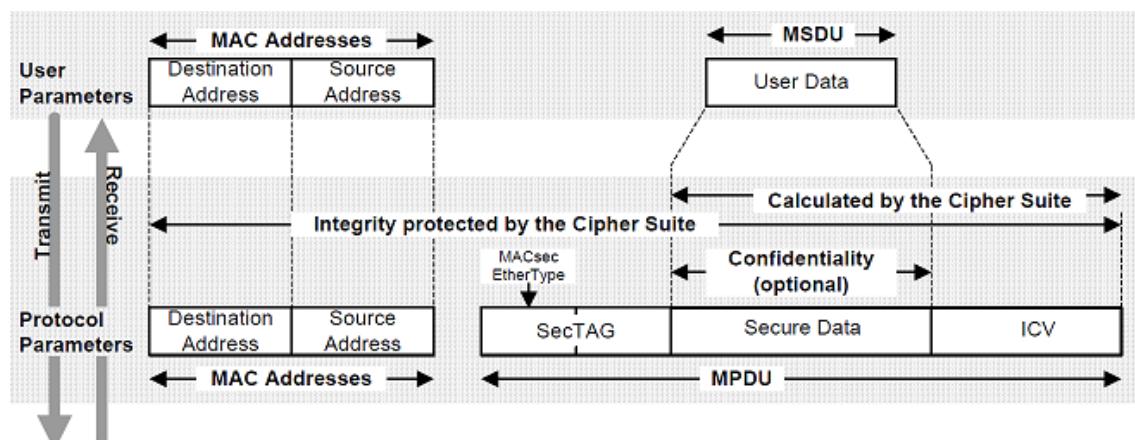


Figure 8-1—MACsec

Figure 7.91: Macsec Packet Structure

Major components

Each MPDU comprises

- A Security TAG (SecTAG)
- Secure Data (9.10 c) An Integrity Check Value (ICV)

Each of these components comprises an integral number of octets and is encoded in successive octets of the MPDU as illustrated in Figure 9-1

NOTE—The MPDU does not include the source and destination MAC addresses, as these are separate parameters of the service requests and indications to and from the insecure service that supports MACsec

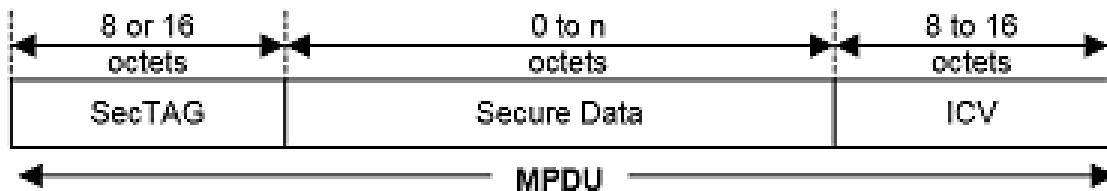


Figure 9-1—MPDU components

Figure 7.92: Packet Data Unit (PDU) For Macsec

Security TAG

The Security TAG (SecTAG) is identified by the MACsec EtherType, and conveys the

- TAG Control Information (TCI, 9.5)
- Association Number (AN, 9.6)
- Short Length (SL, 9.7)
- Packet Number (PN, 9.8)
- Optionally encoded Secure Channel Identifier (SCI, 9.9).

The format of the SecTAG is illustrated in Figure 9-2

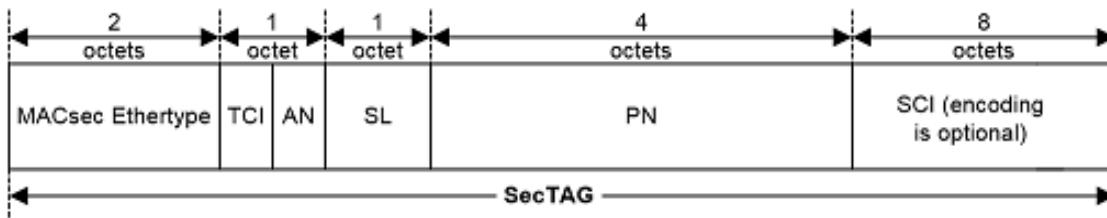


Figure 9-2—SecTAG format

Figure 7.93: Security TAG For Macsec

MACsec EtherType

The MACsec EtherType comprises octet 1 and octet 2 of the SecTAG. It is included to allow

- Coexistence of MACsec capable systems in the same environment as other systems
- Incremental deployment of MACsec capable systems
- Peer SecYs to communicate using the same media as other communicating entities
- Concurrent operation of Key Agreement protocols that are independent of the MACsec protocol and the Current Cipher Suite
- Operation of other protocols and entities that make use of the service provided by the SecY's Uncontrolled Port to communicate independently of the Key Agreement state

Table 9-1—MACsec EtherType allocation

Tag Type	Name	Value
IEEE 802.1AE Security TAG	MACsec EtherType	88-E5

Figure 7.94: Ethernet Type of the Macsec Packet

The encoding of the MACsec EtherType in the MPDU is illustrated in the next figure

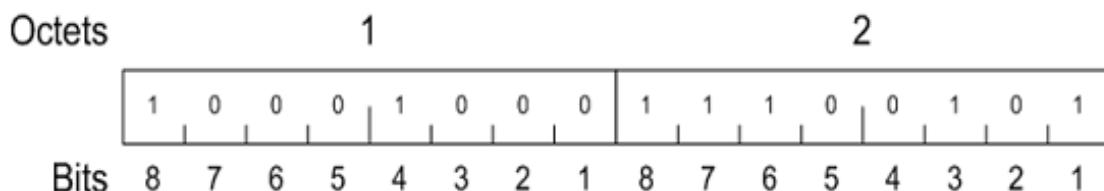


Figure 9-3—MACsec EtherType encoding

Figure 7.95: Position of the EtherType Field in the Macsec Packet

TAG Control Information (TCI)

The TCI field comprises bits 8 through 3 of octet 3 (Figure 9-4) of the SecTAG. These bits facilitate

- Version numbering of the MACsec protocol without changing the MACsec EtherType
- Optional use of the MAC Source Address parameter to convey the SCI
- Optional inclusion of an explicitly encoded SCI (7.1.2, Figure 7-7)
- Use of the EPON (Clause 12) Single Copy Broadcast capability, without requiring an explicit SCI to distinguish the SCB Secure Channel
- Extraction of the User Data from MPDUs by systems that do not possess the SAK (8.1.2, 8.1.4) when confidentiality is not being provided
- Determination of whether confidentiality or integrity alone are in use

The encoding of the MACsec TCI in the MPDU is illustrated in Figure 9-4.

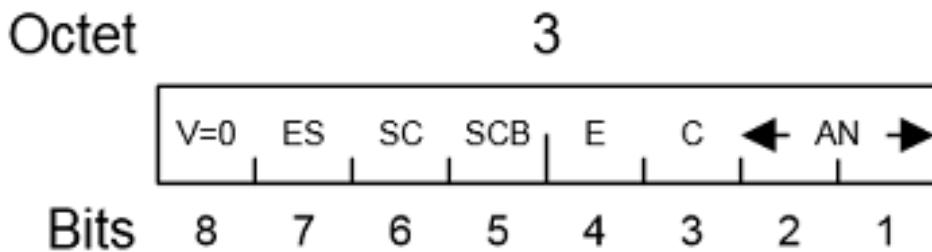


Figure 9-4—MACsec TCI and AN Encoding

Figure 7.96: TAG Control Information for the Macsec Packet

TCIAN bit encoding is as follows:

- The version number shall be 0 and is encoded in bit 8. NOTE—Future versions of the MACsec protocol may use additional bits of the TCI to encode the version number. The fields and format of the remainder of the MPDU may change if the version number changes
- If the MPDU is transmitted by an end station and the first 6 octets of the SCI are equal to the value of the octets of MAC Source Address parameter of the ISS request in canonical format order, bit 7 [the End Station (ES) bit] of the TCI may be set
- If the ES bit is set, bit 6 (the SC bit) shall not be set and an SCI shall not be explicitly encoded in the SecTAG. The ES bit is clear if the Source Address is not used to determine the SCI. If an SCI (9.9, 7.1.2) is explicitly encoded in the SecTAG, bit 6 (the SC bit) of the TCI shall be set. The SC bit shall be clear if an SCI is not present in the SecTAG
- If and only if the MPDU is associated with the Secure Channel that supports the EPON Single Copy Broadcast capability, bit 5 (the SCB bit) of the TCI may be set
- If the SCB is set, bit 6 (the SC bit) shall not be set and an SCI shall not be explicitly included in the SecTAG. If the ES bit is set and the SCB is not set, the SCI comprises a Port Identifier (7.1.2) component of 00-01. If the SCB bit is set, the Port Identifier (7.1.2) component has the reserved SCB value of 00-00
- If the Encryption (E) bit is set and the Changed Text (C) bit is clear, the frame is not processed by the SecY (10.6) but is reserved for use by the KaY. Otherwise, the E bit is set if and only if confidentiality is being provided and is clear if integrity only is being provided
- The C bit is clear if and only if the Secure Data is exactly the same as the User Data and the ICV is 16 octets long. When the Default Cipher Suite (14.5) is used for integrity protection only, the Secure Data is the unmodified User Data, and a 16 octet ICV is used. Both the E bit and the C bit are therefore clear, and the data conveyed by MACsec is available to applications, such as network management, that need to see the data but are not trusted with the SAK that would permit its modification. Other Cipher Suites may also integrity protect data without modifying it, and use a 16 octet ICV, enabling read access to the data by other applications. The E and C bits are also clear for such Cipher Suites when integrity only is provided
- Some cryptographic algorithms modify or add to the data even when integrity only is being provided, or use an ICV that is not 16 octets long. The C bit is never clear for such an algorithm, even if the E bit is clear to indicate that confidentiality is not provided. Recovery of the data from a MACsec frame with the E bit clear and the C bit set requires knowledge of the Cipher Suite at a minimum. That information is not provided in the MACsec frame. If both the C bit and E bit are set, confidentiality of the original User Data is being provided.

Association Number (AN)

The AN is encoded as an integer in bits 1 and 2 of octet 3 of the SecTAG (Figure 9-4) and identifies up to four different SAs within the context of an SC. NOTE—Although each receiving SecY only needs to maintain two SAs per SC, the use of a 2-bit AN simplifies the design of protocols that update values associated with each of the SAs.

Short Length (SL)

SL is an integer encoded in bits 1 through 6 of octet 4 of the SecTAG and is set to the number of octets in the Secure Data (9.10) field, i.e., the number of octets between the last octet of the SecTAG and the first octet of the ICV, if that number is less than 48. Otherwise, SL is set to zero. If the number is zero then the frame is deemed not to have been short. The Secure Data field always comprises at least one octet. Bits 7 and 8 of octet 4 shall be zero.

Packet Number (PN)

The PN is encoded in octets 5 through 8 of the SecTAG to

- Provide a unique IV PDU for all MPDUs transmitted using the same SA
- Support replay protection

NOTE—As specified in this clause, the IV used by the default Cipher Suite (GCM-AES-128) comprises the SCI (even if the SCI is not transmitted in the SectAG) and the PN. Subject to proper unique MAC Address allocation procedures, the SCI is a globally unique identifier for a SecY. To satisfy the IV uniqueness requirements of CTR mode of operation, a fresh key is used before PN values are reused.

Secure Channel Identifier (SCI)

If the SC bit in the TCI is set, the SCI (7.1.2, 8.2.1) is encoded in octets 9 through 16 of the SecTAG, and facilitates

- Identification of the SA where the CA comprises three or more peers
- Network management identification of the SecY that has transmitted the frame Octets 9 through 14 of the SecTAG encode the System Identifier component of the SCI. This comprises the six octets of a globally unique MAC address uniquely associated with the transmitting SecY. The octet values and their sequence conform to the Canonical Format specified by IEEE Std 802. Octets 15 and 16 of the SecTAG encode the Port Identifier component of the SCI, as an integer.

NOTE —The 64-bit value FF-FF-FF-FF-FF-FF is never used as an SCI and is reserved for use by implementations to indicate the absence of an SC or an SCI in contexts where an SC can be present.

An explicitly encoded SCI field in the SecTAG is not required on point-to-point links, which are identified by the operPointToPointMAC status parameter of the service provider. In the point-to-point case, the secure association created by the SecY for the peer SecYs, together with the direction of transmission of the secured MPDU, can be used to identify the transmitting SecY and therefore an explicitly encoded SCI is unnecessary. Although the SCI does not have to be repeated in each frame when only two SecYs participate in a CA (see Clause 8, Clause9, and Clause10), the SCI still forms part of the cryptographic computation.

For API Documentation:

See Also

[ProtocolPP::jmacsec](#)
[ProtocolPP::jmacsecsa](#)
[ProtocolPP::jprotocol](#)
[ProtocolPP::jmodes](#)
[ProtocolPP::jarray](#)
[ProtocolPP::jrand](#)
[ProtocolPP::jenum](#)

For Additional Documentation:

See Also

[jmacsec](#)
[jmacsecsa](#)
[jprotocol](#)
[jmodes](#)
[jarray](#)
[jrand](#)
[jenum](#)

Protocol++ modified by : John Peter Greninger © John Peter Greninger 2015-2019 All Rights Reserved

All copyrights and trademarks are the property of their respective owners

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer
- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution
- Any and all modifications must be returned to John Peter Greninger at GitHub.com <https://github.com/jpgreninger/protocolpp> for evaluation. Inclusion of modifications in the source code shall be determined solely by John Peter Greninger. Failure to provide modifications shall render this license NULL and VOID and revoke any rights to use of Protocol++
- Commercial use requires a fee-based license obtainable at www.protocolpp.com
- Academic use requires written and notarized permission from John Peter and Sheila Rocha Greninger
- US Copyrights at [https://www.copyright.gov/](http://www.copyright.gov/)
 - TXu002059872 (Version 1.0.0)
 - TXu002066632 (Version 1.2.7)
 - TXu002082674 (Version 1.4.0)
 - TXu002097880 (Version 2.0.0)

The names of its contributors may not be used to endorse or promote products derived from this software without specific prior written permission

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE

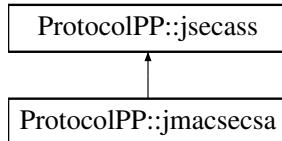
The documentation for this class was generated from the following file:

- [include/jmacsecsa.h](#)

7.73 ProtocolPP::jmacsecsa Class Reference

```
#include <jmacsecsa.h>
```

Inheritance diagram for ProtocolPP::jmacsecsa:



Public Member Functions

- `jmacsecsa ()`
- `jmacsecsa (direction_t dir, macsecmode_t mode, bool usext, bool enreceive, bool entransmit, bool protectframes, uint8_t tcian, uint8_t sl, uint16_t ethtype, uint32_t pn, uint32_t xpn, uint32_t ssci, unsigned int arlen, unsigned int icvlen, unsigned int keylen, uint64_t src, uint64_t dst, uint64_t sci, jarray< uint8_t > arwin, std::shared_ptr< jarray< uint8_t >> sakey, std::shared_ptr< jarray< uint8_t >> salt)`
- `jmacsecsa (jmacsecsa &rhs)`
- `jmacsecsa (std::shared_ptr< jmacsecsa > &rhs)`
- template<typename T >
 `void set_field (field_t field, T fieldval)`
- template<typename T >
 `T get_field (field_t field)`
- `void to_xml (tinyxml2::XMLPrinter &myxml, direction_t direction)`

7.73.1 Constructor & Destructor Documentation

7.73.1.1 ProtocolPP::jmacsecsa::jmacsecsa ()

Standard Constructor with defaults

```
*   std::shared_ptr<jmacsecsa> snd = std::make_shared<jmacsecsa>();
*
*   snd->set_field<uint32_t>(<b>field_t::PN</b>, 0x00000001);
*   snd->set_field<uint8_t>(<b>field_t::TCIAN</b>, 0xEE);
*   snd->set_field<std::shared_ptr<jarray<uint8_t>>>(<b>field_t::CIPHERKEY</b>,
*           std::make_shared<jarray<uint8_t>>>(m_rand->getbyte(16)));
*
```

7.73.1.2 ProtocolPP::jmacsecsa::jmacsecsa (direction_t dir, macsecmode_t mode, bool usext, bool enreceive, bool entransmit, bool protectframes, uint8_t tcian, uint8_t sl, uint16_t ethtype, uint32_t pn, uint32_t xpn, uint32_t ssci, unsigned int arlen, unsigned int icvlen, unsigned int keylen, uint64_t src, uint64_t dst, uint64_t sci, jarray< uint8_t > arwin, std::shared_ptr< jarray< uint8_t >> sakey, std::shared_ptr< jarray< uint8_t >> salt)

Security Association for MacSec

Parameters

<code>dir</code>	- Direction for processing (ENCAP or DECAP)
<code>mode</code>	- Mode for MACSEC (AES-GCM with 128-bit or 256-bit key with or without extended packet number)

		field type
<i>usext</i>	- use extended sequence number	direction_t
<i>enreceive</i>	- SA receiving frames	
<i>entransmit</i>	- SA transmitting frames	
<i>protectframes</i>	- SA protecting frames	
<i>tcian</i>	- TAG control information	
<i>sl</i>	- short length parameter	
<i>ethtype</i>	- ethernet type	
<i>pn</i>	- packet number this Phy is generating/receiving	bool
<i>xpn</i>	- if extended packet numbers are used, upper half of the PN	uint8_t
<i>ssci</i>	- short security control index (for extended packet number)	
<i>arlen</i>	- Number of packets to track in the replay window	
<i>icvlen</i>	- length of the ICV. Default is 16	
<i>keylen</i>	- length of the key (either 16 or 32 bytes)	
<i>src</i>	- source address for the stream	uint16_t
<i>dst</i>	- destination address for the stream	
<i>sci</i>	- security control index (64-bits)	
<i>arwin</i>	- Anti-replay window to track packets	uint32_t
<i>sakey</i>	- AES-GCM key for integrity of user data	
<i>salt</i>	- salt to XOR with SSCI, XPN, and PN when using extended packet numbers	

7.73.1.3 ProtocolPP::jmacsecsa::jmacsecsa (**jmacsecsa & rhs**)

Create an instance of the macsec protocol. Use update_sec to create the security association

Parameters

<i>rhs</i>	- security association for this flow
------------	--------------------------------------

7.73.1.4 ProtocolPP::jmacsecsa::jmacsecsa (std::shared_ptr<**jmacsecsa**> & rhs)

Create an instance of the macsec protocol. Use update_sec to create the security association

Parameters

<i>rhs</i>	- security association for this flow
------------	--------------------------------------

7.73.2 Member Function Documentation

7.73.2.1 template<typename T> T ProtocolPP::jmacsecsa::get_field (**field_t field**)

Return the macsec field

Parameters

<i>field</i>	- MACSEC field to retrieve
--------------	----------------------------

Returns

MACSEC field value

7.73.2.2 template<typename T> void ProtocolPP::jmacsecsa::set_field (**field_t field, T fieldval**)

Update MACSEC field

Parameters

<i>field</i>	- MACSEC field to update
<i>fieldval</i>	- new value for the MACSEC field

7.73.2.3 void **ProtocolPP::jmacsecsa::to_xml** (*tinyxml2::XMLPrinter & myxml*, *direction_t direction*) [virtual]

Write the protocol and security objects as XML

Parameters

<i>myxml</i>	- XMLPrinter object to print with
<i>direction</i>	- randomization

Implements [ProtocolPP::jsecass](#).

The documentation for this class was generated from the following file:

- [include/jmacsecsa.h](#)

7.74 InterfacePP::jmmu Class Reference

```
#include <jmmu.h>
```

Public Member Functions

- **jmmu ()**
constructor
- virtual **~jmmu ()**
standard deconstructor
- void **track** (std::string packet, std::vector< uint8_t * > &pointers)
- void **set_mem** (std::string packet, uint8_t *ptr)
- void **set_mem** (std::string packet, [ProtocolPP::iostream](#) *ptr)
- void **free_mem** (std::string packet)
- void **free ()**
free all pointers

7.74.1 Constructor & Destructor Documentation

7.74.1.1 [InterfacePP::jmmu::jmmu \(\)](#)

constructor

7.74.1.2 virtual [InterfacePP::jmmu::~jmmu \(\)](#) [inline], [virtual]

standard deconstructor

7.74.2 Member Function Documentation

7.74.2.1 void [InterfacePP::jmmu::free \(\)](#)

free all pointers

7.74.2.2 void InterfacePP::jmmu::free_mem (std::string *packet*)

free memory for the packet

Parameters

<i>packet</i>	name of packet associated with pointer
---------------	--

7.74.2.3 void InterfacePP::jmmu::set_mem (std::string *packet*, uint8_t * *ptr*)

Track the memory pointer

Parameters

<i>packet</i>	name of packet to associate pointer with
<i>ptr</i>	byte pointer for memory

7.74.2.4 void InterfacePP::jmmu::set_mem (std::string *packet*, ProtocolPP::jstream * *ptr*)

Track the stream memory pointer

Parameters

<i>packet</i>	name of stream to associate pointer with
<i>ptr</i>	stream pointer for memory

7.74.2.5 void InterfacePP::jmmu::track (std::string *packet*, std::vector< uint8_t * > & *pointers*)

Track the a set of memory pointers

Parameters

<i>packet</i>	name of packet to associate pointer with
<i>pointers</i>	track vector to hold pointers associated with packet

The documentation for this class was generated from the following file:

- jtestbench/include/jmmu.h

7.75 jmmu Class Reference

```
#include "include/jmmu.h"
```

7.75.1 Detailed Description

7.75.2 Management Unit for Protocol++ testbench

Protocol++ written by : John Peter Greninger © John Peter Greninger 2015-2019 All Rights Reserved

All copyrights and trademarks are the property of their respective owners

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer

- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution
- Any and all modifications must be returned to John Peter Greninger at GitHub.com <https://github.com/jpgreninger/protocolpp> for evaluation. Inclusion of modifications in the source code shall be determined solely by John Peter Greninger. Failure to provide modifications shall render this license NULL and VOID and revoke any rights to use of Protocol++
- Commercial use requires a fee-based license obtainable at www.protocolpp.com
- Academic use requires written and notarized permission from John Peter and Sheila Rocha Greninger
- US Copyrights at <https://www.copyright.gov/>
 - TXU002059872 (Version 1.0.0)
 - TXU002066632 (Version 1.2.7)
 - TXU002082674 (Version 1.4.0)
 - TXU002097880 (Version 2.0.0)

The names of its contributors may not be used to endorse or promote products derived from this software without specific prior written permission

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE

The documentation for this class was generated from the following file:

- [jtestbench/include/jmmu.h](#)

7.6 ProtocolPP::jmodes Class Reference

```
#include <jmodes.h>
```

Public Types

- enum **cipher_t** {
 AES, DES, DES_EDE3, Camellia,
 ARIA, SEED, SM4, CHACHA20,
 SNOWE, ZUCE, ARC4, NONE }
- enum **dir_t** { ENC =1, DEC =0, DOWNLINK =1, UPLINK =0 }
- enum **mode_t** {
 ECB, CBC, GCM, CTR,
 CCM, XCBC_MAC, CMAC, AEAD,
 AUTH, STREAM }

Public Member Functions

- jmodes** (**cipher_t** cipher, **dir_t** dir, **mode_t** mode, **Byte** *key, unsigned int keyLen, **Byte** *iv=NULL, unsigned int ivLen=0, unsigned int icvLen=16, uint32_t count=0, uint8_t bearer=0)

- `jmodes (auth_t auth, Byte *authkey=NULL, unsigned int authkeyLen=0, dir_t dir=DEC, uint32_t count=0, uint32_t fresh=0)`
- `~jmodes ()`
Standard deconstructor.
- `void ProcessData (Byte *input, Byte *output, unsigned int length, Byte *aad=NULL, unsigned int aadlen=0)`
- `void ProcessData (const Byte *input, unsigned int length)`
- `void context (Byte *context, unsigned int length)`
- `void result (Byte *result, unsigned int length)`

7.76.1 Member Enumeration Documentation

7.76.1.1 enum ProtocolPP::jmodes::cipher_t

Encryption Algorithm

Parameters

AES	- Advanced Encryption Standard
DES	- Data Encryption Standard
DES_EDE3	- Triple-DES
Camellia	- Japanese national standard cipher
ARIA	- Korean national standard cipher
SEED	- Korean block cipher
SM4	- Chinese block cipher
CHACHA20	- Stream cipher
SNOWE	- Snow3G encryption cipher
ZUCE	- ZUC encryption cipher
ARC4	- ARC4 encryption cipher

Enumerator

AES Advanced Encryption Standard.

DES Data Encryption Standard.

DES_EDE3 Triple-DES with 192-bit key.

Camellia Camellia Cipher.

ARIA ARIA cipher.

SEED SEED cipher.

SM4 SM4 cipher.

CHACHA20 CHACHA20 stream cipher.

SNOWE SNOW3G F8 encryption mode.

ZUCE ZUC F8 encryption mode.

ARC4 ARC4 stream cipher.

NONE No cipher used.

7.76.1.2 enum ProtocolPP::jmodes::dir_t

Direction of processing

Parameters

<i>ENC</i>	- Encrypt the data
<i>DEC</i>	- Decrypt the data
<i>DOWNLINK</i>	- Connection from Tower to User
<i>UPLINK</i>	- Connection from User to Tower

Enumerator

ENC Encryption.**DEC** Decryption.**DOWNLINK** Tower to user.**UPLINK** User to tower.

7.76.1.3 enum ProtocolPP::jmodes::mode_t

Mode of operation

Parameters

<i>ECB</i>	- Electronic Code Book (ECB)
<i>CBC</i>	- Cipher Block Chaining (CBC)
<i>GCM</i>	- Galois Counter Mode (GCM)
<i>CTR</i>	- Counter Mode (CTR)
<i>CCM</i>	- Counter with Cipher Block Chaining-Message Authentication Code (CCM)
<i>XCBC_MAC</i>	- CBC-MAC with 12-byte ICV
<i>CMAC</i>	- Block-Cipher based MAC algorithm
<i>AEAD</i>	- AEAD dual mode with POLY1305
<i>AUTH</i>	- Authentication mode
<i>STREAM</i>	- Stream cipher mode (ARC4, CHACHA20)

Enumerator

ECB Electronic Code Book.**CBC** Cipher Block Chaining.**GCM** Galois Counter Mode.**CTR** Counter Mode.**CCM** Counter with Cipher Block Chaining-Message Authentication Code.**XCBC_MAC** AES-XCBC-MAC-96 one way hash.**CMAC** Cipher-based MAC algorithm.**AEAD** AEAD dual mode with POLY1305.**AUTH** Authentication mode.**STREAM** Stream cipher mode (ARC4, CHACHA20)

7.76.2 Constructor & Destructor Documentation

7.76.2.1 ProtocolPP::jmodes (*cipher_t cipher*, *dir_t dir*, *mode_t mode*, *Byte * key*, *unsigned int keyLen*, *Byte * iv* = *NULL*, *unsigned int ivLen* = *0*, *unsigned int icvLen* = *16*, *uint32_t count* = *0*, *uint8_t bearer* = *0*)

Constructor for different cipher modes which supports key sizes of 128, 192, and 256 bits

Parameters

<i>cipher</i>	- Encryption algorithm to use (AES, DES, DES_EDE3, Camellia, SEED, SM4, ARIA, CHAC-HA20, ARC4)
<i>dir</i>	- direction of processing (ENC or DEC)
<i>mode</i>	- either ECB, CBC, CTR, GCM, XCBC_MAC, CMAC, AEAD, STREAM, or AUTH mode
<i>key</i>	- master key for processing
<i>keyLen</i>	- Length of the key in bytes
<i>iv</i>	- initialization vector for CBC/GCM
<i>ivLen</i>	- Length of the IV in bytes
<i>icvLen</i>	- Length of the ICV in bytes
<i>count</i>	- Count value for SNOW3G and ZUC
<i>bearer</i>	- BEARER value for SNOW3G and ZUC

7.76.2.2 `ProtocolPP::jmodes::jmodes (auth_t auth, Byte * authkey = NULL, unsigned int authkeyLen = 0, dir_t dir = DEC, uint32_t count = 0, uint32_t fresh = 0)`

Constructor for authentication modes

Parameters

<i>auth</i>	- Authentication (see jenum.h for modes)
<i>authkey</i>	- key for processing
<i>authkeyLen</i>	- Length of the key in bytes
<i>dir</i>	- direction of processing for SNOW3G and ZUC
<i>count</i>	- Count value for SNOW3G and ZUC
<i>fresh</i>	- FRESH/BEARER value for SNOW3G and ZUC

7.76.2.3 `ProtocolPP::jmodes::~jmodes () [inline]`

Standard deconstructor.

7.76.3 Member Function Documentation

7.76.3.1 `void ProtocolPP::jmodes::context (Byte * context, unsigned int length)`

Retrieves the context (IV)

Parameters

<i>context</i>	- data to retrieve
<i>length</i>	- length of the data to retrieve (bytes)

7.76.3.2 `void ProtocolPP::jmodes::ProcessData (Byte * input, Byte * output, unsigned int length, Byte * aad = NULL, unsigned int aadlen = 0)`

Encrypts or decrypts the data based on direction and mode

Parameters

<i>input</i>	- input data to process
--------------	-------------------------

<i>output</i>	- processed data
<i>length</i>	- length of the data in bytes
<i>aad</i>	- authentication only data
<i>aadlen</i>	- length of the AAD data in bytes

7.76.3.3 void ProtocolPP::jmodes::ProcessData (const Byte * *input*, unsigned int *length*)

Authenticates the input data

Parameters

<i>input</i>	- input data to process
<i>length</i>	- length of the data in bytes

7.76.3.4 void ProtocolPP::jmodes::result (Byte * *result*, unsigned int *length*)

Retrieves the result (ICV)

Parameters

<i>result</i>	- data to retrieve
<i>length</i>	- length of the data to retrieve (bytes)

The documentation for this class was generated from the following file:

- [include/jmodes.h](#)

7.77 jmodes Class Reference

#include "jmodes.h"

7.77.1 Detailed Description

7.77.2 Block Modes of Operation

See https://en.wikipedia.org/wiki/Block_cipher_mode_of_operation

In cryptography, a mode of operation is an algorithm that uses a block cipher to encrypt messages of arbitrary length in a way that provides confidentiality or authenticity. A block cipher by itself is only suitable for the secure cryptographic transformation (encryption or decryption) of one fixed-length group of bits called a block. A mode of operation describes how to repeatedly apply a cipher's single-block operation to securely transform amounts of data larger than a block.

Most modes require a unique binary sequence, often called an initialization vector (IV), for each encryption operation. The IV has to be non-repeating and, for some modes, random as well. The initialization vector is used to ensure distinct ciphertexts are produced even when the same plaintext is encrypted multiple times independently with the same key.[6] Block ciphers have one or more block size(s), but during transformation the block size is always fixed. Block cipher modes operate on whole blocks and require that the last part of the data be padded to a full block if it is smaller than the current block size. There are, however, modes that do not require padding because they effectively use a block cipher as a stream cipher; such ciphers are capable of encrypting arbitrarily long sequences of bytes or bits.

Historically, encryption modes have been studied extensively in regard to their error propagation properties under various scenarios of data modification. Later development regarded integrity protection as an entirely separate cryptographic goal. Some modern modes of operation combine confidentiality and authenticity in an efficient way, and are known as authenticated encryption modes.

History and standardization

The earliest modes of operation, ECB, CBC, OFB, and CFB (see below for all), date back to 1981 and were specified in FIPS 81, DES modes of Operation. In 2001, the US National Institute of Standards and Technology (NIST) revised its list of approved modes of operation by including AES as a block cipher and adding CTR mode in SP800-38A, Recommendation for Block Cipher modes of Operation. Finally, in January, 2010, NIST added XTS-AES in SP800-38E, Recommendation for Block Cipher modes of Operation: The XTS-AES Mode for Confidentiality on Storage Devices. Other confidentiality modes exist which have not been approved by NIST. For example, CTS is ciphertext stealing mode and available in many popular cryptographic libraries

The block cipher modes ECB, CBC, OFB, CFB, CTR, and XTS provide confidentiality, but they do not protect against accidental modification or malicious tampering. Modification or tampering can be detected with a separate message authentication code such as CBC-MAC, or a digital signature. The cryptographic community recognized the need for dedicated integrity assurances and NIST responded with HMAC, CMAC, and GMAC. HMAC was approved in 2002 as FIPS 198, The Keyed-Hash Message Authentication Code (HMAC), CMAC was released in 2005 under SP800-38B, Recommendation for Block Cipher modes of Operation: The CMAC Mode for Authentication, and GMAC was formalized in 2007 under SP800-38D, Recommendation for Block Cipher modes of Operation: Galois/Counter Mode (GCM) and GMAC

After observing that compositing a confidentiality mode with an authenticity mode could be difficult and error prone, the cryptographic community began to supply modes which combined confidentiality and data integrity into a single cryptographic primitive. The modes are referred to as authenticated encryption, AE or "authenc". Examples of AE modes are CCM (SP800-38C), GCM (SP800-38D), CWC, EAX, IAPM, and OCB

modes of operation are nowadays defined by a number of national and internationally recognized standards bodies. Notable standards organizations include NIST, ISO (with ISO/IEC 10116), the IEC, the IEEE, the national ANSI, and the IETF

Initialization vector (IV)

An initialization vector (IV) or starting variable (SV) is a block of bits that is used by several modes to randomize the encryption and hence to produce distinct ciphertexts even if the same plaintext is encrypted multiple times, without the need for a slower re-keying process

An initialization vector has different security requirements than a key, so the IV usually does not need to be secret. However, in most cases, it is important that an initialization vector is never reused under the same key. For CBC and CFB, reusing an IV leaks some information about the first block of plaintext, and about any common prefix shared by the two messages. For OFB and CTR, reusing an IV completely destroys security. This can be seen because both modes effectively create a bitstream that is XORed with the plaintext, and this bitstream is dependent on the password and IV only. Reusing a bitstream destroys security. In CBC mode, the IV must, in addition, be unpredictable at encryption time; in particular, the (previously) common practice of re-using the last ciphertext block of a message as the IV for the next message is insecure (for example, this method was used by SSL 2.0). If an attacker knows the IV (or the previous block of ciphertext) before he specifies the next plaintext, he can check his guess about plaintext of some block that was encrypted with the same key before (this is known as the TLS CBC IV attack)

Padding

A block cipher works on units of a fixed size (known as a block size), but messages come in a variety of lengths. So some modes (namely ECB and CBC) require that the final block be padded before encryption. Several padding schemes exist. The simplest is to add null bytes to the plaintext to bring its length up to a multiple of the block size, but care must be taken that the original length of the plaintext can be recovered; this is trivial, for example, if the plaintext is a C style string which contains no null bytes except at the end. Slightly more complex is the original DES method, which is to add a single one bit, followed by enough zero bits to fill out the block; if the message ends on a block boundary, a whole padding block will be added. Most sophisticated are CBC-specific schemes such as ciphertext stealing or residual block termination, which do not cause any extra ciphertext, at the expense of some additional complexity. Schneier and Ferguson suggest two possibilities, both simple: append a byte with value 128 (hex 80), followed by as many zero bytes as needed to fill the last block, or pad the last block with n bytes all with value n

CFB, OFB and CTR modes do not require any special measures to handle messages whose lengths are not multiples of the block size, since the modes work by XORing the plaintext with the output of the block cipher. The last partial block of plaintext is XORed with the first few bytes of the last keystream block, producing a final

ciphertext block that is the same size as the final partial plaintext block. This characteristic of stream ciphers makes them suitable for applications that require the encrypted ciphertext data to be the same size as the original plaintext data, and for applications that transmit data in streaming form where it is inconvenient to add padding bytes

Common modes

Many modes of operation have been defined. Some of these are described below

7.77.2.1 Electronic Codebook (ECB)

The simplest of the encryption modes is the Electronic Codebook (ECB) mode. The message is divided into blocks, and each block is encrypted separately

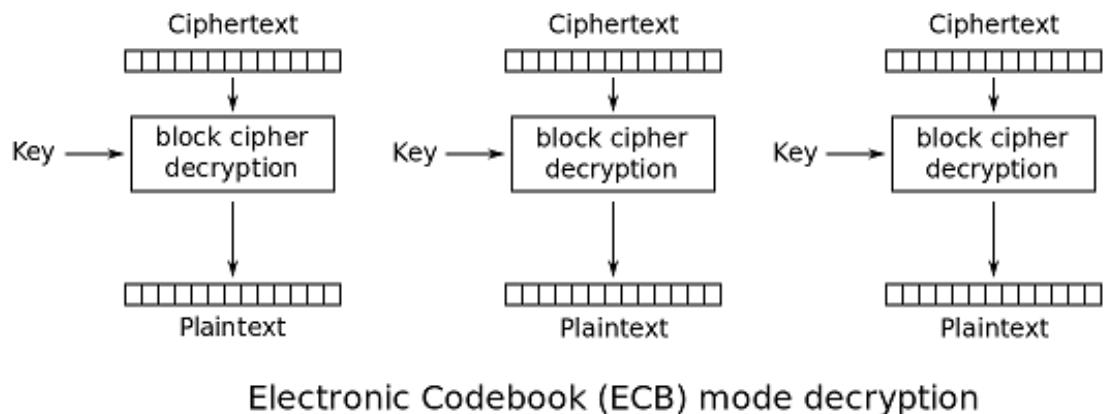
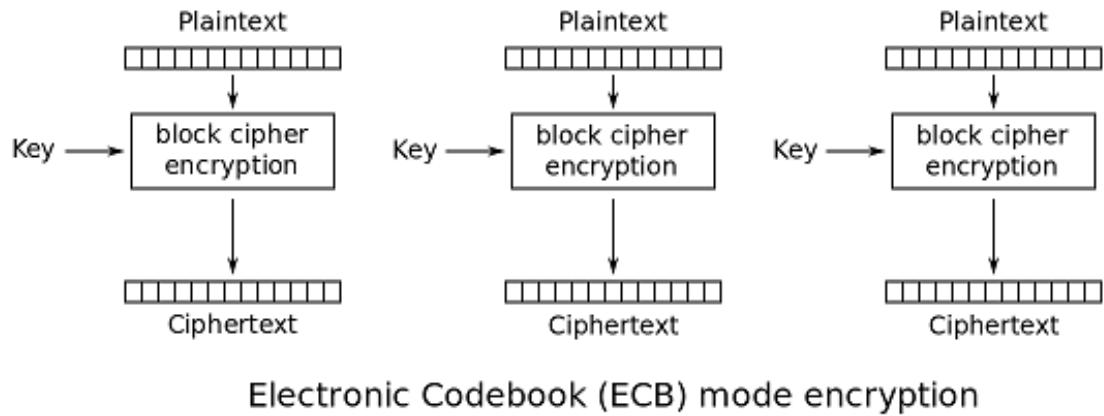


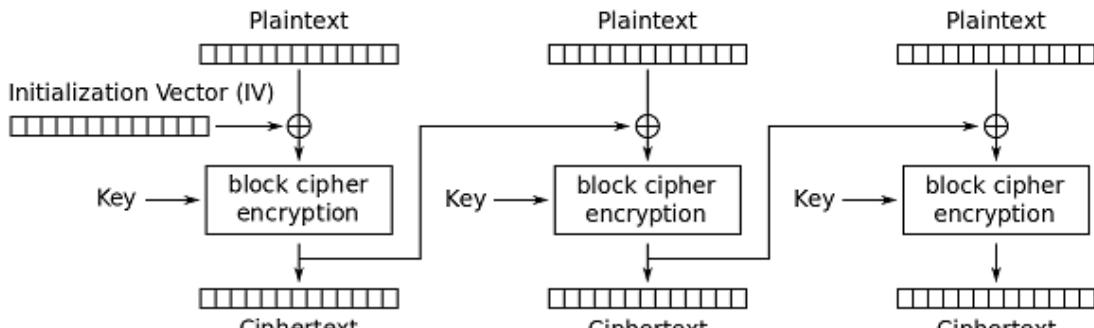
Figure 7.97: Electronic Code Book (ECB) Encryption Mode

The disadvantage of this method is that identical plaintext blocks are encrypted into identical ciphertext blocks; thus, it does not hide data patterns well. In some senses, it doesn't provide serious message confidentiality, and it is not recommended for use in cryptographic protocols at all

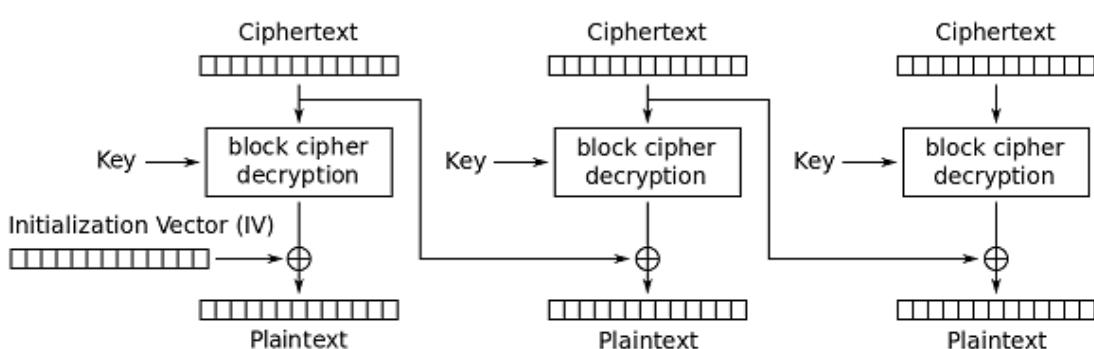
A striking example of the degree to which ECB can leave plaintext data patterns in the ciphertext can be seen when ECB mode is used to encrypt a bitmap image which uses large areas of uniform colour. While the colour of each individual pixel is encrypted, the overall image may still be discerned as the pattern of identically coloured pixels in the original remains in the encrypted version

7.77.2.2 Cipher Block Chaining (CBC)

Ehrsam, Meyer, Smith and Tuchman invented the Cipher Block Chaining (CBC) mode of operation in 1976. In CBC mode, each block of plaintext is XORed with the previous ciphertext block before being encrypted. This way, each ciphertext block depends on all plaintext blocks processed up to that point. To make each message unique, an initialization vector must be used in the first block.



Cipher Block Chaining (CBC) mode encryption



Cipher Block Chaining (CBC) mode decryption

Figure 7.98: Cipher Block Chaining (CBC) Encryption Mode

If the first block has index 1, the mathematical formula for CBC encryption is

$$C_i = E_K(P_i \oplus C_{i-1}), C_0 = IV$$

while the mathematical formula for CBC decryption is

$$P_i = D_K(C_i) \oplus C_{i-1}, C_0 = IV$$

CBC has been the most commonly used mode of operation. Its main drawbacks are that encryption is sequential (i.e., it cannot be parallelized), and that the message must be padded to a multiple of the cipher block size. One way to handle this last issue is through the method known as ciphertext stealing. Note that a one-bit change in a plaintext or IV affects all following ciphertext blocks.

Decrypting with the incorrect IV causes the first block of plaintext to be corrupt but subsequent plaintext blocks will be correct. This is because each block is XORed with the ciphertext of the previous block, not the plaintext, so one does not need to decrypt the previous block before using it as the IV for the decryption of the current one. This means that a plaintext block can be recovered from two adjacent blocks of ciphertext. As a consequence, decryption can be parallelized. Note that a one-bit change to the ciphertext causes complete corruption of the corresponding block of plaintext, and inverts the corresponding bit in the following block of plaintext, but the rest of the blocks remain intact. This peculiarity is exploited in different padding oracle attacks, such as POODLE.

Explicit Initialization Vectors takes advantage of this property by prepending a single random block to the plaintext. Encryption is done as normal, except the IV does not need to be communicated to the decryption routine. Whatever IV decryption uses, only the random block is "corrupted". It can be safely discarded and the rest of the decryption is the original plaintext

7.77.2.3 Counter (CTR)

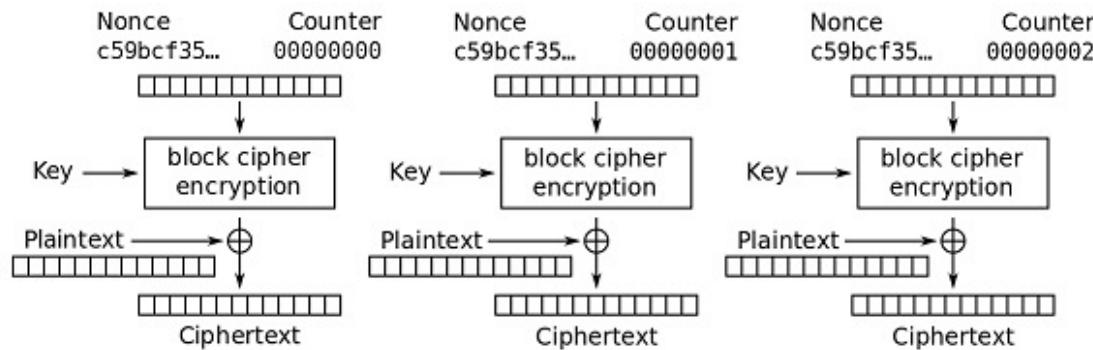
Like OFB, Counter mode turns a block cipher into a stream cipher. It generates the next keystream block by encrypting successive values of a "counter". The counter can be any function which produces a sequence which is guaranteed not to repeat for a long time, although an actual increment-by-one counter is the simplest and most popular. The usage of a simple deterministic input function used to be controversial; critics argued that "deliberately exposing a cryptosystem to a known systematic input represents an unnecessary risk." However, today CTR mode is widely accepted and any problems are considered a weakness of the underlying block cipher, which is expected to be secure regardless of systemic bias in its input. Along with CBC, CTR mode is one of two block cipher jmodes recommended by Niels Ferguson and Bruce Schneier

CTR mode was introduced by Whitfield Diffie and Martin Hellman in 1979

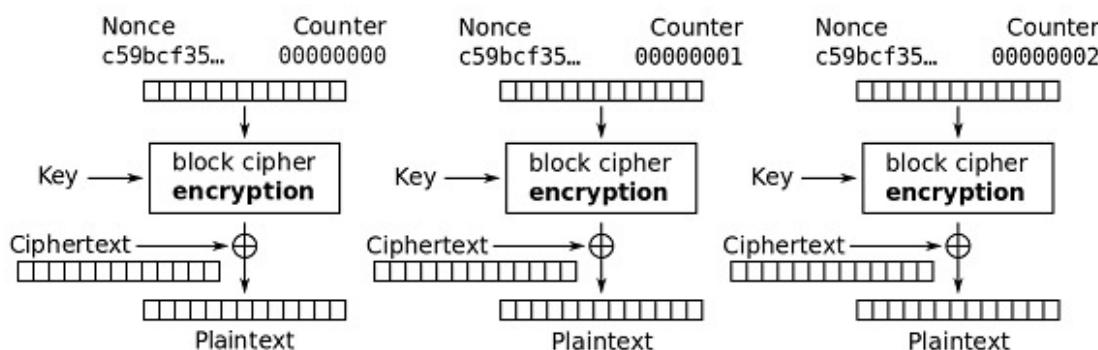
CTR mode has similar characteristics to OFB, but also allows a random access property during decryption. CTR mode is well suited to operate on a multi-processor machine where blocks can be encrypted in parallel. Furthermore, it does not suffer from the short-cycle problem that can affect OFB

If the IV/nonce is random, then they can be combined together with the counter using any lossless operation (concatenation, addition, or XOR) to produce the actual unique counter block for encryption. In case of a non-random nonce (such as a packet counter), the nonce and counter should be concatenated (e.g., storing the nonce in the upper 64 bits and the counter in the lower 64 bits of a 128-bit counter block). Simply adding or XORing the nonce and counter into a single value would break the security under a chosen-plaintext attack in many cases, since the attacker may be able to manipulate the entire IV-counter pair to cause a collision. Once an attacker controls the IV-counter pair and plaintext, XOR of the ciphertext with the known plaintext would yield a value that, when XORed with the ciphertext of the other block sharing the same IV-counter pair, would decrypt that block

Note that the nonce in this diagram is equivalent to the initialization vector (IV) in the other diagrams. However, if the offset/location information is corrupt, it will be impossible to partially recover such data due to the dependence on byte offset



Counter (CTR) mode encryption



Counter (CTR) mode decryption

Figure 7.99: Counter Mode (CTR) Encryption Mode

7.77.2.4 Galois Counter Mode (GCM)

Galois/Counter Mode (GCM) is a mode of operation for symmetric key cryptographic block ciphers that has been widely adopted because of its efficiency and performance. GCM throughput rates for state of the art, high speed communication channels can be achieved with reasonable hardware resources. The operation is an authenticated encryption algorithm designed to provide both data authenticity (integrity) and confidentiality. GCM is defined for block ciphers with a block size of 128 bits. Galois Message Authentication Code (GMAC) is an authentication-only variant of the GCM which can be used as an incremental message authentication code. Both GCM and GMAC can accept initialization vectors of arbitrary length.

Different block cipher modes of operation can have significantly different performance and efficiency characteristics, even when used with the same block cipher. GCM can take full advantage of parallel processing and implementing GCM can make efficient use of an instruction pipeline or a hardware pipeline. In contrast, the cipher block chaining (CBC) mode of operation incurs significant pipeline stalls that hamper its efficiency and performance.

Basic operation

In the normal counter mode, blocks are numbered sequentially, and then this block number is encrypted with a block cipher E, usually AES. The result of this encryption is then xored with the plain text to produce a cipher text. Like all counter modes, this is essentially a stream cipher, and so it is essential that a different initialization vector is used for each stream that is encrypted.

The Galois Mult function then combines the ciphertext with an authentication code in order to produce an authentication tag that can be used to verify the integrity of the data. The encrypted text then contains the IV, cipher text, and authentication code. It therefore has similar security properties to a HMAC.

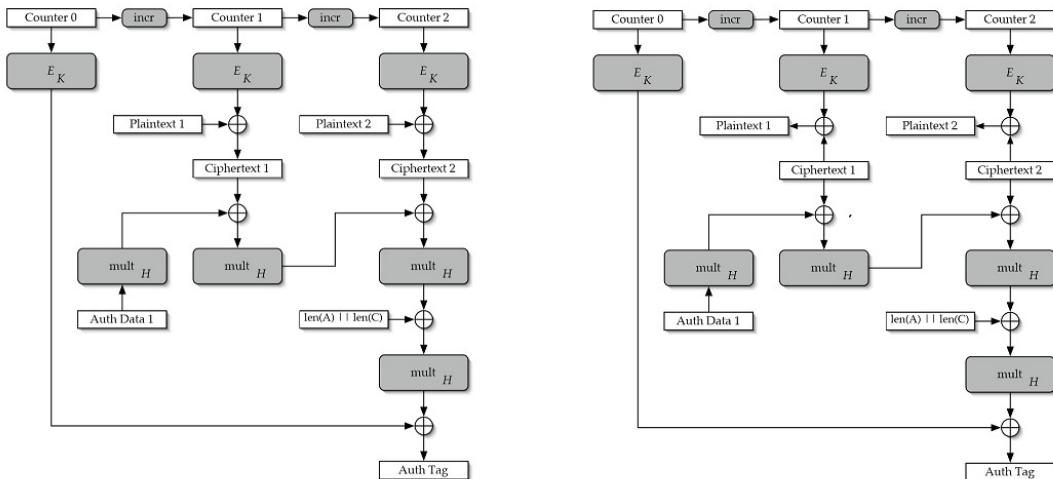


Figure 1: The authenticated encryption operation. For simplicity, a case with only a single block of additional authenticated data (labeled Auth Data 1) and two blocks of plaintext is shown. Here E_K denotes the block cipher encryption using the key K , mult_H denotes multiplication in $GF(2^{128})$ by the hash key H , and incr denotes the counter increment function.

Figure 2: The authenticated decryption operation, showing the same case as in Figure 1.

Figure 7.100: Galios Field Counter Mode (GCM) Encryption Mode

Mathematical basis

GCM combines the well-known counter mode of encryption with the new Galois mode of authentication. The key feature is that the Galois field multiplication used for authentication can be easily computed in parallel. This option permits higher throughput than the authentication algorithms, like CBC, that use chaining modes. The $GF(2^{128})$ field used is defined by the polynomial

$$x^{128} + x^7 + x^2 + x + 1$$

The authentication tag is constructed by feeding blocks of data into the GHASH function and encrypting the result. This GHASH function is defined by

$$GHASH(H, A, C) = X_{m+n+1}$$

where H is the Hash Key, a string of 128 zero bits encrypted using the block cipher, A is data which is only authenticated (not encrypted), C is the ciphertext, m is the number of 128 bit blocks in A , n is the number of 128 bit blocks in C (the final blocks of A and C need not be exactly 128 bits), and the variable X_i for $i = 0, \dots, m + n + 1$ is defined as

$$X_i = \begin{cases} 0 & \text{for } i = 0 \\ (X_{i-1} \oplus A_i) \cdot H & \text{for } i = 1, \dots, m - 1 \\ (X_{m-1} \oplus (A_m^* \| 0^{128-v})) \cdot H & \text{for } i = m \\ (X_{i-1} \oplus C_{i-m}) \cdot H & \text{for } i = m + 1, \dots, m + n - 1 \\ (X_{m+n-1} \oplus (C_n^* \| 0^{128-u})) \cdot H & \text{for } i = m + n \\ (X_{m+n} \oplus (\text{len}(A) \| \text{len}(C))) \cdot H & \text{for } i = m + n + 1 \end{cases}$$

Figure 7.101: GCM Mathematical Computation

where v is the bit length of the final block of A , u is the bit length of the final block of C ; $\|$ denotes concatenation of bit strings, and $\text{len}(A)$ and $\text{len}(C)$ are the 64-bit representations of the bit lengths of A and C , respectively. Note that this is an iterative algorithm: each X_i depends on X_{i-1} and only the final X_i is retained as output

GCM mode was designed by John Viega and David A. McGrew as an improvement to Carter–Wegman Counter CWC mode

In November 2007, NIST announced the release of NIST Special Publication 800-38D Recommendation for Block Cipher modes of Operation: Galois/Counter Mode (GCM) and GMAC making GCM and GMAC official standards

Use

GCM mode is used in the IEEE 802.1AE (MACsec) Ethernet security, IEEE 802.11ad (also known as WiGig), ANSI (INCITS) Fibre Channel Security Protocols (FC-SP), IEEE P1619.1 tape storage, IETF IPsec standards, SSH and TLS 1.2. AES-GCM is included in the NSA Suite B Cryptography

Performance

GCM is ideal for protecting packetized data because it has minimum latency and minimum operation overhead

GCM requires one block cipher operation and one 128-bit multiplication in the Galois field per each block (128 bit) of encrypted and authenticated data. The block cipher operations are easily pipelined or parallelized; the multiplication operations are easily pipelined and can be parallelized with some modest effort (either by parallelizing the actual operation, by adapting Horner's method as described in the original NIST submission, or both)

Intel has added the PCLMULQDQ instruction, highlighting its use for GCM. This instruction enables fast multiplication over GF(2ⁿ), and can be used with any field representation

Impressive performance results have been published for GCM on a number of platforms. Käasper and Schwabe described a "Faster and Timing-Attack Resistant AES-GCM" that achieves 10.68 cycles per byte AES-GCM authenticated encryption on 64-bit Intel processors. Dai et al. report 3.5 cycles per byte for the same algorithm when using Intel's AES-NI and PCLMULQDQ instructions. Shay Gueron and Vlad Krasnov achieved 2.47 cycles per byte on the 3rd generation Intel processors. Appropriate patches were prepared for the OpenSSL and NSS libraries

When both authentication and encryption need to be performed on a message, a software implementation can achieve speed gains by overlapping the execution of those operations. Performance is increased by exploiting instruction level parallelism by interleaving operations. This process is called function stitching, and while in principle it can be applied to any combination of cryptographic algorithms, GCM is especially suitable. Manley and Gregg show the ease of optimizing when using function-stitching with GCM. They present a program generator that takes an annotated C version of a cryptographic algorithm and generates code that runs well on the target processor

7.77.2.5 Counter with CBC-MAC (CCM)

CCM mode (Counter with CBC-MAC) is a mode of operation for cryptographic block ciphers. It is an authenticated encryption algorithm designed to provide both authentication and confidentiality. CCM mode is only defined for block ciphers with a block length of 128 bits. In RFC 3610, it is defined for use with AES

The Initialization Vector (IV) of CCM must be carefully chosen to never be used more than once for a given key. This is because CCM is a derivation of CTR mode and the latter is effectively a stream cipher

Encryption and authentication

As the name suggests, CCM mode combines the well known CBC-MAC with the well known counter mode of encryption. These two primitives are applied in an "authenticate-then-encrypt" manner, that is, CBC-MAC is first computed on the message to obtain a tag t ; the message and the tag are then encrypted using counter mode. One key insight is that the same encryption key can be used for both, provided that the counter values used in the encryption do not collide with the (pre-)initialization vector used in the authentication. A proof of security exists for this combination, based on the security of the underlying block cipher. The proof also applies to a generalization of CCM for any size block cipher, and for any size cryptographically strong pseudo-random function (since in both counter mode and CBC-MAC, the block cipher is only ever used in one direction)

CCM mode was designed by Russ Housley, Doug Whiting and Niels Ferguson. At the time CCM mode was developed, Russ Housley was employed by RSA Laboratories

A minor variation of the CCM, called CCM*, is used in the ZigBee standard. CCM* includes all of the features of CCM and additionally offers encryption-only and integrity-only capabilities

Performance

CCM requires two block cipher encryption operations per each block of an encrypted-and-authenticated message, and one encryption per each block of associated authenticated data

According to Crypto++ benchmarks, AES CCM requires 28.6 cycles per byte on an Intel Core 2 processor in 32-bit

mode

7.77.2.6 The AES-XCBC-MAC-96 Algorithm and Its Use With IPsec

A Message Authentication Code (MAC) is a key-dependent one way hash function. One popular way to construct a MAC algorithm is to use a block cipher in conjunction with the Cipher-Block-Chaining (CBC) mode of operation. The classic CBC-MAC algorithm, while secure for messages of a pre-selected fixed length, has been shown to be insecure across messages of varying lengths such as the type found in typical IP datagrams. This memo specifies the use of AES in CBC mode with a set of extensions to overcome this limitation. This new algorithm is named AES-XCBC-MAC-96

AES-XCBC-MAC-96

[AES] describes the underlying AES algorithm, while [CBC-MAC-1] and [XCBC-MAC-1] describe the AES-XCBC--MAC algorithm

The AES-XCBC-MAC-96 algorithm is a variant of the basic CBC-MAC with obligatory 10* padding; however, AES-XCBC-MAC-96 is secure for messages of arbitrary length. The AES-XCBC-MAC-96 calculations require numerous encryption operations; this encryption MUST be accomplished using AES with a 128-bit key. Given a 128-bit secret key K, AES-XCBC-MAC-96 is calculated as follows for a message M that consists of n blocks, M[1] ... M[n], in which the blocksize of blocks M[1] ... M[n-1] is 128 bits and the blocksize of block M[n] is between 1 and 128 bits:

- Derive 3 128-bit keys (K1, K2 and K3) from the 128-bit secret key K, as follows:
 - K1 = 0x010101010101010101010101010101 encrypted with Key K
 - K2 = 0x020202020202020202020202020202 encrypted with Key K
 - K3 = 0x030303030303030303030303030303 encrypted with Key K
 - Define $E[0] = 0x00000000000000000000000000000000$
 - For each block $M[i]$, where $i = 1 \dots n-1$:
 - XOR $M[i]$ with $E[i-1]$, then encrypt the result with Key K1, yielding $E[i]$
 - For block $M[n]$:
 - If the blocksize of $M[n]$ is 128 bits:
 - XOR $M[n]$ with $E[n-1]$ and Key K2, then encrypt the result with Key K1, yielding $E[n]$
 - If the blocksize of $M[n]$ is less than 128 bits:
 - Pad $M[n]$ with a single "1" bit, followed by the number of "0" bits (possibly none) required to increase $M[n]$'s blocksize to 128 bits
 - XOR $M[n]$ with $E[n-1]$ and Key K3, then encrypt the result with Key K1, yielding $E[n]$
 - The authenticator value is the leftmost 96 bits of the 128-bit $E[n]$

NOTE1: If M is the empty string, pad and encrypt as in (4)(b) to create M[1] and E[1]. This will never be the case for ESP or AH, but is included for completeness sake

NOTE2: [CBC-MAC-1] defines K1 as follows:

- K1 = Constant1A encrypted with Key K | Constant1B encrypted with Key K

However, the second encryption operation is only needed for AES-XCBC-MAC with keys greater than 128 bits; thus, it is not included in the definition of AES-XCBC-MAC-96.

AES-XCBC-MAC-96 verification is performed as follows:

- Upon receipt of the AES-XCBC-MAC-96 authenticator, the entire 128-bit value is computed and the first 96 bits are compared to the value stored in the authenticator field.

Keying Material

AES-XCBC-MAC-96 is a secret key algorithm. For use with either ESP or AH a fixed key length of 128-bits MUST be supported. Key lengths other than 128-bits MUST NOT be supported (i.e., only 128-bit keys are to be used by AES-XCBC-MAC-96).

AES-XCBC-MAC-96 actually requires 384 bits of keying material (128 bits for the AES keysize + 2 times the block-size). This keying material can either be provided through the key generation mechanism or it can be generated from a single 128-bit key. The latter approach has been selected for AES-XCBC-MAC-96, since it is analogous to other authenticators used within IPsec. The reason AES-XCBC-MAC-96 uses 3 keys is so the length of the input stream does not need to be known in advance. This may be useful for systems that do one-pass assembly of large packets

A strong pseudo-random function MUST be used to generate the required 128-bit key. This key, along with the 3 derived keys (K_1 , K_2 and K_3), should be used for no purposes other than those specified in the algorithm. In particular, they should not be used as keys in another cryptographic setting. Such abuses will invalidate the security of the authentication algorithm

At the time of this writing there are no specified weak keys for use with AES-XCBC-MAC-96. This does not mean to imply that weak keys do not exist. If, at some point, a set of weak keys for AES-XCBC-MAC-96 are identified, the use of these weak keys MUST be rejected followed by a request for replacement keys or a newly negotiated Security Association

[ARCH] describes the general mechanism for obtaining keying material when multiple keys are required for a single SA (e.g., when an ESP SA requires a key for confidentiality and a key for authentication)

In order to provide data origin authentication, the key distribution mechanism must ensure that unique keys are allocated and that they are distributed only to the parties participating in the communication

Current attacks do not necessitate a specific recommended frequency for key changes. However, periodic key refreshment is a fundamental security practice that helps against potential weaknesses of the function and the keys, reduces the information available to a cryptanalyst, and limits the damage resulting from a compromised key

Padding

AES-XCBC-MAC-96 operates on 128-bit blocks of data. Padding requirements are specified in [CBC-MAC-1] and are part of the Xcbc algorithm. If you build AES-XCBC-MAC-96 according to [CBC-MAC-1] you do not need to add any additional padding as far as AES-XCBC-MAC-96 is concerned. With regard to "implicit packet padding" as defined in [AH], no implicit packet padding is required

Truncation

AES-XCBC-MAC produces a 128-bit authenticator value. AES-XCBC-MAC-96 is derived by truncating this 128-bit value as described in [HMAC] and verified in [XCBC-MAC-2]. For use with either ESP or AH, a truncated value using the first 96 bits MUST be supported. Upon sending, the truncated value is stored within the authenticator field. Upon receipt, the entire 128-bit value is computed and the first 96 bits are compared to the value stored in the authenticator field. No other authenticator value lengths are supported by AES-XCBC-MAC-96

The length of 96 bits was selected because it is the default authenticator length as specified in [AH] and meets the security requirements described in [XCBC-MAC-2]

Interaction with the ESP Cipher Mechanism

As of this writing, there are no known issues which preclude the use of AES-XCBC-MAC-96 with any specific cipher algorithm

Performance

For any CBC MAC variant, the major computational effort is expended in computing the underlying block cipher. This algorithm uses a minimum number of AES invocations, one for each block of the message or fraction thereof, resulting in performance equivalent to classic CBC-MAC. The key expansion requires 3 additional AES encryption operations, but these can be performed once in advance for each secret key

7.77.2.7 The CMAC Mode for Authentication

See NIST800-38b

Block Cipher

The CMAC algorithm depends on the choice of an underlying symmetric key block cipher. The CMAC algorithm is thus a mode of operation (a mode, for short) of the block cipher. The CMAC key is the block cipher key (the key, for short). For any given key, the underlying block cipher of the mode consists of two functions that are inverses of each other. The choice of the block cipher includes the designation of one of the two functions of the block cipher as the forward function/transformation, and the other as the inverse function, as in the specifications of the AES algorithm and TDEA in Ref. [3] and Ref. [10], respectively. The CMAC mode does not employ the inverse function.

The forward cipher function is a permutation on bit strings of a fixed length; the strings are called blocks. The bit length of a block is denoted b , and the length of a block is called the block size. For the AES algorithm, $b = 128$; for TDEA, $b = 64$. The key is denoted K , and the resulting forward cipher function of the block cipher is denoted $CIPH_K$.

The underlying block cipher shall be approved, and the key shall be generated uniformly at random, or close to uniformly at random, i.e., so that each possible key is (nearly) equally likely to be generated. The key shall be secret and shall be used exclusively for the CMAC mode of the chosen block cipher. The message span of the key is discussed in Appendix B. To fulfill the requirements on the key, the key should be established among the parties to the information within an approved key management structure; the details of the establishment and management of keys are outside the scope of this Recommendation.

Subkeys

The block cipher key is used to derive two additional secret values, called the subkeys, denoted K_1 and K_2 . The length of each subkey is the block size. The subkeys are fixed for any invocation of CMAC with the given key. Consequently, the subkeys may be precomputed and stored with the key for repeated use; alternatively, the subkeys may be computed anew for each invocation.

Any intermediate value in the computation of the subkey, in particular, $CIPH_K(0b)$, shall also be secret. This requirement precludes the system in which CMAC is implemented from using this intermediate value publicly for some other purpose, for example, as an unpredictable value or as an integrity check value on the key. One of the elements of the subkey generation process is a bit string, denoted R_b , that is completely determined by the number of bits in a block. In particular, for the two block sizes of the currently approved block ciphers, $R_{128} = 0^{120}10000111$, and $R_{64} = 0^{59}11011$.

In general, R_b is a representation of a certain irreducible binary polynomial of degree b , namely, the lexicographically first among all such polynomials with the minimum possible number of nonzero terms. If this polynomial is expressed as $u^b + c_{b-1}u^{b-1} + \dots + c_2u^2 + c_1u + c_0$, where the coefficients $c_{b-1}, c_{b-2}, \dots, c_2, c_1, c_0$ are either 0 or 1, then R_b is the bit string $c_{b-1}c_{b-2}\dots c_2c_1c_0$.

MAC Generation and Verification

As for any MAC algorithm, an authorized party applies the MAC generation process to the data to be authenticated to produce a MAC for the data. Subsequently, any authorized party can apply the verification process to the received data and the received MAC. Successful verification provides assurance of data authenticity, as discussed in Appendix A, and, hence, of integrity.

Input and Output Data

For a given block cipher and key, the input to the MAC generation function is a bit string called the message, denoted M . The bit length of M is denoted $Mlen$. The value of $Mlen$ is not an essential input for the MAC generation algorithm if the implementation has some other means of identifying the last block in the partition of the message, as discussed in Sec. 6.2. Thus, in such a case, the computation of the MAC may begin “on-line” before the entire message is available. In principle, there is no restriction on the lengths of messages. In practice, however, the system in which CMAC is implemented may restrict the length of the input messages to the MAC generation function.

The output of the MAC generation function is a bit string called the MAC, denoted T . The length of T , denoted $Tlen$, is a parameter that shall be fixed for all invocations of CMAC with the given key. The requirements for the selection of $Tlen$ are given in Appendix A.

CMAC Specification

Subkey generation, MAC generation, and MAC verification are specified in Sections 6.1, 6.2, and 6.3 below. The specifications include the inputs, the outputs, a suggested notation for the function, the steps, and a summary; for MAC generation, a diagram is also given. The inputs that are typically fixed across many invocations of CMAC are called the prerequisites. The prerequisites and the other inputs shall meet the requirements in Sec. 5. The suggested notation does not include the block cipher.

Subkey Generation

The following is a specification of the subkey generation process of CMAC: Prerequisites:

- block cipher CIPH with block size b
- key K

Output:

- subkeys K1, K2

Suggested Notation:

- $\text{SUBK}(K)$

Steps:

1. Let $L = \text{CIPH}_K(0^b)$
2. If $\text{MSB}_1(L) = 0$, then $K1 = L << 1$; Else $K1 = (L << 1) \oplus R_b$; see Sec. 5.3 for the definition of R_b
3. If $\text{MSB}_1(K1) = 0$, then $K2 = K1 << 1$; Else $K2 = (K1 << 1) \oplus R_b$
4. Return K1, K2

In Step 1, the block cipher is applied to the block that consists entirely of '0' bits. In Step 2, the first subkey is derived from the resulting string by a left shift of one bit, and, conditionally, by XORing a constant that depends on the block size. In Step 3, the second subkey is derived in the same manner from the first subkey.¹ As discussed in Sec. 5.3, any intermediate value in the computation of the subkey, in particular, $\text{CIPH}_K(0^b)$, shall be secret

MAC Generation

The following is a specification of the MAC generation process of CMAC: Prerequisites:

- block cipher CIPH with block size b
- key K
- MAC length parameter Tlen

Input:

- message M of bit length Mlen.

Output:

- MAC T of bit length Tlen

Suggested Notation:

- $\text{CMAC}(K, M, Tlen)$ or, if Tlen is understood from the context, $\text{CMAC}(K, M)$

Steps:

1. Apply the subkey generation process in Sec. 6.1 to K to produce K1 and K2
2. If $Mlen = 0$, let $n = 1$; else, let $n = [Mlen/\text{blk_size}]$

3. Let $M_1, M_2, \dots, M_{n-1}, M_n$ denote the unique sequence of bit strings such that $M = M_1 || M_2 || \dots || M_{n-1} || M_n*$, where M_1, M_2, \dots, M_{n-1} are complete blocks
4. If M_n* is a complete block, let $M_n = K1 \oplus M_n*$; else, let $M_n = K2 \oplus (M_n* || 10j)$, where $j = \text{nb_Mlen-1}$
5. Let $C_0 = 0^b$
6. For $i = 1$ to n , let $C_i = \text{CIPH}_K(C_{i-1} \oplus M_i)$
7. Let $T = \text{MSB}_{T\text{len}}(C_n)$
8. Return T

In Step 1, the subkeys are generated from the key. In Steps 2–4, the input message is formatted into a sequence of complete blocks in which the final block has been masked by a subkey. There are two cases:

- If the message length is a positive multiple of the block size, then the message is partitioned into complete blocks. The final block is masked with the first subkey; in other words, the final block in the partition is replaced with the exclusive-OR of the final block with the first subkey. The resulting sequence of blocks is the formatted message
- If the message length is not a positive multiple of the block size, then the message is partitioned into complete blocks to the greatest extent possible, i.e., into a sequence of complete blocks followed by a final bit string whose length is less than the block size. A padding string is appended to this final bit string, in particular, a single ‘1’ bit followed by the minimum number of ‘0’ bits, possibly none, that are necessary to form a complete block. The complete final block is masked, as described in the previous bullet, with the second subkey. The resulting sequence of blocks is the formatted message

In Steps 5 and 6, the cipher block chaining (CBC) technique, with the zero block as the initialization vector, is applied to the formatted message. In Steps 7 and 8, the final CBC output block is truncated according to the MAC length parameter that is associated with the key, and the result is returned as the MAC

Equivalent sets of steps, i.e., procedures that yield the correct output from the same input, are permitted. For example, it is not necessary to complete the formatting of the entire message (Steps 3 and 4) prior to the cipher block chaining (Steps 5 and 6). Instead, the iterations of Step 5 may be executed “on the fly,” i.e., on each successive block of the message as soon as it is available for processing. Step 4 may be delayed until the final bit string in the partition is available; the appropriate case, and value of j , if necessary, can be determined from the length of the final bit string. In such an implementation, the determination in Step 2 of the total number of blocks in the formatted message may be omitted, assuming that the implementation has another way to identify the final string in the partition.

Similarly, the subkeys need not be computed anew for each invocation of CMAC with a given key; instead, they may be precomputed and stored along with the key as algorithm inputs.

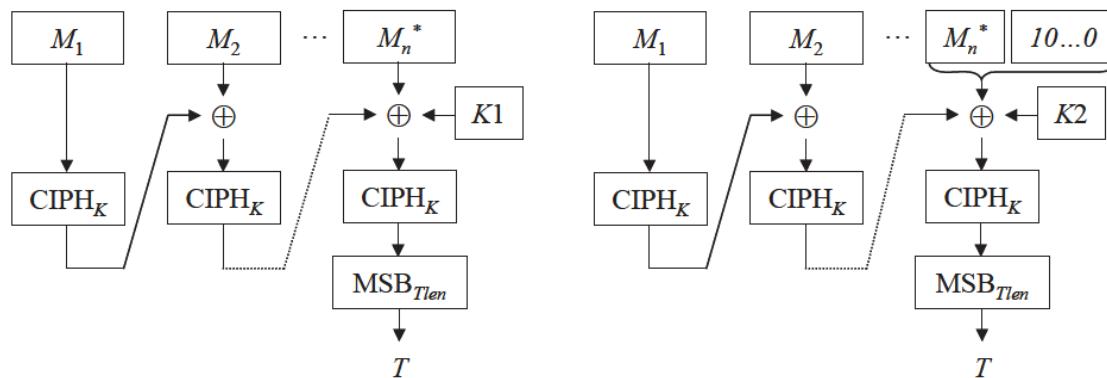


Figure 1: Illustration of the two cases of MAC Generation.

Figure 7.102: Counter Mode MAC Generation

The two cases of MAC Generation are illustrated in Figure 1 above. On the left is the case where the message length is a positive multiple of the block size; on the right is the case where the message length is not a positive multiple of the block length

MAC Verification

The following is a specification of the MAC verification process of CMAC: Prerequisites:

- block cipher CIPH with block size b
- key K
- subkeys K1, K2
- MAC length Tlen

Input:

- message M of bit length Mlen
- received MAC T'

Output:

- VALID or INVALID

Suggested Notation:

- VER(K, M, T')

Steps:

1. Apply the MAC generation process in Sec. 6.2 to M to produce T
2. If T = T', return VALID; else, return INVALID

In Step 1, the MAC generation process in Sec. 6.2 is applied to the message, and, in Step 2, the resulting MAC is compared with the received MAC to determine its validity

For API Documentation:

See Also

[ProtocolPP::aria](#)
[ProtocolPP::sm4](#)
[ProtocolPP::sm3](#)
[ProtocolPP::jsnow3g](#)
[ProtocolPP::jzuc](#)

For Additional Documentation:

See Also

[aria](#)
[sm4](#)
[sm3](#)
[jsnow3g](#)
[jzuc](#)

Protocol++ modified by : John Peter Greninger © John Peter Greninger 2015-2019 All Rights Reserved

All copyrights and trademarks are the property of their respective owners

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer
- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution
- Any and all modifications must be returned to John Peter Greninger at GitHub.com <https://github.com/jpgreninger/protocolpp> for evaluation. Inclusion of modifications in the source code shall be determined solely by John Peter Greninger. Failure to provide modifications shall render this license NULL and VOID and revoke any rights to use of Protocol++
- Commercial use requires a fee-based license obtainable at www.protocolpp.com
- Academic use requires written and notarized permission from John Peter and Sheila Rocha Greninger
- US Copyrights at <https://www.copyright.gov/>
 - TXu002059872 (Version 1.0.0)
 - TXu002066632 (Version 1.2.7)
 - TXu002082674 (Version 1.4.0)
 - TXu002097880 (Version 2.0.0)

The names of its contributors may not be used to endorse or promote products derived from this software without specific prior written permission

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE

The documentation for this class was generated from the following file:

- [include/jmodes.h](#)

7.78 jpacket Class Reference

```
#include "include/jpacket.h"
```

7.78.1 Detailed Description

7.78.2 Packet class for Protocol++

For API Documentation:

See Also

[ProtocolPP::jarray](#)

For Additional Documentation:**See Also**

[jarray](#)

Protocol++ written by : John Peter Greninger © John Peter Greninger 2015-2019 All Rights Reserved

All copyrights and trademarks are the property of their respective owners

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer
- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution
- Any and all modifications must be returned to John Peter Greninger at GitHub.com <https://github.com/jpgreninger/protocolpp> for evaluation. Inclusion of modifications in the source code shall be determined solely by John Peter Greninger. Failure to provide modifications shall render this license NULL and VOID and revoke any rights to use of Protocol++
- Commercial use requires a fee-based license obtainable at www.protocolpp.com
- Academic use requires written and notarized permission from John Peter and Sheila Rocha Greninger
- US Copyrights at <https://www.copyright.gov/>
 - TXu002059872 (Version 1.0.0)
 - TXu002066632 (Version 1.2.7)
 - TXu002082674 (Version 1.4.0)
 - TXu002097880 (Version 2.0.0)

The names of its contributors may not be used to endorse or promote products derived from this software without specific prior written permission

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE

The documentation for this class was generated from the following file:

- [include/jpacket.h](#)

7.79 ProtocolPP::jpacket Class Reference

```
#include <jpacket.h>
```

Public Member Functions

- `jpacket (std::string stream, std::string name, std::string prev, std::shared_ptr< jarray< uint8_t >> &input, uint32_t status=0)`
- `jpacket (std::string stream, std::string name, std::string prev, std::shared_ptr< jarray< uint8_t >> &input, std::shared_ptr< jarray< uint8_t >> &expect, unsigned int outlen=0, uint32_t status=0)`
- `virtual ~jpacket ()`
Standard deconstructor.
- `void set_stream (std::string &name)`
- `void set_name (std::string &name)`
- `void set_previous (std::string &name)`
- `void set_outaddr (uint64_t address)`
- `void set_outlen (unsigned int length)`
- `void set_status (uint32_t status)`
- `void set_input (std::shared_ptr< jarray< uint8_t >> &input)`
- `void set_output (std::shared_ptr< jarray< uint8_t >> &output)`
- `void set_expect (std::shared_ptr< jarray< uint8_t >> &expect)`
- `std::string get_stream ()`
- `std::string get_name ()`
- `std::string get_previous ()`
- `unsigned int get_inlen ()`
- `uint64_t get_outaddr ()`
- `unsigned int get_outplen ()`
- `unsigned int get_outlen ()`
- `unsigned int get_explen ()`
- `uint32_t get_status ()`
- `void get_input (std::shared_ptr< jarray< uint8_t >> &input)`
- `void get_output (std::shared_ptr< jarray< uint8_t >> &output)`
- `void get_expect (std::shared_ptr< jarray< uint8_t >> &expect)`

7.79.1 Constructor & Destructor Documentation

7.79.1.1 `ProtocolIPP::jpacket::jpacket (std::string stream, std::string name, std::string prev, std::shared_ptr< jarray< uint8_t >> & input, uint32_t status = 0) [inline]`

Constructor for jpacket

Parameters

<code>stream</code>	- name of the stream this packet belongs to
<code>name</code>	- unique name of the packet
<code>prev</code>	- name of the previous packet (may be 'start')
<code>input</code>	- shared pointer to input data
<code>status</code>	- expected status of the packet

7.79.1.2 `ProtocolIPP::jpacket::jpacket (std::string stream, std::string name, std::string prev, std::shared_ptr< jarray< uint8_t >> & input, std::shared_ptr< jarray< uint8_t >> & expect, unsigned int outlen = 0, uint32_t status = 0) [inline]`

Constructor for jpacket

Parameters

<i>stream</i>	- name of the stream this packet belongs to
<i>name</i>	- unique name of the packet
<i>prev</i>	- name of the previous packet (may be 'start')
<i>input</i>	- shared pointer to input data
<i>expect</i>	- shared pointer to expected data
<i>outlen</i>	- length of the output data
<i>status</i>	- expected status of the packet

7.79.1.3 **virtual ProtocolPP::jpacket::~jpacket() [inline], [virtual]**

Standard deconstructor.

7.79.2 Member Function Documentation

7.79.2.1 **void ProtocolPP::jpacket::get_expect(std::shared_ptr<jarray<uint8_t>> &expect) [inline]**

Get the expected data for this packet

Returns

- expected data of this packet

7.79.2.2 **unsigned int ProtocolPP::jpacket::get_explen() [inline]**

Get the expected length of this packet

Returns

- expected length

7.79.2.3 **unsigned int ProtocolPP::jpacket::get_inlen() [inline]**

Get the input length of this packet

Returns

- input length

7.79.2.4 **void ProtocolPP::jpacket::get_input(std::shared_ptr<jarray<uint8_t>> &input) [inline]**

Get the input data for this packet

Returns

- input data of this packet

7.79.2.5 **std::string ProtocolPP::jpacket::get_name() [inline]**

Get the name for this packet

Returns

- string containing name of the packet

7.79.2.6 `uint64_t ProtocolPP::jpacket::get_outaddr() [inline]`

Get the output address of this packet

Returns

- output address

7.79.2.7 `unsigned int ProtocolPP::jpacket::get_outlen() [inline]`

Get the output length of this packet

Returns

- output length

7.79.2.8 `void ProtocolPP::jpacket::get_output(std::shared_ptr<jarray< uint8_t >> &output) [inline]`

Get the output data for this packet

Returns

- output data of this packet

7.79.2.9 `unsigned int ProtocolPP::jpacket::get_outplen() [inline]`

Get the length of output for this packet

Returns

- output length

7.79.2.10 `std::string ProtocolPP::jpacket::get_previous() [inline]`

Get the name of the previous packet

Returns

- string containing name of the previous packet

7.79.2.11 `uint32_t ProtocolPP::jpacket::get_status() [inline]`

Get the status of this packet

Returns

- status of this packet

7.79.2.12 `std::string ProtocolPP::jpacket::get_stream() [inline]`

Get the stream name for this packet

Returns

- string containing name of the stream

7.79.2.13 void ProtocolPP::jpacket::set_expect(std::shared_ptr<jarray< uint8_t >> & expect) [inline]

Set the expected data for this packet

Parameters

<i>expect</i>	- expected data for this packet
---------------	---------------------------------

7.79.2.14 void ProtocolPP::jpacket::set_input (std::shared_ptr<jarray<uint8_t>> &*input*) [inline]

Set the input data for this packet

Parameters

<i>input</i>	- input data for this packet
--------------	------------------------------

7.79.2.15 void ProtocolPP::jpacket::set_name (std::string &*name*) [inline]

Set the name for this packet

Parameters

<i>name</i>	- name of the packet
-------------	----------------------

7.79.2.16 void ProtocolPP::jpacket::set_outaddr (uint64_t *address*) [inline]

Set the output address

Parameters

<i>address</i>	- mallocoed address to write output data
----------------	--

7.79.2.17 void ProtocolPP::jpacket::set_outlen (unsigned int *length*) [inline]

Set the output length

Parameters

<i>length</i>	- length of the output
---------------	------------------------

7.79.2.18 void ProtocolPP::jpacket::set_output (std::shared_ptr<jarray<uint8_t>> &*output*) [inline]

Set the output data for this packet

Parameters

<i>output</i>	- output data of the packet
---------------	-----------------------------

7.79.2.19 void ProtocolPP::jpacket::set_previous (std::string &*name*) [inline]

Set the name for the packet previous to this one

Parameters

<i>name</i>	- name of the previous packet
-------------	-------------------------------

7.79.2.20 void ProtocolPP::jpacket::set_status(uint32_t *status*) [inline]

Set the status for this packet

Parameters

<i>status</i>	- status of the packet
---------------	------------------------

7.79.2.21 void ProtocolPP::jpacket::set_stream (std::string & *name*) [inline]

Set the stream name for this packet

Parameters

<i>name</i>	- name of the stream
-------------	----------------------

The documentation for this class was generated from the following file:

- include/jpacket.h

7.80 ProtocolPP::jpoly1305 Class Reference

```
#include <jpoly1305.h>
```

Public Member Functions

- [jpoly1305](#) (const unsigned char *key, unsigned int length)
- [jpoly1305](#) (unsigned char *ctx)
- virtual [~jpoly1305](#) ()

Standard Deconstructor for JPOLY1305 algorithm.

- void [ProcessData](#) (const unsigned char *input, unsigned int length)
- void [context](#) (unsigned char *ctx, unsigned int length=144)
- void [result](#) (unsigned char *mac, unsigned int length=16)

7.80.1 Constructor & Destructor Documentation

7.80.1.1 ProtocolPP::jpoly1305::jpoly1305 (const unsigned char * *key*, unsigned int *length*)

Constructor for JPOLY1305 algorithm

Parameters

<i>key</i>	- initialization key
<i>length</i>	- length of the key in bytes (32)

7.80.1.2 ProtocolPP::jpoly1305::jpoly1305 (unsigned char * *ctx*)

Constructor for JPOLY1305 algorithm with context

Parameters

<i>ctx</i>	- context from previous session
------------	---------------------------------

7.80.1.3 virtual ProtocolPP::jpoly1305::~jpoly1305 () [inline], [virtual]

Standard Deconstructor for JPOLY1305 algorithm.

7.80.2 Member Function Documentation

7.80.2.1 void ProtocolPP::jpoly1305::context (*unsigned char * ctx*, *unsigned int length = 144*)

Returns the context of JPOLY1305

Parameters

<i>ctx</i>	- context of the engine
<i>length</i>	- length of the context in bytes (144)

7.80.2.2 void ProtocolPP::jpoly1305::ProcessData (*const unsigned char * input*, *unsigned int length*)

Calculates the MAC using JPOLY1305

Parameters

<i>input</i>	- data to hash
<i>length</i>	- length of the input data

7.80.2.3 void ProtocolPP::jpoly1305::result (*unsigned char * mac*, *unsigned int length = 16*)

Returns the MAC using JPOLY1305

Parameters

<i>mac</i>	- result of the hash calculation
<i>length</i>	- length of the MAC data

The documentation for this class was generated from the following files:

- [include/jpoly1305.h](#)
- [include/poly1305-16.h](#)
- [include/poly1305-32.h](#)
- [include/poly1305-64.h](#)
- [include/poly1305-8.h](#)

7.81 jpoly1305 Class Reference

```
#include "include/jpoly1305.h"
```

7.81.1 Detailed Description

See the license at:

- [MIT](#) or PUBLIC DOMAIN

Protocol++ modified by : John Peter Greninger © John Peter Greninger 2015-2019 All Rights Reserved

All copyrights and trademarks are the property of their respective owners

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer
- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution
- Any and all modifications must be returned to John Peter Greninger at GitHub.com <https://github.com/jpgreninger/protocolpp> for evaluation. Inclusion of modifications in the source code shall be determined solely by John Peter Greninger. Failure to provide modifications shall render this license NULL and VOID and revoke any rights to use of Protocol++
- Commercial use requires a fee-based license obtainable at www.protocolpp.com
- Academic use requires written and notarized permission from John Peter and Sheila Rocha Greninger
- US Copyrights at <https://www.copyright.gov/>
 - TXU002059872 (Version 1.0.0)
 - TXU002066632 (Version 1.2.7)
 - TXU002082674 (Version 1.4.0)
 - TXU002097880 (Version 2.0.0)

The names of its contributors may not be used to endorse or promote products derived from this software without specific prior written permission

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE

The documentation for this class was generated from the following file:

- [include/jpoly1305.h](#)

7.82 ProtocolPP::jpoly1305_state_internal_t Struct Reference

```
#include <jpoly1305-16.h>
```

Public Attributes

- `unsigned char buffer [jpoly1305_block_size]`
- `size_t leftover`
- `unsigned short r [10]`
- `unsigned short h [10]`
- `unsigned short pad [8]`
- `unsigned char final`
- `unsigned long r [5]`
- `unsigned long h [5]`
- `unsigned long pad [4]`
- `unsigned long long r [3]`
- `unsigned long long h [3]`
- `unsigned long long pad [2]`
- `unsigned char h [17]`
- `unsigned char r [17]`
- `unsigned char pad [17]`

7.82.1 Member Data Documentation

- 7.82.1.1 `unsigned char ProtocolPP::jpoly1305_state_internal_t::buffer`
- 7.82.1.2 `unsigned char ProtocolPP::jpoly1305_state_internal_t::final`
- 7.82.1.3 `unsigned long ProtocolPP::jpoly1305_state_internal_t::h[5]`
- 7.82.1.4 `unsigned short ProtocolPP::jpoly1305_state_internal_t::h[10]`
- 7.82.1.5 `unsigned char ProtocolPP::jpoly1305_state_internal_t::h[17]`
- 7.82.1.6 `unsigned long long ProtocolPP::jpoly1305_state_internal_t::h[3]`
- 7.82.1.7 `size_t ProtocolPP::jpoly1305_state_internal_t::leftover`
- 7.82.1.8 `unsigned long ProtocolPP::jpoly1305_state_internal_t::pad[4]`
- 7.82.1.9 `unsigned short ProtocolPP::jpoly1305_state_internal_t::pad[8]`
- 7.82.1.10 `unsigned char ProtocolPP::jpoly1305_state_internal_t::pad[17]`
- 7.82.1.11 `unsigned long long ProtocolPP::jpoly1305_state_internal_t::pad[2]`
- 7.82.1.12 `unsigned long ProtocolPP::jpoly1305_state_internal_t::r[5]`
- 7.82.1.13 `unsigned short ProtocolPP::jpoly1305_state_internal_t::r[10]`
- 7.82.1.14 `unsigned char ProtocolPP::jpoly1305_state_internal_t::r[17]`
- 7.82.1.15 `unsigned long long ProtocolPP::jpoly1305_state_internal_t::r[3]`

The documentation for this struct was generated from the following files:

- [include/poly1305-16.h](#)
- [include/poly1305-32.h](#)
- [include/poly1305-64.h](#)
- [include/poly1305-8.h](#)

7.83 jproducer Class Reference

```
#include "include/jproducer.h"
```

7.83.1 Detailed Description

7.83.2 Producer for Protocol++

See Also

- [jring](#)
- [jdata](#)
- [jpacket](#)
- [jstream](#)
- [protocolpp](#)

Protocol++ written by : John Peter Greninger © John Peter Greninger 2015-2019 All Rights Reserved

All copyrights and trademarks are the property of their respective owners

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer
- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution
- Any and all modifications must be returned to John Peter Greninger at GitHub.com <https://github.com/jpgreninger/protocolpp> for evaluation. Inclusion of modifications in the source code shall be determined solely by John Peter Greninger. Failure to provide modifications shall render this license NULL and VOID and revoke any rights to use of Protocol++
- Commercial use requires a fee-based license obtainable at www.protocolpp.com
- Academic use requires written and notarized permission from John Peter and Sheila Rocha Greninger
- US Copyrights at <https://www.copyright.gov/>
 - TXu002059872 (Version 1.0.0)
 - TXu002066632 (Version 1.2.7)
 - TXu002082674 (Version 1.4.0)
 - TXu002097880 (Version 2.0.0)

The names of its contributors may not be used to endorse or promote products derived from this software without specific prior written permission

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE

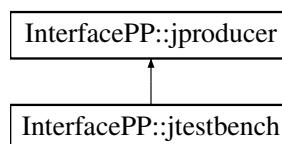
The documentation for this class was generated from the following file:

- jtestbench/include/jproducer.h

7.84 InterfacePP::jproducer Class Reference

```
#include <jproducer.h>
```

Inheritance diagram for InterfacePP::jproducer:



Public Member Functions

- **jproducer** (`ProtocolPP::platform_t` platform, unsigned long seed, int responders, `std::shared_ptr< jmmu >` &`mmu`, `std::shared_ptr< jlogger >` &`logger`, `std::shared_ptr< ProtocolPP::jdata >` &`indata`, `std::shared_ptr< jring< ringflow >>` &`fring`, `std::shared_ptr< jring< ringin >>` &`iring`, `std::shared_ptr< jring< ringout >>` &`oring`, `ProtocolPP::endian_t` endianess=`ProtocolPP::BIG`)
- virtual **~jproducer** ()

standard deconstructor
- virtual uint64_t **get_mem** (std::string packet, unsigned int `length`)=0
- virtual void **write** (uint64_t address, uint32_t data)=0
- virtual void **write** (uint64_t address, uint64_t data)=0
- virtual uint64_t **write** (std::string packet, uint8_t *data, unsigned int `length`)=0
- virtual uint64_t **write** (std::string packet, uint32_t *data, unsigned int `length`)=0
- virtual uint32_t **read** (uint64_t address)=0
- virtual void **read** (uint64_t address, uint8_t *data, unsigned int `length`)=0
- virtual void **read** (uint64_t address, uint32_t *data, unsigned int `length`)=0
- void **setup** (`std::shared_ptr< ProtocolPP::jstream >` &`stream`)
- void **issue** (`std::shared_ptr< ProtocolPP::jpacket >` &`packet`)
- void **release** (std::string stream)
- int **get_mode** ()
- void **run** ()

run the testbench
- uint32_t **get_status** (std::string &packet)

Protected Member Functions

- void **set_mem** (std::string packet, uint8_t *ptr)

run the testbench
- void **free_mem** (std::string packet)

run the testbench

7.84.1 Constructor & Destructor Documentation

7.84.1.1 `InterfacePP::jproducer::jproducer (ProtocolPP::platform_t platform, unsigned long seed, int responders, std::shared_ptr< jmmu > & mmu, std::shared_ptr< jlogger > & logger, std::shared_ptr< ProtocolPP::jdata > & indata, std::shared_ptr< jring< ringflow >> & fring, std::shared_ptr< jring< ringin >> & iring, std::shared_ptr< jring< ringout >> & oring, ProtocolPP::endian_t endianess = ProtocolPP::BIG)`

`jproducer` uses protocol++ to drive packets into a ring which can then be read by any device that supports a ring

Parameters

<code>platform</code>	- Platform to connect testbench to (W.A.S.P or SEC)
<code>seed</code>	- seed for the testbench
<code>responders</code>	- number of responders used
<code>mmu</code>	- tracks dynamic memory
<code>logger</code>	- object to print output
<code>indata</code>	- JDATA object containing flows and packets
<code>fring</code>	- Software ring for the flows
<code>iring</code>	- Software ring for the input packets
<code>oring</code>	- Software ring for the output packets
<code>endianess</code>	- Endianess of the platform to support

7.84.1.2 virtual InterfacePP::jproducer::~jproducer() [inline], [virtual]

standard deconstructor

7.84.2 Member Function Documentation

7.84.2.1 void InterfacePP::jproducer::free_mem (std::string *packet*) [protected]

run the testbench

7.84.2.2 virtual uint64_t InterfacePP::jproducer::get_mem (std::string *packet*, unsigned int *length*) [pure virtual]

Get a chunk of memory

Parameters

<i>packet</i>	- name of the packet to write data for
<i>length</i>	- Length of memory in bytes

Returns

address to memory

Implemented in [InterfacePP::jtestbench](#).

7.84.2.3 int InterfacePP::jproducer::get_mode ()

Returns

- current mode of operation

7.84.2.4 uint32_t InterfacePP::jproducer::get_status (std::string & *packet*)

retrieve the status word

Returns

- status of the indicated packet

7.84.2.5 void InterfacePP::jproducer::issue (std::shared_ptr<ProtocolPP::jpacket> & *packet*)

Issues the packet into the testbench or system must be overloaded in user's code

Parameters

<i>packet</i>	- packet to process
---------------	---------------------

See Also

[jpacket](#)

7.84.2.6 virtual uint32_t InterfacePP::jproducer::read (uint64_t *address*) [pure virtual]

Read function to overload in your testbench

Parameters

<i>address</i>	- Address to register
----------------	-----------------------

Returns

value of the register

Implemented in [InterfacePP::jtestbench](#).

7.84.2.7 virtual void InterfacePP::jproducer::read (*uint64_t address*, *uint8_t * data*, *unsigned int length*) [pure virtual]

Read function to overload in your testbench

Parameters

<i>address</i>	- Address to read data from
<i>data</i>	- Retrieved data
<i>length</i>	- Length of data to read

Implemented in [InterfacePP::jtestbench](#).

7.84.2.8 virtual void InterfacePP::jproducer::read (*uint64_t address*, *uint32_t * data*, *unsigned int length*) [pure virtual]

Read function to overload in your testbench

Parameters

<i>address</i>	- Address to read data from
<i>data</i>	- Retrieved data
<i>length</i>	- Length of data to read

Implemented in [InterfacePP::jtestbench](#).

7.84.2.9 void InterfacePP::jproducer::release (*std::string stream*)

Stream is done, release the stream and deallocate

Parameters

<i>stream</i>	- name of the stream to releasae
---------------	----------------------------------

7.84.2.10 void InterfacePP::jproducer::run ()

run the testbench

7.84.2.11 void InterfacePP::jproducer::set_mem (*std::string packet*, *uint8_t * ptr*) [protected]

run the testbench

7.84.2.12 void InterfacePP::jproducer::setup (*std::shared_ptr<ProtocolPP::jstream> & stream*)

Setup the stream to process descriptors. Using the security association, program your device to process the flow of packets. The security association is accessed as a VOID pointer and must be recast to the correct association type based on the TYPE field in the JSTREAM object

Parameters

<i>stream</i>	- Stream object to setup
---------------	--------------------------

See Also

[jstream](#)

7.84.2.13 virtual void InterfacePP::jproducer::write (*uint64_t address, uint32_t data*) [pure virtual]

Write function to overload in your testbench

Parameters

<i>address</i>	- Address to write data to
<i>data</i>	- Data to write to memory

Implemented in [InterfacePP::jtestbench](#).

7.84.2.14 virtual void InterfacePP::jproducer::write (*uint64_t address, uint64_t data*) [pure virtual]

Write function to overload in your testbench

Parameters

<i>address</i>	- Address to write data to
<i>data</i>	- Data to write to memory

Implemented in [InterfacePP::jtestbench](#).

7.84.2.15 virtual *uint64_t* InterfacePP::jproducer::write (*std::string packet, uint8_t * data, unsigned int length*) [pure virtual]

Write function to overload in your testbench

Parameters

<i>packet</i>	- name of the packet to write data for
<i>data</i>	- Data to write to memory
<i>length</i>	- length of data to write

Returns

address - Address data written to

Implemented in [InterfacePP::jtestbench](#).

7.84.2.16 virtual *uint64_t* InterfacePP::jproducer::write (*std::string packet, uint32_t * data, unsigned int length*) [pure virtual]

Write function to overload in your testbench

Parameters

<i>packet</i>	- name of the packet to write data for
<i>data</i>	- Word oriented data to write
<i>length</i>	- length of data to write

Returns

address - Address data written to

Implemented in [InterfacePP::jtestbench](#).

The documentation for this class was generated from the following file:

- [jtestbench/include/jproducer.h](#)

7.85 jprotocol Class Reference

```
#include "include/jprotocol.h"
```

7.85.1 Detailed Description

For API Documentation:

See Also

[tinyxml2::tinyxml2](#)
[ProtocolPP::jarray](#)
[ProtocolPP::jrand](#)
[ProtocolPP::jreplay](#)
[ProtocolPP::ciphers](#)

For Additional Documentation:

See Also

[tinyxml2](#)
[jarray](#)
[jrand](#)
[jreplay](#)
[ciphers](#)

Protocol++ written by : John Peter Greninger © John Peter Greninger 2015-2019 All Rights Reserved

All copyrights and trademarks are the property of their respective owners

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer
- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution
- Any and all modifications must be returned to John Peter Greninger at GitHub.com <https://github.com/jpgreninger/protocolpp> for evaluation. Inclusion of modifications in the source code shall be determined solely by John Peter Greninger. Failure to provide modifications shall render this license NULL and VOID and revoke any rights to use of Protocol++
- Commercial use requires a fee-based license obtainable at www.protocolpp.com
- Academic use requires written and notarized permission from John Peter and Sheila Rocha Greninger
- US Copyrights at [https://www.copyright.gov/](http://www.copyright.gov/)

- TXu002059872 (Version 1.0.0)
- TXu002066632 (Version 1.2.7)
- TXu002082674 (Version 1.4.0)
- TXu002097880 (Version 2.0.0)

The names of its contributors may not be used to endorse or promote products derived from this software without specific prior written permission

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE

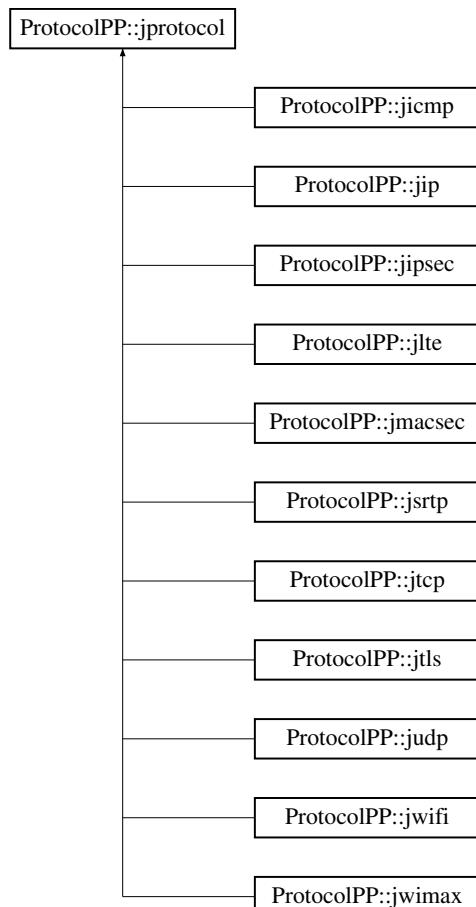
The documentation for this class was generated from the following file:

- [include/jprotocol.h](#)

7.86 ProtocolPP::jprotocol Class Reference

```
#include <jprotocol.h>
```

Inheritance diagram for ProtocolPP::jprotocol:



Public Member Functions

- `jprotocol (direction_t dir)`
 - `jprotocol (direction_t dir, std::string &file)`
 - `jprotocol (direction_t dir, std::shared_ptr< jrand > &rand)`
 - `jprotocol (direction_t dir, std::shared_ptr< jrand > &rand, std::string &file)`
 - `~jprotocol ()`
- Standard deconstructor.*
- `virtual void encapsulate_packet (std::shared_ptr< jarray< uint8_t >> &output)`
 - `virtual void decapsulate_packet (std::shared_ptr< jarray< uint8_t >> &input)`
 - `virtual void encapsulate_packet (std::shared_ptr< jarray< uint8_t >> &input, std::shared_ptr< jarray< uint8_t >> &output)=0`
 - `virtual void decapsulate_packet (std::shared_ptr< jarray< uint8_t >> &input, std::shared_ptr< jarray< uint8_t >> &output)=0`
 - `virtual jarray< uint8_t > get_hdr ()=0`
 - `virtual uint64_t get_field (field_t field)`
 - `virtual uint64_t get_field (field_t field, jarray< uint8_t > &hdr)=0`
 - `virtual void set_hdr (jarray< uint8_t > &hdr)=0`
 - `virtual void set_field (field_t field, uint64_t value)=0`
 - `bool hasfile ()`
 - `uint32_t get_status ()`
 - `virtual void to_xml (tinyxml2::XMLPrinter &myxml, direction_t direction)=0`
 - `jarray< uint8_t > checksum (iana_t prot, direction_t dir, jarray< uint8_t > &check, unsigned int offset=0)`
 - `jarray< uint8_t > pad (pad_t padtype, unsigned int len)`

Static Public Member Functions

- `const static std::string currentDateTime ()`
- `static std::string str_status (uint32_t status)`
- `static jarray< uint8_t > to_array (uint8_t input)`
- `static jarray< uint8_t > to_array (uint16_t input)`
- `static jarray< uint8_t > to_array (uint32_t input)`
- `static jarray< uint8_t > to_array (uint64_t input)`
- `static uint8_t to_u8 (jarray< uint8_t > src_array)`
- `static uint16_t to_u16 (jarray< uint8_t > src_array)`
- `static uint32_t to_u32 (jarray< uint8_t > src_array)`
- `static uint64_t to_u64 (jarray< uint8_t > src_array)`

Protected Member Functions

- `template<typename T , typename TE >`
`std::shared_ptr< jreplay< T, TE > > update_replay (protocol_t prot, T &seqnum, TE &extseq, bool usext, unsigned int size)`
- `template<typename T , typename TE >`
`std::shared_ptr< jreplay< T, TE > > update_replay (protocol_t prot, T &seqnum, TE &extseq, bool usext, unsigned int size, jarray< uint8_t > &window)`
- `void update_status (uint32_t stat)`
- `void get_data (std::shared_ptr< jarray< uint8_t >> &data)`
- `void put_data (std::shared_ptr< jarray< uint8_t >> &wdata)`
- `void encrypt (std::shared_ptr< jarray< uint8_t >> &data)`
- `void decrypt (std::shared_ptr< jarray< uint8_t >> &data)`
- `unsigned int roundup (unsigned int value, unsigned int mult)`

Protected Attributes

- `direction_t m_dir`
`direction of processing`
- `uint32_t m_status`
- `std::shared_ptr<jrand> m_rand`
`random data generator passed into constructor`
- `std::string m_filename`
`file input string for this protocol`
- `std::fstream m_file`
- `endian_t m_endian`
`endianess string for this protocol`

7.86.1 Constructor & Destructor Documentation

7.86.1.1 ProtocolPP::jprotocol::jprotocol (`direction_t dir`)

Constructor for base class of protocols which accepts the maximum transmission unit (MTU)

Parameters

<code>dir</code>	- direction of processing either ENCAP or DECAP
------------------	---

7.86.1.2 ProtocolPP::jprotocol::jprotocol (`direction_t dir, std::string & file`)

Constructor for base class of protocols which accepts the maximum transmission unit (MTU) and a path to a file which contains the data to transmit

Parameters

<code>dir</code>	- direction of processing either ENCAP or DECAP
<code>file</code>	- input file that contains data to transmit

7.86.1.3 ProtocolPP::jprotocol::jprotocol (`direction_t dir, std::shared_ptr<jrand> & rand`)

Constructor for base class of protocols which accepts the maximum transmission unit (MTU), an anti-replay structure, and a pointer to a random number generator for cryptographic purposes

Parameters

<code>dir</code>	- direction of processing either ENCAP or DECAP
<code>rand</code>	- pointer to random number generator for cryptographic functionality such as padding

7.86.1.4 ProtocolPP::jprotocol::jprotocol (`direction_t dir, std::shared_ptr<jrand> & rand, std::string & file`)

Constructor for base class of protocols which accepts the maximum transmission unit (MTU), an anti-replay structure, and a pointer to a random number generator for cryptographic purposes

Parameters

<code>dir</code>	- direction of processing either ENCAP or DECAP
<code>file</code>	- input file that contains data to transmit
<code>rand</code>	- pointer to random number generator for cryptographic functionality such as padding

7.86.1.5 ProtocolPP::jprotocol::~jprotocol()

Standard deconstructor.

7.86.2 Member Function Documentation

7.86.2.1 `jarray<uint8_t> ProtocolPP::jprotocol::checksum(iana_t prot, direction_t dir, jarray<uint8_t> & check, unsigned int offset = 0)`

helper functions for use with all protocols that need them Checksum calculates a 1's complement running sum across all 16-bit values in the input data. Algorithm is based on the protocol requested

Parameters

<i>prot</i>	- protocol to calculate the checksum for
<i>dir</i>	- direction of the data flow
<i>check</i>	- data to checksum
<i>offset</i>	- offset into the data to clear checksum field

Returns

array containing calculated checksum

7.86.2.2 `const static std::string ProtocolPP::jprotocol::currentDateTime() [inline], [static]`

Get current date/time, format is YYYY-MM-DD.HH:mm:ss

Returns

data/time formatted as string

7.86.2.3 `virtual void ProtocolPP::jprotocol::decap_packet(std::shared_ptr<jarray<uint8_t>> & input) [inline], [virtual]`

packet processing functions for use with protocols that access files (TCP, UDP, IP). Decap takes packets from the system decapsulates the packet, and writes the resulting payload to the file given in the constructor with each packet containing MTU worth of data in the packet

Parameters

<i>input</i>	- shared pointer to received packet whose payload will be written to the file passed to the constructor
--------------	---

Reimplemented in [ProtocolPP::jip](#), and [ProtocolPP::judp](#).

7.86.2.4 `virtual void ProtocolPP::jprotocol::decap_packet(std::shared_ptr<jarray<uint8_t>> & input, std::shared_ptr<jarray<uint8_t>> & output) [pure virtual]`

packet processing functions for use with all protocols. Decap takes a packet from an upper-layer protocol and decapsulates the packet and returns the payload. Both the input and output packets are shared pointers to arrays containing payloads and packets respectively

Implemented in [ProtocolPP::jipsec](#), [ProtocolPP::jwifi](#), [ProtocolPP::jmacsec](#), [ProtocolPP::jsrtp](#), [ProtocolPP::jtls](#), [ProtocolPP::jip](#), [ProtocolPP::jtcp](#), [ProtocolPP::jwimax](#), [ProtocolPP::jlte](#), [ProtocolPP::judp](#), and [ProtocolPP::jic平](#).

7.86.2.5 void ProtocolPP::jprotocol::decrypt (std::shared_ptr< jarray< uint8_t >> & *data*) [inline],
[protected]

Decrypt data (key, context, etc) If information needed for the protocol has key material in it decrypt the data before use

Parameters

<i>data</i>	- data to decrypt
-------------	-------------------

7.86.2.6 virtual void ProtocolPP::jprotocol::encap_packet (std::shared_ptr<jarray< uint8_t >> & *output*) [inline], [virtual]

packet processing functions for use with protocols that access files (TCP, UDP, IP). Encap generates packets from the file given in the constructor with each packet containing MTU worth of data in the packet

Parameters

<i>output</i>	- shared pointer to generated output packet
---------------	---

Reimplemented in [ProtocolPP::jip](#), and [ProtocolPP::judp](#).

7.86.2.7 virtual void ProtocolPP::jprotocol::encap_packet (std::shared_ptr<jarray< uint8_t >> & *input*, std::shared_ptr<jarray< uint8_t >> & *output*) [pure virtual]

packet processing functions for use with all protocols. Encap takes a payload or packet generated by a lower-level protocol encapsulates the packet according to the protocol selected, and returns the resulting packet. Both the input and output packets are shared pointers to arrays containing payloads and packets respectively

Implemented in [ProtocolPP::jipsec](#), [ProtocolPP::jwifi](#), [ProtocolPP::jmacsec](#), [ProtocolPP::jsrtp](#), [ProtocolPP::jtls](#), [ProtocolPP::jip](#), [ProtocolPP::jtcp](#), [ProtocolPP::jwimax](#), [ProtocolPP::jlte](#), [ProtocolPP::judp](#), and [ProtocolPP::jicmp](#).

7.86.2.8 void ProtocolPP::jprotocol::encrypt (std::shared_ptr<jarray< uint8_t >> & *data*) [inline], [protected]

Encrypt data (key, context, etc) If information needed for the protocol contains key material encrypt the data before use

Parameters

<i>data</i>	- data to encrypt
-------------	-------------------

7.86.2.9 void ProtocolPP::jprotocol::get_data (std::shared_ptr<jarray< uint8_t >> & *data*) [protected]

accessor functions for file I/O Get data accesses the file and retrieves MTU length of data from it

Parameters

<i>data</i>	- array filled with data from the file
-------------	--

Returns

returns true until EOF is reached then returns FALSE when last packet is returned with FALSE set, the array will be properly trimmed to the size of the remaining data

7.86.2.10 virtual uint64_t ProtocolPP::jprotocol::get_field (field_t *field*) [inline], [virtual]

Retrieve the value of the field

Parameters

<i>field</i>	- field to retrieve
--------------	---------------------

Returns

current value of the field

Reimplemented in [ProtocolPP::jipsec](#), [ProtocolPP::jwifi](#), [ProtocolPP::jmacsec](#), [ProtocolPP::jip](#), [ProtocolPP::jtcp](#), [ProtocolPP::jlte](#), and [ProtocolPP::judp](#).

7.86.2.11 virtual uint64_t ProtocolPP::jprotocol::get_field (*field_t field, jarray<uint8_t > &hdr*) [pure virtual]

Retrieve the value of the field

Parameters

<i>field</i>	- field to retrieve
<i>hdr</i>	- header to retrieve the field from

Returns

current value of the field

Implemented in [ProtocolPP::jipsec](#), [ProtocolPP::jwifi](#), [ProtocolPP::jmacsec](#), [ProtocolPP::jsrtp](#), [ProtocolPP::jtls](#), [ProtocolPP::jip](#), [ProtocolPP::jtcp](#), [ProtocolPP::jwimax](#), [ProtocolPP::jlte](#), [ProtocolPP::judp](#), and [ProtocolPP::jicmp](#).

7.86.2.12 virtual jarray<uint8_t> ProtocolPP::jprotocol::get_hdr () [pure virtual]

Retrieve the header for the packet

Returns

the full header

Implemented in [ProtocolPP::jipsec](#), [ProtocolPP::jwifi](#), [ProtocolPP::jmacsec](#), [ProtocolPP::jsrtp](#), [ProtocolPP::jtls](#), [ProtocolPP::jip](#), [ProtocolPP::jtcp](#), [ProtocolPP::jwimax](#), [ProtocolPP::jlte](#), [ProtocolPP::judp](#), and [ProtocolPP::jicmp](#).

7.86.2.13 uint32_t ProtocolPP::jprotocol::get_status () [inline]

return the status for the packet with the following format

```
* -----
* | Protocol (8-bits) | Error (8-bits) | Line Number (16-bits) |
* -----
*
```

- Protocol codes in STATUS:

- PROTOCOL - JProtocol base class (value of 1)
- MACSEC - Macsec protocol (value of 2)
- WIFI - Wifi protocol (value of 3)
- WIGIG - WiGig protocol (value of 4)
- WIMAX - WiMax protocol (value of 5)
- LTE - Long Term Evolution (LTE/3GPP) protocol (value of 6)
- RLC - Radio Link Control (value of 7)

- TLS - Transport Layer Security (TLS/SSL) protocol (value of 8)
 - SRTP - Secure RTP protocol (value of 9)
 - UDPP - User Datagram Protocol (UDP) (value of 10)
 - TCPP - Transport Control Protocol (TCP) (value of 11)
 - ICMPP - Internet Control Message Protocol (ICMP) (value of 12)
 - IP - Internet Protocol (IP) (value of 13)
 - IPSEC - Encapsulating Security Protocol (ESP) (value of 14)
 - XMLPARSER - TinyXML2 parser (value of 15)
 - WASP - W.A.S.P randomizer and packet generator front end (value of 16)
- Error Codes in STATUS:
 - ERR_NONE - No error (value of 0)
 - ERR_LATE - Anti-replay late packet error found (value of 1)
 - ERR_REPLAY - Anti-replay error found (value of 2)
 - ERR_ROLLOVER - Packet number rollover error (value of 3)
 - ERR_ROLLUNDER - Packet number rollunder error (value of 4)
 - ERR_PROGRAM - Programming error (value of 5)
 - ERR_ICV - ICV check failure (value of 6)
 - ERR_CRC - CRC check failure (value of 7)
 - ERR_READ_ENDIANFILE - Read end of file (value of 8)
 - ERR_READ_FAILFILE - Read failed in input file (value of 9)
 - ERR_READ_BADFILE - Read of missing file (value of 10)
 - ERR_WRITE_FAILFILE - Write to missing file (value of 11)
 - ERR_WRITE_BADFILE - Write error to output file (value of 12)
 - ERR_CHECKSUM - Checksum error (value of 13)
 - ERR_LISTEN - TCP server received packet while listening without SYN=1 (value of 14)
 - ERR_ACKNUM - TCP ACKNUM is incorrect (value of 15)
 - ERR_CLOSED - TCP session CLOSED during TX/RX (value of 16)
 - ERR_BITS - TCP header reserved bits error (value of 17)
 - ERR_TTL - TTL field of packet is zero (value of 18)
 - WARN_DUMMY - Dummy packet received warning (value of 19)
 - WARN_ZERO_DATA - Zero length payload warning (value of 20)
 - ERR_ICMP_PARAM - Parameter error requiring an ICMP message be sent (value of 21)
 - ERR_UNKNOWN - Unknown error (value of 22)
 - ERR_NONE - No error
 - ERR_LATE - Anti-replay late packet error found
 - ERR_REPLAY - Anti-replay error found
 - ERR_ROLLOVER - Packet number rollover error
 - ERR_ROLLUNDER - Packet number rollunder error
 - ERR_PROGRAM - Programming error
 - ERR_ICV - ICV check failure
 - ERR_CRC - CRC check failure
 - ERR_READ_ENDIANFILE - Read end of file
 - ERR_READ_FAILFILE - Read failed to input file handle
 - ERR_READ_BADFILE - Read of missing file handle

- ERR_WRITE_FAILFILE - Write failed to file output
- ERR_WRITE_BADFILE - Write error to file output
- ERR_CHECKSUM - Checksum error
- ERR_LISTEN - TCP server received packet while listening without SYN=1
- ERR_ACKNUM - TCP received ACKNUM is incorrect
- ERR_CLOSED - TCP session CLOSED during TX/RX
- ERR_BITS - TCP header reserved bits error
- ERR_TTL - TTL field of packet is zero
- ERR_JUMBOGRAM_FORMAT - IPv6 JUMBOGRAM formatting error
- WARN_DUMMY - Dummy packet warning
- WARN_IPV6_ROUTE - IPv6 Route header but not for this destination
- WARN_ZERO_DATA - Payload had zero length data
- ERR_ICMP_HDR_EXT_LEN - Header extension length is ODD for ROUTE header
- ERR_ICMP_SEGMENTS_LEFT - In ROUTE header, Segments left is larger than N
- ERR_MULTICAST_EXT_HDR - In ROUTE header, Multicast address present
- ERR_UNKNOWN_ROUTE_TYPE - In ROUTE header, Unknown route type (only type 0 is supported)
- ERR_CIPHER_KEY_SIZE - For the cipher requested, the key size is incorrect
- ERR_AUTH_KEY_SIZE - For the authentication requested, the key size is incorrect
- ERR_IV_SIZE - For the initialization vector requested, the size is incorrect
- ERR_SALT_SIZE - For the cipher salt requested, the size is incorrect
- ERR_ICV_SIZE - For the authentication requested, the ICV size is incorrect
- ERR_UNKNOWN_NXTHDR - For the IP packet, unknown NH value
- ERR_FORMAT_ERROR - Malformed packet
- WARN_INPUT_QUEUE_FULL - Send queue is full
- WARN_OUTPUT_QUEUE_EMPTY - Receive queue is empty
- ERR_SA_NOT_FOUND - Security Association not found
- ERR_NO_KEY - The key specified is invalid or not found
- ERR_KEY_EXPIRED - The key specified has expired
- ERR_KEY_REVOKED - The key specified has been revoked
- ERR_KEY_ACCESS - The key exists but is not readable by the calling process or can't be modified by the user
- ERR_KEY_NOT_SUPP - The key type does not support reading of the payload data
- ERR_KEY_QUOTA - The key quota for this user would be exceeded by creating this key
- ERR_KEY_INVALID - The payload data was invalid
- ERR_KEY_REJECTED - The attempt to generate a new key was rejected
- ERR_KEY_NOMEM - Insufficient memory to create a key
- ERR_KEY_INTR - The request was interrupted by a signal
- ERR_UNKNOWN - Unknown error
- Line number in code where the error occurred

7.86.2.14 bool ProtocolPP::jprotocol::hasfile() [inline]

check for file input/output

Returns

- file is present or not

7.86.2.15 `jarray<uint8_t> ProtocolPP::jprotocol::pad(pad_t padtype, unsigned int len)`

helper functions for use with all protocols that need them Pad generates padding material to be appended to the packet if needed. Supports either ZERO, INCREMENTING, SIZE, or RANDOM padding

Parameters

<i>padtype</i>	- Type of padding to generate (ZERO, INCREMENT, SIZE, RANDOM)
<i>len</i>	- the amount of padding to generate

Returns

array containing padding requested

7.86.2.16 void ProtocolPP::jprotocol::put_data (std::shared_ptr<jarray<uint8_t>> &*wdata*) [protected]

accessor functions for file I/O Put data accesses the file and writes the payload to the file

Parameters

<i>wdata</i>	- an array of data to write to the file
--------------	---

7.86.2.17 unsigned int ProtocolPP::jprotocol::roundup (unsigned int *value*, unsigned int *mult*) [inline], [protected]

roundup the value

Parameters

<i>value</i>	- Value to round up
<i>mult</i>	- Next multiple of value to round to

Returns

value rounded up to next multiple of mult

7.86.2.18 virtual void ProtocolPP::jprotocol::set_field (field_t *field*, uint64_t *value*) [pure virtual]

Set the specified field with a new value

Parameters

<i>field</i>	- field to set
<i>value</i>	- new value for the field

Implemented in [ProtocolPP::jipsec](#), [ProtocolPP::jwifi](#), [ProtocolPP::jmacsec](#), [ProtocolPP::jsrtp](#), [ProtocolPP::jtls](#), [ProtocolPP::jip](#), [ProtocolPP::jtcp](#), [ProtocolPP::jwimax](#), [ProtocolPP::jlte](#), [ProtocolPP::judp](#), and [ProtocolPP::jicmp](#).

7.86.2.19 virtual void ProtocolPP::jprotocol::set_hdr (jarray<uint8_t> &*hdr*) [pure virtual]

Set the header to the current value

Parameters

<i>hdr</i>	- new header value
------------	--------------------

Implemented in [ProtocolPP::jipsec](#), [ProtocolPP::jwifi](#), [ProtocolPP::jmacsec](#), [ProtocolPP::jsrtp](#), [ProtocolPP::jtls](#), [ProtocolPP::jip](#), [ProtocolPP::jtcp](#), [ProtocolPP::jwimax](#), [ProtocolPP::jlte](#), [ProtocolPP::judp](#), and [ProtocolPP::jicmp](#).

7.86.2.20 static std::string ProtocolPP::jprotocol::str_status (uint32_t *status*) [static]

Interpret the status word and return as a printable string

Parameters

<i>status</i>	- Status word to interpret
---------------	----------------------------

Returns

Printable string representation of the status word

7.86.2.21 static jarray<uint8_t> ProtocolPP::jprotocol::to_array (uint8_t *input*) [static]

Convert uint8_t into a byte array

Parameters

<i>input</i>	- uint8_t to convert
--------------	----------------------

Returns

byte array in current endian format

7.86.2.22 static jarray<uint8_t> ProtocolPP::jprotocol::to_array (uint16_t *input*) [static]

Convert uint16_t into a byte array

Parameters

<i>input</i>	- uint16_t to convert
--------------	-----------------------

Returns

byte array in current endian format

7.86.2.23 static jarray<uint8_t> ProtocolPP::jprotocol::to_array (uint32_t *input*) [static]

Convert uint32_t into a byte array

Parameters

<i>input</i>	- uint32_t to convert
--------------	-----------------------

Returns

byte array in current endian format

7.86.2.24 static jarray<uint8_t> ProtocolPP::jprotocol::to_array (uint64_t *input*) [static]

Convert uint64_t into a byte array

Parameters

<i>input</i>	- uint64_t to convert
--------------	-----------------------

Returns

byte array in current endian format

7.86.2.25 static uint16_t ProtocolPP::jprotocol::to_u16(*jarray<uint8_t> src_array*) [static]

Convert a byte array into a uint16_t

Parameters

<i>src_array</i>	- jarray to convert, must be 2 or fewer bytes
------------------	---

Returns

uint16_t in current endian format

7.86.2.26 static uint32_t ProtocolPP::jprotocol::to_u32 (*jarray< uint8_t > src_array*) [static]

Convert a byte array into a uint32_t

Parameters

<i>src_array</i>	- jarray to convert, must be 4 or fewer bytes
------------------	---

Returns

uint32_t in current endian format

7.86.2.27 static uint64_t ProtocolPP::jprotocol::to_u64 (*jarray< uint8_t > src_array*) [static]

Convert a byte array into a uint64_t

Parameters

<i>src_array</i>	- jarray to convert, must be 8 or fewer bytes
------------------	---

Returns

uint64_t in current endian format

7.86.2.28 static uint8_t ProtocolPP::jprotocol::to_u8 (*jarray< uint8_t > src_array*) [static]

Convert a byte array into a uint8_t

Parameters

<i>src_array</i>	- jarray to convert, must be 1 byte
------------------	-------------------------------------

Returns

uint8_t in current endian format

7.86.2.29 virtual void ProtocolPP::jprotocol::to_xml (*tinyxml2::XMLPrinter & myxml*, *direction_t direction*) [pure virtual]

print the protocol and security objects as XML

Parameters

<i>myxml</i>	- object to print to
<i>direction</i>	- randomization

Implemented in [ProtocolPP::ipsec](#), [ProtocolPP::jwifi](#), [ProtocolPP::jmacsec](#), [ProtocolPP::jsrtp](#), [ProtocolPP::jtls](#), [ProtocolPP::jip](#), [ProtocolPP::jtcp](#), [ProtocolPP::jwimax](#), [ProtocolPP::jlte](#), [ProtocolPP::judp](#), and [ProtocolPP::jic平](#).

7.86.2.30 template<typename T , typename TE > std::shared_ptr<jreplay<T,TE>> ProtocolPP::jprotocol::update_replay (protocol_t *prot*, T & *seqnum*, TE & *extseq*, bool *usext*, unsigned int *size*) [inline], [protected]

helper functions for use with all protocols that need them Update_replay returns a shared_ptr pointing at a new [jreplay\(\)](#) object for anti-replay functionality

Parameters

<i>prot</i>	- protocol to use anti-replay
<i>seqnum</i>	- initial sequence number
<i>extseq</i>	- initial extended sequence number
<i>usext</i>	- extended sequence number is used for anti-replay
<i>size</i>	- number of packets to track

Returns

shared_ptr to the anti-replay object

7.86.2.31 `template<typename T, typename TE> std::shared_ptr<jreplay<T,TE>> ProtocolPP::jprotocol::update_replay(protocol_t prot, T & seqnum, TE & extseq, bool usext, unsigned int size, jarray<uint8_t> & window) [inline], [protected]`

helper functions for use with all protocols that need them Update_replay returns a shared_ptr pointing at a new [jreplay\(\)](#) object for anti-replay functionality

Parameters

<i>prot</i>	- protocol to use anti-replay
<i>seqnum</i>	- initial sequence number
<i>extseq</i>	- initial extended sequence number
<i>usext</i>	- extended sequence number is used for anti-replay
<i>size</i>	- number of packets to track
<i>window</i>	- initial anti-replay window

Returns

shared_ptr to the anti-replay object

7.86.2.32 `void ProtocolPP::jprotocol::update_status(uint32_t stat) [inline], [protected]`

accessor functions for the status Updates the status with the new value

Parameters

<i>stat</i>	- new value for the status
-------------	----------------------------

7.86.3 Member Data Documentation

7.86.3.1 `direction_t ProtocolPP::jprotocol::m_dir [protected]`

direction of processing

7.86.3.2 `endian_t ProtocolPP::jprotocol::m_endian [protected]`

endianess string for this protocol

7.86.3.3 `std::fstream ProtocolPP::jprotocol::m_file [protected]`

7.86.3.4 `std::string ProtocolPP::jprotocol::m_filename [protected]`

file input string for this protocol

7.86.3.5 `std::shared_ptr<jrand> ProtocolPP::jprotocol::m_rand` [protected]

random data generator passed into constructor

7.86.3.6 `uint32_t ProtocolPP::jprotocol::m_status` [protected]

Status value for packet processing with format

```
* -----  
* | Protocol (8-bits) | Error (8-bits) | Line Number (16-bits) |  
* -----  
*
```

The documentation for this class was generated from the following file:

- [include/jprotocol.h](#)

7.87 jprotocolpp Class Reference

`#include "include/jprotocolpp.h"`

The documentation for this class was generated from the following file:

- [include/jprotocolpp.h](#)

7.88 ProtocolPP::jprotocolpp Class Reference

`#include <jprotocolpp.h>`

Static Public Member Functions

- static `std::shared_ptr<jmacsec> get_macsec (std::shared_ptr<jmacsecsa> &security)`
- static `std::shared_ptr<jipsec> get_ipsec (std::shared_ptr<jrand> &rand, std::shared_ptr<jipsecsa> &security)`
- static `std::shared_ptr<jtls> get_tls (std::shared_ptr<jrand> &rand, std::shared_ptr<jtlrsa> &security)`
- static `std::shared_ptr<jtls> get_tls (std::shared_ptr<jrand> &rand, std::shared_ptr<jtlrsa> &security, std::string file)`
- static `std::shared_ptr<jsrtp> get_srtp (std::shared_ptr<jrand> &rand, std::shared_ptr<jsrtpsa> &security)`
- static `std::shared_ptr<jwifi> get_wifi (std::shared_ptr<jwifisa> &security)`
- static `std::shared_ptr<jwimax> get_wimax (std::shared_ptr<jwimaxsa> &security)`
- static `std::shared_ptr<jtcp> get_tcp (jtcp::tcpstate_t state, std::shared_ptr<jtcpса> &security)`
- static `std::shared_ptr<jtcp> get_tcp (jtcp::tcpstate_t state, std::shared_ptr<jtcpса> &security, std::string file)`
- static `std::shared_ptr<judp> get_udp (std::shared_ptr<judpsa> &security)`
- static `std::shared_ptr<judp> get_udp (std::shared_ptr<judpsa> &security, std::string file)`
- static `std::shared_ptr<jicmp> get_icmp (std::shared_ptr<jicmpса> &security)`
- static `std::shared_ptr<jip> get_ip (std::shared_ptr<jipsa> &security)`
- static `std::shared_ptr<jip> get_ip (std::shared_ptr<jipsa> &security, std::string file)`
- static `std::shared_ptr<jlte> get_lte (std::shared_ptr<jltesa> &security)`

7.88.1 Member Function Documentation

7.88.1.1 static std::shared_ptr<jicmp> ProtocolPP::jprotocolpp::get_icmp (std::shared_ptr<jicm PSA > & security)
[static]

Factory method to create an instance of the ICMP protocol

Parameters

<i>security</i>	- parameters necessary to setup an ICMP flow
-----------------	--

7.88.1.2 static std::shared_ptr<jip> ProtocolPP::jprotocolpp::get_ip (std::shared_ptr<jipsa> & *security*) [static]

Factory method to create an instance of the IP protocol

Parameters

<i>security</i>	- parameters necessary to setup an IP flow
-----------------	--

7.88.1.3 static std::shared_ptr<jip> ProtocolPP::jprotocolpp::get_ip (std::shared_ptr<jipsa> & *security*, std::string *file*) [static]

Factory method to create an instance of the IP protocol with file input

Parameters

<i>security</i>	- parameters necessary to setup an IP flow
<i>file</i>	- filename for input/output data

7.88.1.4 static std::shared_ptr<jipsec> ProtocolPP::jprotocolpp::get_ipsec (std::shared_ptr<jrand> & *rand*, std::shared_ptr<jipsecsa> & *security*) [static]

Factory method for IPsec

Parameters

<i>rand</i>	- Random data generator for IVs and padding
<i>security</i>	- Security association (SA) for this IPsec flow

7.88.1.5 static std::shared_ptr<jlte> ProtocolPP::jprotocolpp::get_lte (std::shared_ptr<jltesa> & *security*) [static]

Factory method to create an instance of the LTE protocol

Parameters

<i>security</i>	- parameters necessary to setup a LTE flow
-----------------	--

7.88.1.6 static std::shared_ptr<jmacsec> ProtocolPP::jprotocolpp::get_macsec (std::shared_ptr<jmacsecsa> & *security*) [static]

Factory method to create an instance of the MACSEC protocol

Parameters

<i>security</i>	- security association for this flow
-----------------	--------------------------------------

7.88.1.7 static std::shared_ptr<jsrtp> ProtocolPP::jprotocolpp::get_srtp (std::shared_ptr<jrand> & *rand*, std::shared_ptr<jsrtpsa> & *security*) [static]

Factory method to create an instance of the SRTP protocol

Parameters

<i>rand</i>	- Random data generator for IVs and padding
<i>security</i>	- security association for this flow

7.88.1.8 static std::shared_ptr<jtcp> ProtocolPP::jprotocolpp::get_tcp (*jtcp::tcpstate_t state*, std::shared_ptr<*jtcpsa*> &*security*) [static]

Factory method to create an instance of the TCP protocol

Parameters

<i>state</i>	- State of initial operation (ESTABLISHED for randomization, CLOSED for normal operation)
<i>security</i>	- parameters necessary to setup a TCP flow

7.88.1.9 static std::shared_ptr<jtcp> ProtocolPP::jprotocolpp::get_tcp (*jtcp::tcpstate_t state*, std::shared_ptr<*jtcpsa*> &*security*, std::string *file*) [static]

Factory method to create an instance of the TCP protocol

Parameters

<i>state</i>	- State of initial operation (ESTABLISHED for randomization, CLOSED for normal operation)
<i>security</i>	- parameters necessary to setup a TCP flow
<i>file</i>	- filename for input/output data

7.88.1.10 static std::shared_ptr<jtls> ProtocolPP::jprotocolpp::get_tls (std::shared_ptr<*jrand*> &*rand*, std::shared_ptr<*jtsa*> &*security*) [static]

Factory method to create an instance of the TLS protocol

Parameters

<i>rand</i>	- Random data generator for IVs and padding
<i>security</i>	- security association for this flow

7.88.1.11 static std::shared_ptr<jtls> ProtocolPP::jprotocolpp::get_tls (std::shared_ptr<*jrand*> &*rand*, std::shared_ptr<*jtsa*> &*security*, std::string *file*) [static]

Factory method to create an instance of the TLS protocol

Parameters

<i>rand</i>	- Random data generator for IVs and padding
<i>security</i>	- security association for this flow
<i>file</i>	- filename for input/output data

7.88.1.12 static std::shared_ptr<judp> ProtocolPP::jprotocolpp::get_udp (std::shared_ptr<*judpsa*> &*security*) [static]

Factory method to create an instance of the UDP protocol

Parameters

<i>security</i>	- parameters necessary to setup a UDP flow
-----------------	--

7.88.1.13 static std::shared_ptr<judp> ProtocolPP::jprotocolpp::get_udp (std::shared_ptr<judpsa> & *security*, std::string *file*) [static]

Factory method to create an instance of the UDP protocol

Parameters

<i>security</i>	- parameters necessary to setup a UDP flow
<i>file</i>	- filename for input/output data

7.88.1.14 static std::shared_ptr<jwifi> ProtocolPP::jprotocolpp::get_wifi (std::shared_ptr<jwifisa> & *security*) [static]

Factory method to create an instance of the Wifi/WiGig protocol

Parameters

<i>security</i>	- security association for this flow
-----------------	--------------------------------------

7.88.1.15 static std::shared_ptr<jwimax> ProtocolPP::jprotocolpp::get_wimax (std::shared_ptr<jwimaxsa> & *security*) [static]

Factory method to create an instance of the WiMax protocol

Parameters

<i>security</i>	- security association for this flow
-----------------	--------------------------------------

The documentation for this class was generated from the following file:

- [include/jprotocolpp.h](#)

7.89 jrand Class Reference

```
#include "include/jrand.h"
```

7.89.1 Detailed Description

For API Documentation:

See Also

[ProtocolPP::jarray](#)

For Additional Documentation:

See Also

[jarray](#)

Protocol++ written by : John Peter Greninger © John Peter Greninger 2015-2019 All Rights Reserved

All copyrights and trademarks are the property of their respective owners

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer
- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution
- Any and all modifications must be returned to John Peter Greninger at GitHub.com <https://github.com/jpgreninger/protocolpp> for evaluation. Inclusion of modifications in the source code shall be determined solely by John Peter Greninger. Failure to provide modifications shall render this license NULL and VOID and revoke any rights to use of Protocol++
- Commercial use requires a fee-based license obtainable at www.protocolpp.com
- Academic use requires written and notarized permission from John Peter and Sheila Rocha Greninger
- US Copyrights at <https://www.copyright.gov/>
 - TXU002059872 (Version 1.0.0)
 - TXU002066632 (Version 1.2.7)
 - TXU002082674 (Version 1.4.0)
 - TXU002097880 (Version 2.0.0)

The names of its contributors may not be used to endorse or promote products derived from this software without specific prior written permission

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE

The documentation for this class was generated from the following file:

- [include/jrand.h](#)

7.90 ProtocolPP::jrand Class Reference

```
#include <jrand.h>
```

Public Member Functions

- [jrand \(\)](#)
 - [jrand \(unsigned long myseed\)](#)
 - [jrand \(const unsigned long *newseed, int size\)](#)
 - [virtual ~jrand \(\)](#)
- Standard deconstructor.*
- [bool getbool \(\)](#)
 - [std::string getname \(unsigned int length\)](#)

- std::string `getstr` (const char *input)
- `jarray< char > getchar` (unsigned int amount)
- `jarray< uint8_t > getbyte` (unsigned int amount)
- `jarray< uint8_t > getbyte` (const char *range)
- `jarray< uint16_t > getu16` (unsigned int amount)
- `jarray< uint16_t > getu16` (const char *range)
- `jarray< uint32_t > getword` (unsigned int amount)
- `jarray< uint32_t > getword` (const char *range)
- `jarray< uint64_t > getdouble` (unsigned int amount)
- `jarray< uint64_t > getdouble` (const char *range)
- int `get_int` ()
- int `get_int` (const char *range)
- unsigned int `get_uint` (const char *range)
- `uint8_t get_u8` ()
- `uint16_t get_u16` ()
- `uint32_t get_u32` ()
- `uint64_t get_u64` ()
- void `seed` (unsigned long myseed)
- void `seed` (const unsigned long *newseed, int size)
- `jarray< uint8_t > get_crypto` (unsigned int size)

Static Public Member Functions

- static std::vector< std::string > `tokenizer` (const char *input, const char *delimiters)

7.90.1 Constructor & Destructor Documentation

7.90.1.1 ProtocolPP::jrand::jrand ()

Standard constructor for the jrand class. Seeded with 0x5489UL

7.90.1.2 ProtocolPP::jrand::jrand (unsigned long *myseed*)

Constructor for jrand that accepts an unsigned int for the seed

Parameters

<i>myseed</i>	- seed to initialize psuedo-random number generator
---------------	---

7.90.1.3 ProtocolPP::jrand::jrand (const unsigned long * *newseed*, int *size*)

Constructor for jrand that accepts an unsigned int for the seed

Parameters

<i>newseed</i>	- array of seeds to initialize psuedo-random number generator
<i>size</i>	- Size of the initialization array

NOTE: This constructor is only available previous to the C++11 standard

7.90.1.4 virtual ProtocolPP::jrand::~jrand () [inline], [virtual]

Standard deconstructor.

7.90.2 Member Function Documentation

7.90.2.1 `jarray<uint8_t> ProtocolPP::jrand::get_crypto (unsigned int size)`

Method to obtain cryptographic random data to be used for keys, salt, IV

Parameters

<code>size</code>	- length of cryptographic data
-------------------	--------------------------------

Returns

array filled with hardware generated random numbers

7.90.2.2 int ProtocolPP::jrand::get_int()

Method to obtain a single random integer from the random number generator

Returns

integer

7.90.2.3 int ProtocolPP::jrand::get_int(const char * range)

Method to obtain a single random integer from the random number generator

Parameters

<code>range</code>	Range of integers to randomize over. Can either be of the form "1..32" or "16:32:64"
--------------------	--

Returns

integer

7.90.2.4 uint16_t ProtocolPP::jrand::get_u16()

Method to obtain a single random uint16_t from the random number generator

Returns

`uint16_t`

7.90.2.5 uint32_t ProtocolPP::jrand::get_u32()

Method to obtain a single random uint32_t from the random number generator

Returns

`uint32_t`

7.90.2.6 uint64_t ProtocolPP::jrand::get_u64()

Method to obtain a single random uint64_t from the random number generator

Returns

`uint64_t`

7.90.2.7 uint8_t ProtocolPP::jrand::get_u8()

Method to obtain a single random byte from the random number generator

Returns

uint8_t

7.90.2.8 unsigned int ProtocolPP::jrand::get_uint(const char * range)

Method to obtain a single random unsigned integer from the random number generator

Parameters

<i>range</i>	Range of integers to randomize over. Can either be of the form "1..32" or "16:32:64"
--------------	--

Returns

integer

7.90.2.9 bool ProtocolPP::jrand::getbool()

Method to obtain a random boolean from the random number generator

Returns

randomly chooseen True or False

7.90.2.10 jarray<uint8_t> ProtocolPP::jrand::getbyte(unsigned int amount)

Method to obtain an array of bytes from the random number generator

Parameters

<i>amount</i>	- number of bytes to generate
---------------	-------------------------------

Returns

Array of random bytes

7.90.2.11 jarray<uint8_t> ProtocolPP::jrand::getbyte(const char * range)

Method to obtain an array of bytes from the random number generator

Parameters

<i>range</i>	- String representing a range of sizes ex., "1..256"
--------------	--

Returns

Array of random bytes

7.90.2.12 jarray<char> ProtocolPP::jrand::getchar(unsigned int amount)

Method to obtain an array of bytes from the random number generator

Parameters

<i>amount</i>	- number of bytes to generate
---------------	-------------------------------

Returns

Array of random bytes

7.90.2.13 `jarray<uint64_t> ProtocolPP::jrand::getdouble (unsigned int amount)`

Method to obtain an array of uint64_t from the random number generator

Parameters

<i>amount</i>	- number of uint64_t to generate
---------------	----------------------------------

Returns

Array of random uint64_t

7.90.2.14 `jarray<uint64_t> ProtocolPP::jrand::getdouble (const char * range)`

Method to obtain an array of double words from the random number generator

Parameters

<i>range</i>	- String representing a range of sizes ex., "1..256"
--------------	--

Returns

Array of random double words

7.90.2.15 `std::string ProtocolPP::jrand::getname (unsigned int length)`

Method to obtain a random name from the random number generator

Parameters

<i>length</i>	- length of random string to generate
---------------	---------------------------------------

Returns

randomly generated name

7.90.2.16 `std::string ProtocolPP::jrand::getstr (const char * input)`

Method to obtain a random string from the random number generator

Parameters

<i>input</i>	- string of colon separated strings to choose from
--------------	--

Returns

randomly selected string from the input

7.90.2.17 `jarray<uint16_t> ProtocolPP::jrand::getu16 (unsigned int amount)`

Method to obtain an array of uint16_t from the random number generator

Parameters

<i>amount</i>	- number of uint16_t to generate
---------------	----------------------------------

Returns

Array of random uint16_t

7.90.2.18 jarray<uint16_t> ProtocolPP::jrand::getu16 (const char * *range*)

Method to obtain an array of shorts from the random number generator

Parameters

<i>range</i>	- String representing a range of sizes ex., "1..256"
--------------	--

Returns

Array of random shorts

7.90.2.19 jarray<uint32_t> ProtocolPP::jrand::getword (unsigned int *amount*)

Method to obtain an array of uint32_t from the random number generator

Parameters

<i>amount</i>	- number of uint32_t to generate
---------------	----------------------------------

Returns

Array of random uint32_t

7.90.2.20 jarray<uint32_t> ProtocolPP::jrand::getword (const char * *range*)

Method to obtain an array of words from the random number generator

Parameters

<i>range</i>	- String representing a range of sizes ex., "1..256"
--------------	--

Returns

Array of random words

7.90.2.21 void ProtocolPP::jrand::seed (unsigned long *myseed*)

Method to reseed generator with unsigned long

Parameters

<i>myseed</i>	- new seed for random number generator
---------------	--

7.90.2.22 void ProtocolPP::jrand::seed (const unsigned long * *newseed*, int *size*)

Method to reseed generator with array of unsigned long of size

Parameters

<i>newseed</i>	- array of new seeds for random number generator
<i>size</i>	- length of array of seeds

NOTE: This method is only available previous to the C++11 standard

7.90.2.23 static std::vector<std::string> ProtocolPP::jrand::tokenizer (const char * *input*, const char * *delimiters*)
 [inline], [static]

tokenize the string

Parameters

<i>input</i>	- String to tokenize
<i>delimiters</i>	- Delimiters to split the string (with no spaces e.g., "-,. ")

Returns

vector containing tokens

The documentation for this class was generated from the following file:

- include/jrand.h

7.91 jreplay Class Reference

```
#include "include/jreplay.h"
```

7.91.1 Detailed Description

7.91.2 Anti-Replay

This class uses the sliding window anti-replay method found in the IPsec specification to track any number of packets. To allow large numbers of packets, the class uses a BYTE array instead of an unsigned integer to track received packets. The replay window can either be initialized to an empty window or with a windows that's provided. The double template is for use with current protocols that may have an extended packet number. This assumes that the extended packet number is the same size as the seqnum.

Example initialized empty window to track 16 packets

```
*      16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
*
*-----*
* | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
*
*      16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1   C
*      *   *   *   *   *   *   *   *   *   *   *   *   *   *   *   *
*      P   P   P   P   P   P   P   P   P   P   P   P   P   P   P   P
*      r   r   r   r   r   r   r   r   r   r   r   r   r   r   r   r
*      e   e   e   e   e   e   e   e   e   e   e   e   e   e   e   e
*      v   v   v   v   v   v   v   v   v   v   v   v   v   v   v   v
*      i   i   i   i   i   i   i   i   i   i   i   i   i   i   i   i
*      o   o   o   o   o   o   o   o   o   o   o   o   o   o   o   o
*      u   u   u   u   u   u   u   u   u   u   u   u   u   u   u   u
*      s   s   s   s   s   s   s   s   s   s   s   s   s   s   s   s   s
```

Example of LATE packet

```
*      16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
*-----*
```

```

* | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
*
* 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 C
* * * * * * * * * * * * * * * * * * * * * * u
* P P P P P P P P P P P P P P P P P P P P r
* r r r r r r r r r r r r r r r r r r r r r
* e e e e e e e e e e e e e e e e e e e e e
* v v v v v v v v v v v v v v v v v v v v n
* i i i i i i i i i i i i i i i i i i i i t
* o o o o o o o o o o o o o o o o o o o o o
* u u u u u u u u u u u u u u u u u u u u u
* s s s s s s s s s s s s s s s s s s s s s
*
*

```

For the LATE example, say we receive a packet that has a packet number placing it 18 packets behind the current packet. As shown in the example, the packet would not be recorded in the window and would be dropped. Packet numbers would not be updated and a LATE error would be returned

Example of packet in the window

```

* 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
*
* | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
*
* 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 C
* * * * * * * * * * * * * * * * * * * * * * u
* P P P P P P P P P P P P P P P P P P P P r
* r r r r r r r r r r r r r r r r r r r r r
* e e e e e e e e e e e e e e e e e e e e e
* v v v v v v v v v v v v v v v v v v v v n
* i i i i i i i i i i i i i i i i i i i i t
* o o o o o o o o o o o o o o o o o o o o o
* u u u u u u u u u u u u u u u u u u u u u
* s s s s s s s s s s s s s s s s s s s s s
*
*
```

For the packet in window example, say we receive a packet that has a packet number placing it 9 packets behind the current packet. As shown in the example, the packet would be recorded in the window in the spot nine packets behind the current packet. Packet numbers would not be updated and no error would be returned

Example of REPLAY packet

```

* 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
*
* | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
*
* 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 C
* * * * * * * * * * * * * * * * * * * * * * u
* P P P P P P P P P P P P P P P P P P P P r
* r r r r r r r r r r r r r r r r r r r r r
* e e e e e e e e e e e e e e e e e e e e e
* v v v v v v v v v v v v v v v v v v v v n
* i i i i i i i i i i i i i i i i i i i i t
* o o o o o o o o o o o o o o o o o o o o o
* u u u u u u u u u u u u u u u u u u u u u
* s s s s s s s s s s s s s s s s s s s s s
*
*
```

For the REPLAY example, say we receive a packet that has a packet number placing it 9 packets behind the current packet. As shown in the example, the packet would coincide with the previously received packet nine packets behind the current packet. The packet would be discarded and a REPLAY error would be returned. Packet numbers would not be updated

Example of EARLY packet

```

* 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
*
* | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 |
*
* 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 C
* * * * * * * * * * * * * * * * * * * * * * u
* P P P P P P P P P P P P P P P P P P P P r
* r r r r r r r r r r r r r r r r r r r r r

```

```

*   e   e   e   e   e   e   e   e   e   e   e   e   e   e   e   e
*   v   v   v   v   v   v   v   v   v   v   v   v   v   v   v   v   n
*   i   i   i   i   i   i   i   i   i   i   i   i   i   i   i   i   t
*   o   o   o   o   o   o   o   o   o   o   o   o   o   o   o   o
*   u   u   u   u   u   u   u   u   u   u   u   u   u   u   u   u
*   s   s   s   s   s   s   s   s   s   s   s   s   s   s   s   s   s
*
*

```

For the EARLY example, say we receive a packet that has a packet number placing it 2 packets ahead of the current packet. As shown in the example, window would be shifted left by two packets and a one would be placed in the current packet position. Packet numbers will be updated to reflect the newer packet number received. No error is reported

For API Documentation:

See Also

[ProtocolPP::jarray](#)

For Additional Documentation:

See Also

[jarray](#)

Protocol++ written by : John Peter Greninger © John Peter Greninger 2015-2019 All Rights Reserved

All copyrights and trademarks are the property of their respective owners

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer
- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution
- Any and all modifications must be returned to John Peter Greninger at GitHub.com <https://github.com/jpgreninger/protocolpp> for evaluation. Inclusion of modifications in the source code shall be determined solely by John Peter Greninger. Failure to provide modifications shall render this license NULL and VOID and revoke any rights to use of Protocol++
- Commercial use requires a fee-based license obtainable at www.protocolpp.com
- Academic use requires written and notarized permission from John Peter and Sheila Rocha Greninger
- US Copyrights at [https://www.copyright.gov/](http://www.copyright.gov/)
 - TXU002059872 (Version 1.0.0)
 - TXU002066632 (Version 1.2.7)
 - TXU002082674 (Version 1.4.0)
 - TXU002097880 (Version 2.0.0)

The names of its contributors may not be used to endorse or promote products derived from this software without specific prior written permission

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT

OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE

The documentation for this class was generated from the following file:

- [include/jreplay.h](#)

7.92 ProtocolPP::jreplay< T, TE > Class Template Reference

```
#include <jreplay.h>
```

Public Member Functions

- [jreplay \(protocol_t prot, T &seqnum, TE &extseq, bool usext, unsigned int winsize\)](#)
- [jreplay \(protocol_t prot, T &seqnum, TE &extseq, bool usext, unsigned int winsize, jarray< uint8_t > &window\)](#)
- [virtual ~jreplay \(\)](#)
Standard deconstructor.
- [uint32_t antireplay \(T &curseq, TE ¤t\)](#)
- [unsigned int size \(\)](#)
- [int next \(T &curseq, TE ¤t, replay_t type=NORMAL\)](#)
- [T get_seqnum \(\)](#)
- [TE get_extseq \(\)](#)
- [jarray< uint8_t > get_window \(\)](#)
- [std::string print \(bool pretty=false, int indent=10\)](#)

7.92.1 Constructor & Destructor Documentation

7.92.1.1 template<typename T , typename TE > jreplay::jreplay (`protocol_t prot, T & seqnum, TE & extseq, bool usext, unsigned int winsize`)

Constructor for replay

Parameters

<code>prot</code>	- protocol for anti-replay
<code>seqnum</code>	- sequence number to verify
<code>extseq</code>	- extended sequence number to verify
<code>usext</code>	- flag to indicate use of extended sequence number
<code>winsize</code>	- number of packets to track in the window

7.92.1.2 template<typename T , typename TE > jreplay::jreplay (`protocol_t prot, T & seqnum, TE & extseq, bool usext, unsigned int winsize, jarray< uint8_t > & window`)

Constructor for replay

Parameters

<code>prot</code>	- protocol for anti-replay
<code>seqnum</code>	- sequence number to verify
<code>extseq</code>	- extended sequence number to verify
<code>usext</code>	- flag to indicate use of extended sequence number
<code>winsize</code>	- number of packets to track in the window
<code>window</code>	- initial replay window to start with populated with UNIT8_T values where one bit equals a packet e.g., 0xF0 would mean four packets have been received

7.92.1.3 `template<typename T, typename TE> virtual ProtocolPP::jreplay< T, TE >::~jreplay() [inline], [virtual]`

Standard deconstructor.

7.92.2 Member Function Documentation

7.92.2.1 `template<typename T, typename TE> uint32_t jreplay::antireplay(T & curseq, TE & currseq)`

anti-replay function, uses an BYTE array to support any size window instead of being limited to 32, 64, or 128 packets

Parameters

<code>curseq</code>	- received sequence number to verify
<code>currseq</code>	- received extended sequence number to verify

Returns

value indicating type of error

0 - no error

1 - late error

2 - replay error

3 - rollover error (overflow)

4 - rollunder error (underflow)

5 - programming error

7.92.2.2 `template<typename T, typename TE> TE ProtocolPP::jreplay< T, TE >::get_extseq() [inline]`

Return the current extended sequence number

Returns

size of the window

7.92.2.3 `template<typename T, typename TE> T ProtocolPP::jreplay< T, TE >::get_seqnum() [inline]`

Return current sequence number

Returns

size of the window

7.92.2.4 `template<typename T, typename TE> jarray<uint8_t> ProtocolPP::jreplay< T, TE >::get_window() [inline]`

Return the current anti-replay window

Returns

the window

7.92.2.5 template<typename T , typename TE > int jreplay::next (T & currseq, TE & currext, replay_t type = NORMAL)

Return value to add to seqnum for next value

Parameters

<i>type</i>	- Type of seqnum/extseqnum to retrieve (NORMAL, WINDOW, REPLAY, LATE, OVERFLOW, UNDERFLOW)
<i>currseq</i>	- Current seqnum
<i>currext</i>	- Current extended seqnum

Returns

amount to add (or subtract) to the current seqnum for the type requested

7.92.2.6 template<typename T , typename TE > std::string jreplay::print (bool pretty = false, int indent = 10)

Return string representation of replay window

Parameters

<i>pretty</i>	- adds formatting to the replay window to make it more readable as follows
<i>indent</i>	- causes each line of the pretty print window to be indented the number of spaces requested

Returns

string representation of replay window array

```

Anti-replay window (size=500)

| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | C | | | | | |
| * | + | * | + | * | + | * | + | * | + | * | + | * | + | * | + | * | + | * | + | * | + | * | + | * | + | * | U |
| P | P | P | P | P | P | P | P | P | P | P | P | P | P | P | P | P | P | P | P | P | P | P | P | P | P | P | r |
| r | r | r | r | r | r | r | r | r | r | r | r | r | r | r | r | r | r | r | r | r | r | r | r | r | r | r | r |
| e | e | e | e | e | e | e | e | e | e | e | e | e | e | e | e | e | e | e | e | e | e | e | e | e | e | e | e |
| v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | n |
| i | i | i | i | i | i | i | i | i | i | i | i | i | i | i | i | i | i | i | i | i | i | i | i | i | i | i | t |
| o | o | o | o | o | o | o | o | o | o | o | o | o | o | o | o | o | o | o | o | o | o | o | o | o | o | o | o |
| u | u | u | u | u | u | u | u | u | u | u | u | u | u | u | u | u | u | u | u | u | u | u | u | u | u | u | u |
| s | s | s | s | s | s | s | s | s | s | s | s | s | s | s | s | s | s | s | s | s | s | s | s | s | s | s | s | V |

| oldest | <----- | most recent |

| 0 | 1 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | * 0 |
| 1 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | * 24 |
| 1 | 1 | 0 | 0 | 1 | 0 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 0 | 0 | 1 | 1 | 1 | 1 | * 48 |
| 1 | 0 | 1 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 1 | 0 | 1 | 1 | 1 | 1 | 0 | * 72 |
| 0 | 0 | 0 | 1 | 0 | 1 | 1 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | * 96 |
| 0 | 0 | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 0 | 0 | 0 | 1 | 1 | 0 | 1 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | * 120 |
| 1 | 1 | 1 | 0 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | * 144 |
| 1 | 1 | 1 | 1 | 0 | 1 | 0 | 1 | 1 | 1 | 0 | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 1 | * 168 |
| 1 | 0 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 0 | * 192 |
| 0 | 1 | 1 | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | * 216 |
| 0 | 0 | 0 | 1 | 0 | 0 | 1 | 1 | 0 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 1 | 1 | 1 | * 240 |
| 1 | 0 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | * 264 |
| 0 | 1 | 1 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | * 288 |
| 0 | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | * 312 |
| 0 | 1 | 0 | 1 | 1 | 0 | 1 | 1 | 1 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | * 336 |
| 1 | 1 | 1 | 1 | 0 | 0 | 1 | 0 | 1 | 1 | 0 | 1 | 1 | 0 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | * 360 |
| 1 | 1 | 0 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | * 384 |
| 0 | 1 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 0 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | * 408 |
| 1 | 1 | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | * 432 |
| 1 | 1 | 1 | 1 | 0 | 1 | 0 | 1 | 1 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | * 456 |
| X | X | X | X | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | * 480 |

```

Figure 7.103: Screen Capture of Anti-Replay Pretty-Print

7.92.2.7 template<typename T , typename TE > unsigned int ProtocolPP::jreplay< T, TE >::size() [inline]

Return the size of the anti-replay window

Returns

size of the window

The documentation for this class was generated from the following file:

- [include/jreplay.h](#)

7.93 jresponder Class Reference

```
#include "include/jresponder.h"
```

7.93.1 Detailed Description

7.93.2 Responder unit for use in testbench for Protocol++

Protocol++ written by : John Peter Greninger © John Peter Greninger 2015-2019 All Rights Reserved

All copyrights and trademarks are the property of their respective owners

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer
- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution
- Any and all modifications must be returned to John Peter Greninger at GitHub.com <https://github.com/jpgreninger/protocolpp> for evaluation. Inclusion of modifications in the source code shall be determined solely by John Peter Greninger. Failure to provide modifications shall render this license NULL and VOID and revoke any rights to use of Protocol++
- Commercial use requires a fee-based license obtainable at www.protocolpp.com
- Academic use requires written and notarized permission from John Peter and Sheila Rocha Greninger
- US Copyrights at [https://www.copyright.gov/](http://www.copyright.gov/)
 - TXU002059872 (Version 1.0.0)
 - TXU002066632 (Version 1.2.7)
 - TXU002082674 (Version 1.4.0)
 - TXU002097880 (Version 2.0.0)

The names of its contributors may not be used to endorse or promote products derived from this software without specific prior written permission

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE

The documentation for this class was generated from the following file:

- jresponder/include/jresponder.h

7.94 InterfacePP::jresponder Class Reference

```
#include <jresponder.h>
```

Public Member Functions

- `jresponder` (`std::shared_ptr<jlogger> &logger, std::shared_ptr<jring<ringflow>> &flowring, std::shared_ptr<jring<ringin>> &inputring, std::shared_ptr<jring<ringout>> &outputring, unsigned long seed, std::string readlatency=std::string("80..120"), std::string computelatency=std::string("2000..5000"), std::string writelatency=std::string("50..100"))`
- void `dequeue` (`std::string flow`)

enqueue flows to responder
- void `enqueue` (`std::string &flow`)
- void `push` (`uint64_t packet, unsigned int size, uint32_t status`)
- `ringin pop` ()
- void `run` ()

run the responder
- void `teardown` ()

teardown the responder

7.94.1 Constructor & Destructor Documentation

7.94.1.1 `InterfacePP::jresponder::jresponder(std::shared_ptr<jlogger> & logger, std::shared_ptr<jring<ringflow>> & flowring, std::shared_ptr<jring<ringin>> & inputring, std::shared_ptr<jring<ringout>> & outputring, unsigned long seed, std::string readlatency = std::string("80..120"), std::string computelatency = std::string("2000..5000"), std::string writelatency = std::string("50..100"))`

constructor for responder to test wasp interface

Parameters

<code>logger</code>	- object to print output
<code>flowring</code>	- software ring for the flows
<code>inputring</code>	- software ring for the input packets
<code>outputring</code>	- software ring for the output packets
<code>seed</code>	- seed for the random number generator
<code>readlatency</code>	- latency for read transactions
<code>computelatency</code>	- latency for compute transactions
<code>writelatency</code>	- latency for write transactions

7.94.2 Member Function Documentation

7.94.2.1 `void InterfacePP::jresponder::dequeue(std::string flow)`

enqueue flows to responder

7.94.2.2 `void InterfacePP::jresponder::enqueue(std::string & flow)`

dequeue flows from responder

Parameters

<code>flow</code>	- flow to dequeue
-------------------	-------------------

7.94.2.3 `ringin InterfacePP::jresponder::pop()`

pop next packet from the input ring

Returns

- input ring object

7.94.2.4 void InterfacePP::jresponder::push (uint64_t packet, unsigned int size, uint32_t status)

push another packet to the output ring

Parameters

<i>packet</i>	- address for the packet
<i>size</i>	- size of the packet in bytes
<i>status</i>	- status of the finished packet

7.94.2.5 void InterfacePP::jresponder::run ()

run the responder

7.94.2.6 void InterfacePP::jresponder::teardown ()

teardown the responder

The documentation for this class was generated from the following file:

- [jresponder/include/jresponder.h](#)

7.95 jring Class Reference

```
#include "include/jring.h"
```

7.95.1 Detailed Description

7.95.2 Software ring for use in testbench for Protocol++

Protocol++ written by : John Peter Greninger © John Peter Greninger 2015-2019 All Rights Reserved

All copyrights and trademarks are the property of their respective owners

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer
- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution
- Any and all modifications must be returned to John Peter Greninger at GitHub.com <https://github.com/jpgreninger/protocolpp> for evaluation. Inclusion of modifications in the source code shall be determined solely by John Peter Greninger. Failure to provide modifications shall render this license NULL and VOID and revoke any rights to use of Protocol++
- Commercial use requires a fee-based license obtainable at www.protocolpp.com
- Academic use requires written and notarized permission from John Peter and Sheila Rocha Greninger

- US Copyrights at <https://www.copyright.gov/>
 - TXu002059872 (Version 1.0.0)
 - TXu002066632 (Version 1.2.7)
 - TXu002082674 (Version 1.4.0)
 - TXu002097880 (Version 2.0.0)

The names of its contributors may not be used to endorse or promote products derived from this software without specific prior written permission

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE

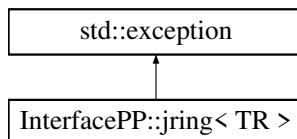
The documentation for this class was generated from the following file:

- jtestbench/include/jring.h

7.96 InterfacePP::jring< TR > Class Template Reference

```
#include <jring.h>
```

Inheritance diagram for InterfacePP::jring< TR >:



Public Member Functions

- **jring** (uint64_t address, unsigned int **size**)
- virtual **~jring** ()
 - standard deconstructor*
- void **push** (TR &input)
- TR **pop** ()
- TR **pop** (std::string stream)
- void **move** (uint64_t newaddr, unsigned int newsize)
- unsigned int **size** ()
- void **reset** ()
 - sets the pointers to original values*
- unsigned int **get_avail** ()
- unsigned int **get_proc** ()

7.96.1 Constructor & Destructor Documentation

7.96.1.1 template<typename TR > jring::jring (uint64_t address, unsigned int size)

constructor for software ring

Parameters

<i>address</i>	- location of the software ring
<i>size</i>	- size of the ring in number of entries

7.96.1.2 template<typename TR > virtual InterfacePP::jring< TR >::~jring() [inline], [virtual]

standard deconstructor

7.96.2 Member Function Documentation

7.96.2.1 template<typename TR > unsigned int jring::get_avail()

retrieve available slots in the ring

Returns

- returns number of slots available in the ring

7.96.2.2 template<typename TR > unsigned int jring::get_proc()

retrieve number of packets available to process

Returns

- returns the number of objects ready for processing

7.96.2.3 template<typename TR > void jring::move(uint64_t *newaddr*, unsigned int *newsiz*)

move the ring to a new location with new size

Parameters

<i>newaddr</i>	- New address for the software ring
<i>newsiz</i>	- New size of the software ring in number of entries

7.96.2.4 template<typename TR > TR InterfacePP::jring< TR >::pop()

removes the object

Returns

- returns an object from the software ring

7.96.2.5 template<typename TR > TR InterfacePP::jring< TR >::pop(std::string *stream*)

removes the object

Parameters

<i>stream</i>	- stream to return from the ring
---------------	----------------------------------

Returns

- returns an object from the software ring

7.96.2.6 template<typename TR > void InterfacePP::jring< TR >::push (TR & *input*)

adds the object

Parameters

<i>input</i>	- Object to add to the software ring
--------------	--------------------------------------

7.96.2.7 template<typename TR > void jring::reset ()

sets the pointers to original values

7.96.2.8 template<typename TR > unsigned int jring::size ()

size of the ring in number of objects

Returns

- size of the ring

The documentation for this class was generated from the following file:

- [jtestbench/include/jring.h](#)

7.97 jringdrive Class Reference

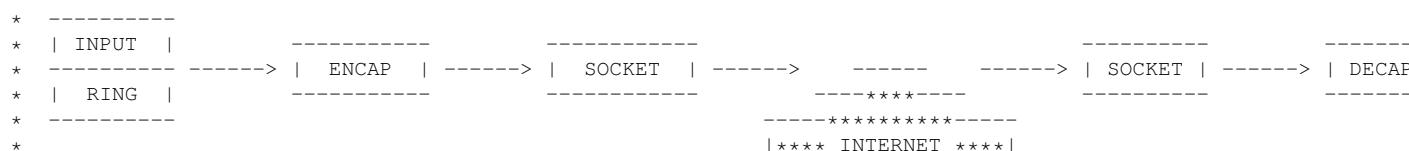
```
#include "include/jringdrive.h"
```

7.97.1 Detailed Description

7.97.2 Driver for Protocol++ with input and output driven by a software ring

Ring Driver opens a socket to the host and port that's passed in, reads packets from the input ring, encapsulates the packet with the protocol this driver belongs to, update the security association with any new values. The driver will continue to write packets to the socket as long as they are available in the input ring. Status is logged to the file. The ring will periodically check for available packets if not busy

On the receive side, the driver will occasionally check the socket for available packets. If packets are available, they are read from the socket, decapsulated with the Protocol++, and the payload is pushed to the output ring. The system can read available packets from the ring to see if packets have been received and processed



```

* ----- *----- *----- *----- *----- *----- *----- *----- *
* | OUTPUT | ----- | DECAP | <----- | SOCKET | <----- ----- | SOCKET | <----- | ENCAP
* | RING | ----- ----- ----- ----- ----- ----- ----- ----- |
* -----
*
```

See Also

[jring](#)
[jdata](#)
[jpacket](#)
[jstream](#)
[jprotocolpp](#)

Protocol++ written by : John Peter Greninger © John Peter Greninger 2015-2019 All Rights Reserved

All copyrights and trademarks are the property of their respective owners

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer
- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution
- Any and all modifications must be returned to John Peter Greninger at GitHub.com for evaluation. Inclusion of modifications in the source code shall be determined solely by John Peter Greninger. Failure to provide modifications shall render this license NULL and VOID and revoke any rights to use of Protocol++
- Commercial use requires a fee-based license obtainable at www.protocolpp.com
- US Copyrights at <https://www.copyright.gov/>
 - TXu002059872 (Version 1.0.0)
 - TXu002066632 (Version 1.2.7)
 - TXu002082674 (Version 1.4.0)
 - TXu002097880 (Version 2.0.0)

The names of its contributors may not be used to endorse or promote products derived from this software without specific prior written permission

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE

The documentation for this class was generated from the following file:

- [drivers/include/jringdrive.h](#)

7.98 DriverPP::jringdrive Class Reference

```
#include <jringdrive.h>
```

Public Member Functions

- `jringdrive` (bool *listen*, int *port*, `ProtocolPP::protocol_t` *prot*, `ProtocolPP::iana_t` *lvl3*, `ProtocolPP::protocol_t` *lvl2*, bool *lvl3_sec*, std::string &*host*, unsigned long *seed*, `ProtocolPP::endian_t` *endianess*, std::shared_ptr<`InterfacePP::jmmu`> &*mmu*, std::shared_ptr<`InterfacePP::jlogger`> &*logger*, std::shared_ptr<`InterfacePP::jring`<`InterfacePP::ringin`>> &*iring*, std::shared_ptr<`InterfacePP::jring`<`InterfacePP::ringout`>> &*oring*)
standard deconstructor
- `uint64_t write_mem` (`uint8_t` **data*, unsigned int *length*)
- `void send` ()
- `void receive` ()
- `uint32_t get_status` (std::string &*packet*)
- `void teardown` ()
terminate the driver

7.98.1 Constructor & Destructor Documentation

7.98.1.1 `DriverPP::jringdrive` (bool *listen*, int *port*, `ProtocolPP::protocol_t` *prot*, `ProtocolPP::iana_t` *lvl3*, `ProtocolPP::protocol_t` *lvl2*, bool *lvl3_sec*, std::string & *host*, unsigned long *seed*, `ProtocolPP::endian_t` *endianess*, std::shared_ptr<`InterfacePP::jmmu`> & *mmu*, std::shared_ptr<`InterfacePP::jlogger`> & *logger*, std::shared_ptr<`InterfacePP::jring`<`InterfacePP::ringin`>> & *iring*, std::shared_ptr<`InterfacePP::jring`<`InterfacePP::ringout`>> & *oring*)

`jringdrive` uses protocol++ to drive packets into a ring which can then be read by any device that supports a ring. Application layer protocols supported are TLS, SRTP, TCP, and UDP

Parameters

<i>listen</i>	- if set to true, configures to listen on the port specified
<i>port</i>	- port number
<i>prot</i>	- transport or application level protocol (TCP, UDP, TLS, SRTP)
<i>lvl3</i>	- network layer protocol (IPV4, IPV6)
<i>lvl2</i>	- data-link layer protocol (MACSEC, WIFI, WIMAX, LTE)
<i>lvl3_sec</i>	- Enable ESP for network level
<i>host</i>	- string of the host name (or IP Address in standard byte format XXX.XXX.XXX...)
<i>seed</i>	- seed for the randomizer
<i>mmu</i>	- tracks dynamic memory
<i>logger</i>	- object for writing
<i>iring</i>	- Software ring for the input packets
<i>oring</i>	- Software ring for the output packets
<i>endianess</i>	- Endianess of the platform to support

7.98.1.2 `virtual DriverPP::jringdrive::~jringdrive` () [inline], [virtual]

standard deconstructor

7.98.2 Member Function Documentation

7.98.2.1 `uint32_t DriverPP::jringdrive::get_status (std::string & packet)`

retrieve the status word

Returns

- status of the indicated packet

7.98.2.2 `void DriverPP::jringdrive::receive ()`

Receive function for the driver, must be called from a separate thread

7.98.2.3 `void DriverPP::jringdrive::send ()`

Send function for the driver must be called from a separate thread

7.98.2.4 `void DriverPP::jringdrive::teardown ()`

terminate the driver

7.98.2.5 `uint64_t DriverPP::jringdrive::write_mem (uint8_t * data, unsigned int length)`

Write function to overload in your testbench

Parameters

<code>data</code>	- Data to write to memory
<code>length</code>	- length of data to write

Returns

- address - Address data written to

The documentation for this class was generated from the following file:

- drivers/include/jringdrive.h

7.99 jrsa Class Reference

```
#include "include/jrsa.h"
```

7.99.1 Detailed Description**7.99.2 RSA CryptoSystem**

see [https://en.wikipedia.org/wiki/RSA_\(cryptosystem\)](https://en.wikipedia.org/wiki/RSA_(cryptosystem))

RSA (Rivest–Shamir–Adleman) is one of the first public-key cryptosystems and is widely used for secure data transmission. In such a cryptosystem, the encryption key is public and it is different from the decryption key which is kept secret (private). In RSA, this asymmetry is based on the practical difficulty of the factorization of the product of two large prime numbers, the "factoring problem". The acronym RSA is made of the initial letters of the surnames of Ron Rivest, Adi Shamir, and Leonard Adleman, who first publicly described the algorithm in 1978. Clifford Cocks,

an English mathematician working for the British intelligence agency Government Communications Headquarters (GCHQ), had developed an equivalent system in 1973, but this was not declassified until 1997

A user of RSA creates and then publishes a public key based on two large prime numbers, along with an auxiliary value. The prime numbers must be kept secret. Anyone can use the public key to encrypt a message, but with currently published methods, and if the public key is large enough, only someone with knowledge of the prime numbers can decode the message feasibly.[2] Breaking RSA encryption is known as the RSA problem. Whether it is as difficult as the factoring problem remains an open question. RSA is a relatively slow algorithm, and because of this, it is less commonly used to directly encrypt user data. More often, RSA passes encrypted shared keys for symmetric key cryptography which in turn can perform bulk encryption-decryption operations at much higher speed

The idea of an asymmetric public-private key cryptosystem is attributed to Whitfield Diffie and Martin Hellman, who published this concept in 1976. They also introduced digital signatures and attempted to apply number theory. Their formulation used a shared-secret-key created from exponentiation of some number, modulo a prime number. However, they left open the problem of realizing a one-way function, possibly because the difficulty of factoring was not well-studied at the time

Ron Rivest, Adi Shamir, and Leonard Adleman at the Massachusetts Institute of Technology made several attempts, over the course of a year, to create a one-way function that was hard to invert. Rivest and Shamir, as computer scientists, proposed many potential functions, while Adleman, as a mathematician, was responsible for finding their weaknesses. They tried many approaches including "knapsack-based" and "permutation polynomials". For a time, they thought what they wanted to achieve was impossible due to contradictory requirements.[4] In April 1977, they spent Passover at the house of a student and drank a good deal of Manischewitz wine before returning to their homes at around midnight. Rivest, unable to sleep, lay on the couch with a math textbook and started thinking about their one-way function. He spent the rest of the night formalizing his idea, and he had much of the paper ready by daybreak. The algorithm is now known as RSA – the initials of their surnames in same order as their paper

Clifford Cocks, an English mathematician working for the British intelligence agency Government Communications Headquarters (GCHQ), described an equivalent system in an internal document in 1973. However, given the relatively expensive computers needed to implement it at the time, RSA was considered to be mostly a curiosity and, as far as is publicly known, was never deployed. His discovery, however, was not revealed until 1997 due to its top-secret classification.

Operation

The RSA algorithm involves four steps: key generation, key distribution, encryption and decryption

A basic principle behind RSA is the observation that it is practical to find three very large positive integers e , d and n such that with modular exponentiation for all integers m (with $0 \leq m < n$):

$$(m^e)^d \equiv m \pmod{n}$$

and that even knowing e and n or even m it can be extremely difficult to find d

In addition, for some operations it is convenient that the order of the two exponentiations can be changed and that this relation also implies:

$$(m^d)^e \equiv m \pmod{n}$$

RSA involves a public key and a private key. The public key can be known by everyone, and it is used for encrypting messages. The intention is that messages encrypted with the public key can only be decrypted in a reasonable amount of time by using the private key. The public key is represented by the integers n and e ; and, the private key, by the integer d (although n is also used during the decryption process. Thus, it might be considered to be a part of the private key, too). m represents the message (previously prepared with a certain technique explained below)

Key generation

The keys for the RSA algorithm are generated the following way:

- Choose two distinct prime numbers p and q
 - For security purposes, the integers p and q should be chosen at random, and should be similar in magnitude but differ in length by a few digits to make factoring harder. Prime integers can be efficiently found using a primality test
- Compute $n = pq$

- n is used as the modulus for both the public and private keys. Its length, usually expressed in bits, is the key length
- Compute $\lambda(n) = \text{lcm}(\lambda(p), \lambda(q)) = \text{lcm}(p-1, q-1)$, where λ is Carmichael's totient function. This value is kept private
- Choose an integer e such that $1 < e < \lambda(n)$ and $\text{gcd}(e, \lambda(n)) = 1$; i.e., e and $\lambda(n)$ are coprime
- Determine d as $d \equiv e^{-1}(\text{mod } \lambda(n))$; i.e., d is the modular multiplicative inverse of $e \text{ mod } \lambda(n)$
 - This means: solve for d the equation $d \cdot e \equiv 1(\text{mod } \lambda(n))$
 - e having a short bit-length and small Hamming weight results in more efficient encryption – most commonly $e = 2^{16} + 1 = 65,537$. However, much smaller values of e (such as 3) have been shown to be less secure in some settings
 - e is released as the public key exponent
 - d is kept as the private key exponent

The public key consists of the modulus n and the public (or encryption) exponent e . The private key consists of the private (or decryption) exponent d , which must be kept secret. p, q , and $\lambda(n)$ must also be kept secret because they can be used to calculate d

In the original RSA paper, the Euler totient function $\phi(n) = (p-1)(q-1)$ is used instead of $\lambda(n)$ for calculating the private exponent d . Since $\phi(n)$ is always divisible by $\lambda(n)$ the algorithm works as well

Key distribution

Suppose that Bob wants to send information to Alice. If they decide to use RSA, Bob must know Alice's public key to encrypt the message and Alice must use her private key to decrypt the message. To enable Bob to send his encrypted messages, Alice transmits her public key (n, e) to Bob via a reliable, but not necessarily secret, route. Alice's private key (d) is never distributed

Encryption

After Bob obtains Alice's public key, he can send a message M to Alice. To do it, he first turns M (strictly speaking, the un-padded plaintext) into an integer m (strictly speaking, the padded plaintext), such that $0 < m < n$ by using an agreed-upon reversible protocol known as a padding scheme. He then computes the ciphertext c , using Alice's public key e , corresponding to $c \equiv m^e \pmod{n}$

This can be done reasonably quickly, even for 500-bit numbers, using modular exponentiation. Bob then transmits c to Alice

Decryption

Alice can recover m from c by using her private key exponent d by computing

$$c^d \equiv (m^e)^d \equiv m \pmod{n}$$

Given m , she can recover the original message M by reversing the padding scheme

Signing messages

Suppose Alice uses Bob's public key to send him an encrypted message. In the message, she can claim to be Alice but Bob has no way of verifying that the message was actually from Alice since anyone can use Bob's public key to send him encrypted messages. In order to verify the origin of a message, RSA can also be used to sign a message

Suppose Alice wishes to send a signed message to Bob. She can use her own private key to do so. She produces a hash value of the message, raises it to the power of $d \pmod{n}$ (as she does when decrypting a message), and attaches it as a "signature" to the message. When Bob receives the signed message, he uses the same hash algorithm in conjunction with Alice's public key. He raises the signature to the power of $e \pmod{n}$ (as he does when encrypting a message), and compares the resulting hash value with the message's actual hash value. If the two agree, he knows that the author of the message was in possession of Alice's private key, and that the message has not been tampered with since

Padding schemes

To avoid these problems, practical RSA implementations typically embed some form of structured, randomized padding into the value m before encrypting it. This padding ensures that m does not fall into the range of insecure

plaintexts, and that a given message, once padded, will encrypt to one of a large number of different possible ciphertexts

Standards such as PKCS#1 have been carefully designed to securely pad messages prior to RSA encryption. Because these schemes pad the plaintext m with some number of additional bits, the size of the un-padded message M must be somewhat smaller. RSA padding schemes must be carefully designed so as to prevent sophisticated attacks which may be facilitated by a predictable message structure. Early versions of the PKCS#1 standard (up to version 1.5) used a construction that appears to make RSA semantically secure

Secure padding schemes such as RSA-PSS are as essential for the security of message signing as they are for message encryption

Security

The security of the RSA cryptosystem is based on two mathematical problems: the problem of factoring large numbers and the RSA problem. Full decryption of an RSA ciphertext is thought to be infeasible on the assumption that both of these problems are hard, i.e., no efficient algorithm exists for solving them. Providing security against partial decryption may require the addition of a secure padding scheme

Importance of strong random number generation

A cryptographically strong random number generator, which has been properly seeded with adequate entropy, must be used to generate the primes p and q

Strong random number generation is important throughout every phase of public key cryptography. For instance, if a weak generator is used for the symmetric keys that are being distributed by RSA, then an eavesdropper could bypass RSA and guess the symmetric keys directly

For API information:

See Also

[ProtocolPP::jrsa](#)
[ProtocolPP::jrand](#)
[ProtocolPP::jenum](#)

For Additional Documentation:

See Also

[jrsa](#)
[jrand](#)
[jenum](#)

Protocol++ written by : John Peter Greninger © John Peter Greninger 2015-2019 All Rights Reserved

All copyrights and trademarks are the property of their respective owners

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer
- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution
- Any and all modifications must be returned to John Peter Greninger at GitHub.com <https://github.com/jpgreninger/protocolpp> for evaluation. Inclusion of modifications in the source code shall be determined solely by John Peter Greninger. Failure to provide modifications shall render this license NULL and VOID and revoke any rights to use of Protocol++
- Commercial use requires a fee-based license obtainable at www.protocolpp.com

- Academic use requires written and notarized permission from John Peter and Sheila Rocha Greninger
- US Copyrights at <https://www.copyright.gov/>
 - TXu002059872 (Version 1.0.0)
 - TXu002066632 (Version 1.2.7)
 - TXu002082674 (Version 1.4.0)
 - TXu002097880 (Version 2.0.0)

The names of its contributors may not be used to endorse or promote products derived from this software without specific prior written permission

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE

The documentation for this class was generated from the following file:

- [include/jrsa.h](#)

7.100 ProtocolPP::jrsa Class Reference

```
#include <jrsa.h>
```

Public Member Functions

- [jrsa \(int bitsize, rsapadtype_t padding, std::shared_ptr< InterfacePP::jlogger > &logger\)](#)
- [~jrsa \(\)](#)

Standard deconstructor.
- template<typename T >
[void set_field \(field_t field, T value\)](#)
- template<typename T >
[T get_field \(field_t field\)](#)
- [void gen_keypair \(\)](#)

Generate key pair.
- [void encrypt \(std::shared_ptr< jarray< uint8_t >> &data, std::shared_ptr< jarray< uint8_t >> &ciphertext\)](#)
- [void decrypt \(std::shared_ptr< jarray< uint8_t >> &ciphertext, std::shared_ptr< jarray< uint8_t >> &data\)](#)
- [void sign \(std::shared_ptr< jarray< uint8_t >> &data, std::shared_ptr< jarray< uint8_t >> &signature\)](#)
- [bool verify \(std::shared_ptr< jarray< uint8_t >> &signature\)](#)

7.100.1 Constructor & Destructor Documentation

7.100.1.1 ProtocolPP::jrsa (int *bitsize*, rsapadtype_t *padding*, std::shared_ptr< InterfacePP::jlogger > & *logger*)

Standard constructor

	field name	Example
	<code>RSAPAD</code> <code>bitsize</code> - size of the private key <code>padding</code> - padding types (PKCS#5 and PSS) <code>logger</code> - logging object <code>BITSIZE</code>	<code>rsapad_t rsapad = get_field<ProtocolPP::rsapad_t>(ProtocolPP::field_t::RSAPAD)</code> <code>int bitsize = get_field<int>(ProtocolPP::BITSIZE)</code>
cByteBlock	PRVKEY 7.100.1.2 <code>ProtocolPP::jrsa::~jrsa()</code> [inlined]	<code>CryptoPP::SecByteBlock prvkey = get_field<CryptoPP::SecByteBlock>(ProtocolPP::PRVKEY)</code>
cByteBlock	PUBKEY Standard destructor.	<code>CryptoPP::SecByteBlock pubkey = get_field<CryptoPP::SecByteBlock>(ProtocolPP::PUBKEY)</code>
<code>ProtocolPP::SecByteBlock ></code>	7.100.2 KEYPAIR Function Documentation 7.100.2.1 <code>void ProtocolPP::jrsa::decrypt(std::shared_ptr<CryptoPP::SecByteBlock> keypair, std::shared_ptr<jarray< uint8_t >> & ciphertext)</code> Decrypt the data	<code>std::shared_ptr<std::pair<CryptoPP::SecByteBlock, CryptoPP::SecByteBlock>> keypair = get_field<std::pair<CryptoPP::SecByteBlock, CryptoPP::SecByteBlock>>(ProtocolPP::KEYPAIR)</code>

Parameters Table 7.56: RSA Set Fields

<code>ciphertext</code>	- encrypted data
<code>data</code>	- decrypted data

7.100.2.2 `void ProtocolPP::jrsa::encrypt(std::shared_ptr<jarray< uint8_t >> & data, std::shared_ptr<jarray< uint8_t >> & ciphertext)`

Encrypt the data

Parameters

<code>data</code>	- data to encrypt
<code>ciphertext</code>	- encrypted data

7.100.2.3 `void ProtocolPP::jrsa::gen_keypair()`

Generate key pair.

7.100.2.4 `template<typename T > T ProtocolPP::jrsa::get_field(field_t field)`

Returns the version field RSA object

Parameters

<code>field</code>	- field to retrieve from the IP security association
--------------------	--

Returns

field of the IP security association

7.100.2.5 `template<typename T > void ProtocolPP::jrsa::set_field(field_t field, T value)`

Allows the user to update the field RSA object

Parameters

<i>field</i>	- field to update the IP security association
<i>value</i>	- value to update the IP security association

7.100.2.6 void ProtocolPP::jrsa::sign (std::shared_ptr<jarray<uint8_t>> &*data*, std::shared_ptr<jarray<uint8_t>> &*signature*)

cipher
NULL_CIPHER
DES_CBC
3DES_CBC
AES_CBC
AES_CTR
AES_CCM
AES_GCM

Sign the data

Parameters

<i>data</i>	- Data to sign
<i>signature</i>	- Generated signature

7.100.2.7 bool ProtocolPP::jrsa::verify (std::shared_ptr<jarray<uint8_t>> &*signature*)

Verify the data

Parameters

<i>signature</i>	- Signature to verify
------------------	-----------------------

Returns

result of the verification

The documentation for this class was generated from the following file:

- [include/jrsa.h](#)

7.101 jsec Class Reference

```
#include "include/jsec.h"
```

7.101.1 Detailed Description

7.101.2 Security Encryption CoProcessor (SEC)

Class to create shared descriptors for the SEC/CAAM architecture found in the Freescale/NXP/Qualcomm QorIQ and Layerscape network processors See the Security Reference Manual for the QorIQ processor desired for shared descriptor layout of each protocol in the ENCAP and DECAP directions

Also created job descriptors to process input and output data in the SEC/CAAM architecture. Contains support for reading and writing SG tables in the formats supported by SEC/CAAM

NOTE: Commands in the shared descriptor are big endian but inline data may be little endian if the platform requires it such as some of the ARM based Layerscape processors. Examples of data that may be little endian are the IV, NONCE, SALT, and INITCNT

Protocol++ written by : John Peter Greninger © John Peter Greninger 2015-2019 All Rights Reserved

All copyrights and trademarks are the property of their respective owners

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer
- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution
- Any and all modifications must be returned to John Peter Greninger at GitHub.com <https://github.com/jpgreninger/protocolpp> for evaluation. Inclusion of modifications in the source code shall be determined solely by John Peter Greninger. Failure to provide modifications shall render this license NULL and VOID and revoke any rights to use of Protocol++
- Commercial use requires a fee-based license obtainable at www.protocolpp.com
- Academic use requires written and notarized permission from John Peter and Sheila Rocha Greninger
- US Copyrights at <https://www.copyright.gov/>
 - TXU002059872 (Version 1.0.0)
 - TXU002066632 (Version 1.2.7)
 - TXU002082674 (Version 1.4.0)
 - TXU002097880 (Version 2.0.0)

The names of its contributors may not be used to endorse or promote products derived from this software without specific prior written permission

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE

The documentation for this class was generated from the following file:

- [platforms/SEC/include/jsec.h](#)

7.102 PlatformPP::jsec Class Reference

```
#include <jsec.h>
```

Classes

- struct [sgt_t](#)

Public Member Functions

- [jsec \(sgt_t sgt, unsigned long seed\)](#)
- virtual [~jsec \(\)](#)
Standard deconstructor.
- void [get_shared](#) (std::shared_ptr<[ProtocolPP::jarray< uint8_t >>> &shared, \[ProtocolPP::protocol_t\]\(#\) type, std::shared_ptr<\[ProtocolPP::jsecass\]\(#\) > security\)](#)
- void [get_desc](#) (std::shared_ptr<[ProtocolPP::jarray< uint8_t >>> &desc, uint64_t shareaddr, unsigned int sharesize, uint64_t inaddr, unsigned int inlen, bool insgf, uint64_t outaddr, unsigned int outlen, bool outsgf\)](#)
- void [read_sgt](#) (uint8_t *sgtin, std::shared_ptr<[ProtocolPP::jarray< uint8_t >>> &data\)](#)

7.102.1 Constructor & Destructor Documentation

7.102.1.1 PlatformPP::jsec (*sgt_t sgt, unsigned long seed*)

Constructor for JSEC

Parameters

<i>sgt</i>	- Structure of SG table for SEC/CAAM
<i>seed</i>	- Seed for random size generation for SG table construction

7.102.1.2 virtual PlatformPP::jsec::~jsec () [inline], [virtual]

Standard deconstructor.

7.102.2 Member Function Documentation

7.102.2.1 void PlatformPP::jsec::get_desc (std::shared_ptr< ProtocolPP::jarray< uint8_t >> & desc, uint64_t shareaddr, unsigned int sharesize, uint64_t inaddr, unsigned int inlen, bool insgf, uint64_t outaddr, unsigned int outlen, bool outsgf)

Return a properly formatted job descriptor

Parameters

<i>desc</i>	- pointer to hold descriptor
<i>shareaddr</i>	- shared descriptor address
<i>sharesize</i>	- size of the shared descriptor
<i>inaddr</i>	- address of input data
<i>inlen</i>	- length of the input data
<i>insgf</i>	- flag for scattered input data (true for scattered)
<i>outaddr</i>	- address of output data
<i>outlen</i>	- length of the output data
<i>outsgf</i>	- flag for scattered output data (true for scattered)

7.102.2.2 void PlatformPP::jsec::get_shared (std::shared_ptr< ProtocolPP::jarray< uint8_t >> & shared, ProtocolPP::protocol_t type, std::shared_ptr< ProtocolPP::jsecass > security)

Return the properly formatted shared descriptor for QorIQ or Layerscape processors

Parameters

<i>shared</i>	- Byte pointer to shared descriptor
<i>type</i>	- protocol to request shared descriptor for
<i>security</i>	- security association to derive shared descriptor from

7.102.2.3 void PlatformPP::jsec::read_sgt (uint8_t * sgtin, std::shared_ptr< ProtocolPP::jarray< uint8_t >> & data)

Read the scatter-gather table and return the data

Parameters

<i>sgtin</i>	- pointer to SG table
<i>data</i>	- data from SG table

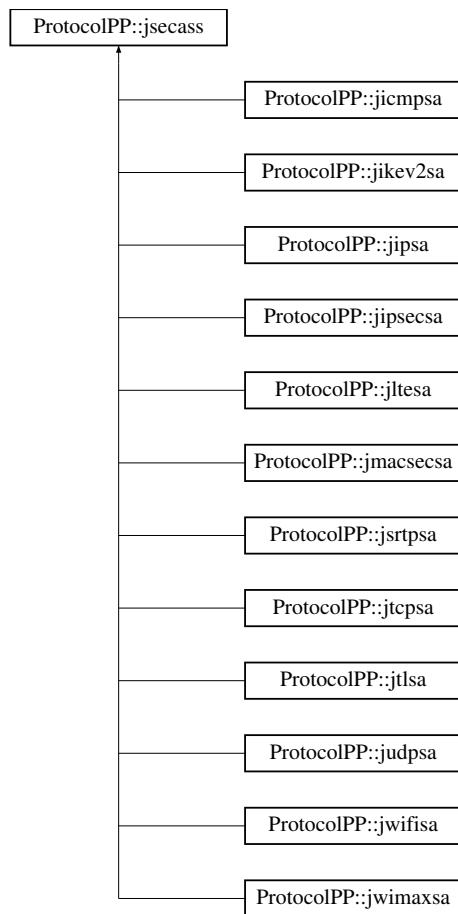
The documentation for this class was generated from the following file:

- platforms/SEC/include/[jsec.h](#)

7.103 ProtocolPP::jsecass Class Reference

```
#include <jsecass.h>
```

Inheritance diagram for ProtocolPP::jsecass:



Public Member Functions

- [jsecass \(\)](#)
- virtual [~jsecass \(\)](#)
Standard deconstructor.
- virtual void [to_xml \(tinyxml2::XMLPrinter &myxml, direction_t direction\)=0](#)

7.103.1 Constructor & Destructor Documentation

7.103.1.1 ProtocolPP::jsecass::jsecass ()

Constructor for base class of security associations which is pure virtual

7.103.1.2 virtual ProtocolPP::jsecass::~jsecass() [inline], [virtual]

Standard deconstructor.

7.103.2 Member Function Documentation

7.103.2.1 virtual void ProtocolPP::jsecass::to_xml (tinyxml2::XMLPrinter & *myxml*, direction_t *direction*) [pure virtual]

print the protocol and security objects as XML

Parameters

<i>myxml</i>	- object to print to
<i>direction</i>	- randomization

Implemented in [ProtocolPP::jwifisa](#), [ProtocolPP::jikev2sa](#), [ProtocolPP::jipsecsa](#), [ProtocolPP::jmacecsa](#), [ProtocolPP::jipsa](#), [ProtocolPP::jsrtpsa](#), [ProtocolPP::jtlsa](#), [ProtocolPP::jltesa](#), [ProtocolPP::jwimaxsa](#), [ProtocolPP::jtcpsa](#), [ProtocolPP::jicmpsa](#), and [ProtocolPP::judpsa](#).

The documentation for this class was generated from the following file:

- [include/jsecass.h](#)

7.104 jsecass Class Reference

```
#include "include/jsecass.h"
```

7.104.1 Detailed Description

This class is the base for all security associations in Protocol++. Access to the different elements in the security associations is by calling the templated function with the type needed. For example, to access a uint8_t type from jipsecsa one would call the function as

- `uint8_t myuint8 = mysecass->get_field<uint8_t>(field_t::TTLHOP);`

To set the same field one would call the function as

- `mysecass->set_field<uint8_t>(field_t::TTLHOP, myuint8);`

Each of the security associations directly return the types needed. To access an ENUM of type direction_t a user would call the jwifisa function as

- `direction_t mydirection = mysecass->get_field<direction_t>(field_t::DIRECTION);`

To set the same field it would be

- `mysecass->set_field<direction_t>(field_t::DIRECTION, mydirection);`

To retrieve the jarray that holds the ARWIN for jtlsa, the call would be

- `jarray<uint8_t> myarwin = mysecass->get_field<jarray<uint8_t>>(field_t::ARWIN);`

To set the ARWIN with a new array it would be

- mysecass->set_field<[jarray<uint8_t>](#)

See [jenum](#) field_t for the field names for all security associations and protocols

For API Documentation:

See Also

[ProtocolPP::tinyxml2](#)
[ProtocolPP::jarray](#)
[ProtocolPP::jrand](#)
[ProtocolPP::jreplay](#)
[ProtocolPP::ciphers](#)
[ProtocolPP::jsecass](#)
[ProtocolPP::jprotocol](#)
[ProtocolPP::judpsa](#)
[ProtocolPP::jtcpssa](#)
[ProtocolPP::jipsa](#)
[ProtocolPP::jicmpsa](#)
[ProtocolPP::jipsecrsa](#)
[ProtocolPP::jmaccsecsa](#)
[ProtocolPP::jsrtpsa](#)
[ProtocolPP::jwifisa](#)
[ProtocolPP::jwimaxsa](#)
[ProtocolPP::jltesa](#)
[ProtocolPP::jtlsa](#)

For Additional Documentation:

See Also

[tinyxml2](#)
[jarray](#)
[jrand](#)
[jreplay](#)
[ciphers](#)
[jsecass](#)
[jprotocol](#)
[judpsa](#)
[jtcpssa](#)
[jipsa](#)
[jicmpsa](#)
[jipsecrsa](#)
[jmaccsecsa](#)
[jsrtpsa](#)
[jwifisa](#)
[jwimaxsa](#)
[jltesa](#)
[jtlsa](#)

Protocol++ written by : John Peter Greninger © John Peter Greninger 2015-2019 All Rights Reserved

All copyrights and trademarks are the property of their respective owners

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer

- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution
- Any and all modifications must be returned to John Peter Greninger at GitHub.com <https://github.com/jpgreninger/protocolpp> for evaluation. Inclusion of modifications in the source code shall be determined solely by John Peter Greninger. Failure to provide modifications shall render this license NULL and VOID and revoke any rights to use of Protocol++
- Commercial use requires a fee-based license obtainable at www.protocolpp.com
- Academic use requires written and notarized permission from John Peter and Sheila Rocha Greninger
- US Copyrights at <https://www.copyright.gov/>
 - TXU002059872 (Version 1.0.0)
 - TXU002066632 (Version 1.2.7)
 - TXU002082674 (Version 1.4.0)
 - TXU002097880 (Version 2.0.0)

The names of its contributors may not be used to endorse or promote products derived from this software without specific prior written permission

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE

The documentation for this class was generated from the following file:

- [include/jsecass.h](#)

7.105 jsecproducer Class Reference

```
#include "include/jsecproducer.h"
```

7.105.1 Detailed Description

7.105.2 Producer for Protocol++ for SEC/CAAM platform

See Also

[jring](#)
[jdata](#)
[jpacket](#)
[iostream](#)
[jsec](#)
[protocolpp](#)

Protocol++ written by : John Peter Greninger © John Peter Greninger 2015-2019 All Rights Reserved

All copyrights and trademarks are the property of their respective owners

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer
- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution
- Any and all modifications must be returned to John Peter Greninger at GitHub.com <https://github.com/jpgreninger/protocolpp> for evaluation. Inclusion of modifications in the source code shall be determined solely by John Peter Greninger. Failure to provide modifications shall render this license NULL and VOID and revoke any rights to use of Protocol++
- Commercial use requires a fee-based license obtainable at www.protocolpp.com
- Academic use requires written and notarized permission from John Peter and Sheila Rocha Greninger
- US Copyrights at <https://www.copyright.gov/>
 - TXU002059872 (Version 1.0.0)
 - TXU002066632 (Version 1.2.7)
 - TXU002082674 (Version 1.4.0)
 - TXU002097880 (Version 2.0.0)

The names of its contributors may not be used to endorse or promote products derived from this software without specific prior written permission

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE

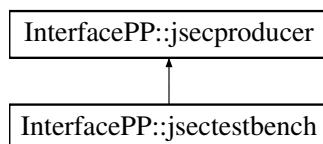
The documentation for this class was generated from the following file:

- [jtestbench/include/jsecproducer.h](#)

7.106 InterfacePP::jsecproducer Class Reference

```
#include <jsecproducer.h>
```

Inheritance diagram for InterfacePP::jsecproducer:



Public Member Functions

- `jsecproducer (ProtocolPP::platform_t platform, unsigned long seed, int responders, std::shared_ptr< jmmu > &mmu, std::shared_ptr< jlogger > &logger, std::shared_ptr< ProtocolPP::jdata > &indata, std::shared_ptr< jring< ringflow >> &fring, std::shared_ptr< jring< secin >> &iring, std::shared_ptr< jring< secout >> &oring, ProtocolPP::endian_t endianess=ProtocolPP::BIG, unsigned int ptrsize=8, unsigned int sgtsize=16)`
- virtual `~jsecproducer ()`
standard deconstructor
- virtual `uint64_t get_mem (std::string name, unsigned int length)=0`
- virtual `void write (uint64_t address, uint32_t data)=0`
- virtual `void write (uint64_t address, uint64_t data)=0`
- virtual `uint64_t write (std::string name, uint8_t *data, unsigned int length)=0`
- virtual `uint64_t write (std::string name, uint32_t *data, unsigned int length)=0`
- virtual `uint32_t read (uint64_t address)=0`
- virtual `void read (uint64_t address, uint8_t *data, unsigned int length)=0`
- virtual `void read (uint64_t address, uint32_t *data, unsigned int length)=0`
- void `setup (std::shared_ptr< ProtocolPP::jstream > &stream)`
- void `issue (std::shared_ptr< ProtocolPP::jpacket > &packet)`
- void `release (std::string stream)`
- int `get_mode ()`
- void `run ()`
run the testbench
- `uint32_t get_status (uint64_t address)`
- `uint64_t get_sgt (unsigned int datasize, uint8_t *data)`

Protected Member Functions

- void `set_mem (std::string packet, uint8_t *ptr)`
run the testbench
- void `free_mem (std::string packet)`
run the testbench

7.106.1 Constructor & Destructor Documentation

7.106.1.1 `InterfacePP::jsecproducer::jsecproducer (ProtocolPP::platform_t platform, unsigned long seed, int responders, std::shared_ptr< jmmu > & mmu, std::shared_ptr< jlogger > & logger, std::shared_ptr< ProtocolPP::jdata > & indata, std::shared_ptr< jring< ringflow >> & fring, std::shared_ptr< jring< secin >> & iring, std::shared_ptr< jring< secout >> & oring, ProtocolPP::endian_t endianess = ProtocolPP::BIG, unsigned int ptrsize = 8, unsigned int sgtsize = 16)`

jsecproducer uses protocol++ to drive packets into a ring which can then be read by any device that supports a ring

Parameters

<code>platform</code>	- Platform to connect testbench to (W.A.S.P or SEC)
<code>seed</code>	- seed for the testbench
<code>responders</code>	- number of responders used
<code>mmu</code>	- dynamic memory tracking
<code>logger</code>	- object to print output
<code>indata</code>	- JDATA object containing flows and packets

<i>fring</i>	- Software ring for the flows
<i>iring</i>	- Software ring for the input packets
<i>oring</i>	- Software ring for the output packets
<i>endianess</i>	- Endianess of the platform to support
<i>ptrsize</i>	- size of pointers in descriptors (4 or 8 bytes)
<i>sgtsize</i>	- size of the Scatter-Gather tables (8 or 16 byte entries)

7.106.1.2 virtual InterfacePP::jsecproducer::~jsecproducer() [inline], [virtual]

standard deconstructor

7.106.2 Member Function Documentation

7.106.2.1 void InterfacePP::jsecproducer::free_mem(std::string *packet*) [protected]

run the testbench

7.106.2.2 virtual uint64_t InterfacePP::jsecproducer::get_mem(std::string *name*, unsigned int *length*) [pure virtual]

Get a chunk of memory

Parameters

<i>name</i>	- name of packet to retrieve data for
<i>length</i>	- Length of memory in bytes

Returns

address to memory

Implemented in [InterfacePP::jsectestbench](#).

7.106.2.3 int InterfacePP::jsecproducer::get_mode()

Returns

- current mode of operation

7.106.2.4 uint64_t InterfacePP::jsecproducer::get_sgt(unsigned int *datasize*, uint8_t * *data*)

Create and return a SG table for SEC/CAAM

Parameters

<i>datasize</i>	- size of the input data found in the pointer
<i>data</i>	- byte pointer to data for SG table

Returns

- address to SG table

7.106.2.5 `uint32_t InterfacePP::jsecproducer::get_status (uint64_t address)`

retrieve the status word

Returns

- status of the indicated packet

7.106.2.6 `void InterfacePP::jsecproducer::issue (std::shared_ptr< ProtocolPP::jpacket > & packet)`

Issues the packet into the testbench or system must be overloaded in user's code

Parameters

<code>packet</code>	- packet to process
---------------------	---------------------

See Also

[jpacket](#)

7.106.2.7 `virtual uint32_t InterfacePP::jsecproducer::read (uint64_t address) [pure virtual]`

Read function to overload in your testbench

Parameters

<code>address</code>	- Address to register
----------------------	-----------------------

Returns

value of the register

Implemented in [InterfacePP::jsectestbench](#).

7.106.2.8 `virtual void InterfacePP::jsecproducer::read (uint64_t address, uint8_t * data, unsigned int length) [pure virtual]`

Read function to overload in your testbench

Parameters

<code>address</code>	- Address to read data from
<code>data</code>	- Retrieved data
<code>length</code>	- Length of data to read

Implemented in [InterfacePP::jsectestbench](#).

7.106.2.9 `virtual void InterfacePP::jsecproducer::read (uint64_t address, uint32_t * data, unsigned int length) [pure virtual]`

Read function to overload in your testbench

Parameters

<i>address</i>	- Address to read data from
<i>data</i>	- Retrieved data
<i>length</i>	- Length of data to read

Implemented in [InterfacePP::jsectestbench](#).

7.106.2.10 void InterfacePP::jsecproducer::release (std::string *stream*)

Stream is done, release the stream and deallocate

Parameters

<i>stream</i>	- name of the stream to releasae
---------------	----------------------------------

7.106.2.11 void InterfacePP::jsecproducer::run ()

run the testbench

7.106.2.12 void InterfacePP::jsecproducer::set_mem (std::string *packet*, uint8_t * *ptr*) [protected]

run the testbench

7.106.2.13 void InterfacePP::jsecproducer::setup (std::shared_ptr< ProtocolPP::jstream > & *stream*)

Setup the stream to process descriptors. Using the security association, program your device to process the flow of packets The security association is accessed as a VOID pointer and must be recast to the correct association type based on the TYPE field in the JSTREAM object

Parameters

<i>stream</i>	- Stream object to setup
---------------	--------------------------

See Also

[jstream](#)

7.106.2.14 virtual void InterfacePP::jsecproducer::write (uint64_t *address*, uint32_t *data*) [pure virtual]

Write function to overload in your testbench

Parameters

<i>address</i>	- Address to write data to
<i>data</i>	- Data to write to memory

Implemented in [InterfacePP::jsectestbench](#).

7.106.2.15 virtual void InterfacePP::jsecproducer::write (uint64_t *address*, uint64_t *data*) [pure virtual]

Write function to overload in your testbench

Parameters

<i>address</i>	- Address to write data to
<i>data</i>	- Data to write to memory

Implemented in [InterfacePP::jsectestbench](#).

7.106.2.16 virtual uint64_t InterfacePP::jsecproducer::write (std::string *name*, uint8_t * *data*, unsigned int *length*) [pure virtual]

Write function to overload in your testbench

Parameters

<i>name</i>	- name of packet to retrieve data for
<i>data</i>	- Data to write to memory
<i>length</i>	- length of data to write

Returns

address - Address data written to

Implemented in [InterfacePP::jsectestbench](#).

7.106.2.17 virtual uint64_t InterfacePP::jsecproducer::write (std::string *name*, uint32_t * *data*, unsigned int *length*) [pure virtual]

Write function to overload in your testbench

Parameters

<i>name</i>	- name of packet to retrieve data for
<i>data</i>	- Word oriented data to write
<i>length</i>	- length of data to write

Returns

address - Address data written to

Implemented in [InterfacePP::jsectestbench](#).

The documentation for this class was generated from the following file:

- jtestbench/include/jsecproducer.h

7.107 jsecresponder Class Reference

```
#include "include/jsecresponder.h"
```

7.107.1 Detailed Description

7.107.2 Responder unit for use in testbench for Protocol++

Protocol++ written by : John Peter Greninger © John Peter Greninger 2015-2019 All Rights Reserved

All copyrights and trademarks are the property of their respective owners

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer
- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution
- Any and all modifications must be returned to John Peter Greninger at GitHub.com <https://github.com/jpgreninger/protocolpp> for evaluation. Inclusion of modifications in the source code shall be determined solely by John Peter Greninger. Failure to provide modifications shall render this license NULL and VOID and revoke any rights to use of Protocol++
- Commercial use requires a fee-based license obtainable at www.protocolpp.com
- Academic use requires written and notarized permission from John Peter and Sheila Rocha Greninger
- US Copyrights at <https://www.copyright.gov/>
 - TXu002059872 (Version 1.0.0)
 - TXu002066632 (Version 1.2.7)
 - TXu002082674 (Version 1.4.0)
 - TXu002097880 (Version 2.0.0)

The names of its contributors may not be used to endorse or promote products derived from this software without specific prior written permission

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE

The documentation for this class was generated from the following file:

- jresponder/include/jsecresponder.h

7.108 InterfacePP::jsecresponder Class Reference

```
#include <jsecresponder.h>
```

Public Member Functions

- **jsecresponder** (std::shared_ptr< **jlogger** > &logger, std::shared_ptr< **jring**< **ringflow** >> &flowring, std::shared_ptr< **jring**< **secin** >> &inputring, std::shared_ptr< **jring**< **secout** >> &outputring, unsigned long seed, std::string readlatency=std::string("80..120"), std::string computelatency=std::string("2000..5000"), std::string writelatency=std::string("50..100"))
- void **dequeue** (std::string flow)
 enqueue flows to responder
- void **enqueue** (std::string &flow)
- void **push** (uint64_t packet, unsigned int size, uint32_t status)
- **secin pop** ()
- void **run** ()
 run the responder

7.108.1 Constructor & Destructor Documentation

7.108.1.1 `InterfacePP::jsecresponder::jsecresponder (std::shared_ptr< jlogger > & logger, std::shared_ptr< jring< ringflow >> & flowring, std::shared_ptr< jring< secin >> & inputring, std::shared_ptr< jring< secout >> & outputring, unsigned long seed, std::string readlatency = std::string("80..120"), std::string computelatency = std::string("2000..5000"), std::string writelatency = std::string("50..100"))`

constructor for responder to test wasp interface

Parameters

<i>logger</i>	- object to print output
<i>flowring</i>	- software ring for the flows
<i>inputring</i>	- software ring for the input packets
<i>outputring</i>	- software ring for the output packets
<i>seed</i>	- seed for the random number generator
<i>readlatency</i>	- latency for read transactions
<i>computelatency</i>	- latency for compute transactions
<i>writelatency</i>	- latency for write transactions

7.108.2 Member Function Documentation

7.108.2.1 `void InterfacePP::jsecresponder::dequeue (std::string flow)`

enqueue flows to responder

7.108.2.2 `void InterfacePP::jsecresponder::enqueue (std::string & flow)`

dequeue flows from responder

Parameters

<i>flow</i>	- flow to dequeue
-------------	-------------------

7.108.2.3 `secin InterfacePP::jsecresponder::pop ()`

pop next packet from the input ring

Returns

- input ring object

7.108.2.4 `void InterfacePP::jsecresponder::push (uint64_t packet, unsigned int size, uint32_t status)`

push another packet to the output ring

Parameters

<i>packet</i>	- address for the packet
<i>size</i>	- size of the packet in bytes
<i>status</i>	- status of the finished packet

7.108.2.5 void InterfacePP::jsecresponder::run()

run the responder

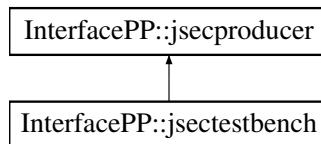
The documentation for this class was generated from the following file:

- jresponder/include/jsecresponder.h

7.109 InterfacePP::jsectestbench Class Reference

```
#include <jsectestbench.h>
```

Inheritance diagram for InterfacePP::jsectestbench:



Public Member Functions

- `jsectestbench (ProtocolPP::platform_t platform, unsigned long seed, int responders, std::shared_ptr< jmmu > &mmu, std::shared_ptr< jlogger > &logger, std::shared_ptr< ProtocolPP::jdata > &indata, std::shared_ptr< jring< ringflow >> &fring, std::shared_ptr< jring< secin >> &iring, std::shared_ptr< jring< sec-out >> &oring, ProtocolPP::endian_t endianess=ProtocolPP::BIG, unsigned int ptrsize=8, unsigned int sgt-size=16)`
- virtual `~jsectestbench ()`
standard deconstructor
- `uint64_t get_mem (std::string packet, unsigned int length)`
- `void write (uint64_t address, uint32_t data)`
- `void write (uint64_t address, uint64_t data)`
- `uint64_t write (std::string packet, uint8_t *data, unsigned int length)`
- `uint64_t write (std::string packet, uint32_t *data, unsigned int length)`
- `uint32_t read (uint64_t address)`
- `void read (uint64_t address, uint8_t *data, unsigned int length)`
- `void read (uint64_t address, uint32_t *data, unsigned int length)`

Additional Inherited Members

7.109.1 Constructor & Destructor Documentation

- 7.109.1.1 `InterfacePP::jsectestbench::jsectestbench (ProtocolPP::platform_t platform, unsigned long seed, int responders, std::shared_ptr< jmmu > & mmu, std::shared_ptr< jlogger > & logger, std::shared_ptr< ProtocolPP::jdata > & indata, std::shared_ptr< jring< ringflow >> & fring, std::shared_ptr< jring< secin >> & iring, std::shared_ptr< jring< secout >> & oring, ProtocolPP::endian_t endianess = ProtocolPP::BIG, unsigned int ptrsize = 8, unsigned int sgtsize = 16)`

jsectestbench uses protocol++ to drive packets into a ring which can then be read by any device that supports a ring

Parameters

<i>platform</i>	- Platform to connect testbench to (W.A.S.P or SEC)
<i>seed</i>	- seed for the testbench
<i>responders</i>	- number of responders used
<i>mmu</i>	- dynamic memory tracking
<i>logger</i>	- object to print output
<i>indata</i>	- JDATA object containing flows and packets
<i>fring</i>	- Software ring for the flows
<i>iring</i>	- Software ring for the input packets
<i>oring</i>	- Software ring for the output packets
<i>endianess</i>	- Endianess of the platform to support
<i>ptrsize</i>	- size of pointer (either 4 or 8 default=8)
<i>sgtsize</i>	- size of SG table entries (either 8 or 16 default=16)

7.109.1.2 virtual InterfacePP::jsectestbench::~jsectestbench() [inline], [virtual]

standard deconstructor

7.109.2 Member Function Documentation

7.109.2.1 uint64_t InterfacePP::jsectestbench::get_mem(std::string *packet*, unsigned int *length*) [virtual]

Retrieve pointer to uninitialized memory

Parameters

<i>packet</i>	- name of packet to associate memory with
<i>length</i>	- Length in bytes for memory location

Returns

address to memory

Implements [InterfacePP::jsecproducer](#).

7.109.2.2 uint32_t InterfacePP::jsectestbench::read(uint64_t *address*) [virtual]

Read function to overload in your testbench

Parameters

<i>address</i>	- Address to register
----------------	-----------------------

Returns

value of the register

Implements [InterfacePP::jsecproducer](#).

7.109.2.3 void InterfacePP::jsectestbench::read(uint64_t *address*, uint8_t * *data*, unsigned int *length*) [virtual]

Read function to overload in your testbench

Parameters

<i>address</i>	- Address to read data from
<i>data</i>	- Retrieved data
<i>length</i>	- Length of data to read

Implements [InterfacePP::jsecproducer](#).

7.109.2.4 void InterfacePP::jsectestbench::read (uint64_t *address*, uint32_t * *data*, unsigned int *length*) [virtual]

Read function to overload in your testbench

Parameters

<i>address</i>	- Address to read data from
<i>data</i>	- Retrieved data
<i>length</i>	- Length of data to read

Implements [InterfacePP::jsecproducer](#).

7.109.2.5 void InterfacePP::jsectestbench::write (uint64_t *address*, uint32_t *data*) [virtual]

Write function to overload in your testbench

Parameters

<i>address</i>	- Address to write data to
<i>data</i>	- Data to write to memory

Implements [InterfacePP::jsecproducer](#).

7.109.2.6 void InterfacePP::jsectestbench::write (uint64_t *address*, uint64_t *data*) [virtual]

Write function to overload in your testbench

Parameters

<i>address</i>	- Address to write data to
<i>data</i>	- Data to write to memory

Implements [InterfacePP::jsecproducer](#).

7.109.2.7 uint64_t InterfacePP::jsectestbench::write (std::string *packet*, uint8_t * *data*, unsigned int *length*) [virtual]

Write function to overload in your testbench

Parameters

<i>packet</i>	- name of packet to associate memory with
<i>data</i>	- Data to write to memory
<i>length</i>	- length of data to write

Implements [InterfacePP::jsecproducer](#).

7.109.2.8 uint64_t InterfacePP::jsectestbench::write (std::string *packet*, uint32_t * *data*, unsigned int *length*) [virtual]

Write function to overload in your testbench

Parameters

<i>packet</i>	- name of packet to associate memory with
<i>data</i>	- Word oriented data to write
<i>length</i>	- length of data to write

Implements [InterfacePP::jsecproducer](#).

The documentation for this class was generated from the following file:

- jtestbench/include/jsectestbench.h

7.110 jsectestbench Class Reference

```
#include "include/jsectestbench.h"
```

7.110.1 Detailed Description

7.110.2 Testbench for Protocol++

See Also

[jring](#)
[jdata](#)
[jpacket](#)
[jstream](#)
[protocolpp](#)

Protocol++ written by : John Peter Greninger © John Peter Greninger 2015-2019 All Rights Reserved

All copyrights and trademarks are the property of their respective owners

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer
- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution
- Any and all modifications must be returned to John Peter Greninger at GitHub.com <https://github.com/jpgreninger/protocolpp> for evaluation. Inclusion of modifications in the source code shall be determined solely by John Peter Greninger. Failure to provide modifications shall render this license NULL and VOID and revoke any rights to use of Protocol++
- Commercial use requires a fee-based license obtainable at www.protocolpp.com
- Academic use requires written and notarized permission from John Peter and Sheila Rocha Greninger
- US Copyrights at <https://www.copyright.gov/>
 - TXu002059872 (Version 1.0.0)
 - TXu002066632 (Version 1.2.7)
 - TXu002082674 (Version 1.4.0)
 - TXu002097880 (Version 2.0.0)

The names of its contributors may not be used to endorse or promote products derived from this software without specific prior written permission

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE

The documentation for this class was generated from the following file:

- [jtestbench/include/jsectestbench.h](#)

7.111 jsgt Class Reference

```
#include "include/jsgt.h"
```

7.111.1 Detailed Description

7.111.2 Scatter Gather Table (SGT)

This class will create a Scatter Gather table suitable for use with the QorIQ and Layerscape Network Processors from Freescale/NXP/Qualcomm. See the documentation for your respective processor on the NXP/Qualcomm website

Protocol++ written by : John Peter Greninger © John Peter Greninger 2015-2019 All Rights Reserved

All copyrights and trademarks are the property of their respective owners

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer
- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution
- Any and all modifications must be returned to John Peter Greninger at GitHub.com <https://github.com/jpgreninger/protocolpp> for evaluation. Inclusion of modifications in the source code shall be determined solely by John Peter Greninger. Failure to provide modifications shall render this license NULL and VOID and revoke any rights to use of Protocol++
- Commercial use requires a fee-based license obtainable at www.protocolpp.com
- Academic use requires written and notarized permission from John Peter and Sheila Rocha Greninger
- US Copyrights at <https://www.copyright.gov/>
 - TXu002059872 (Version 1.0.0)
 - TXu002066632 (Version 1.2.7)
 - TXu002082674 (Version 1.4.0)
 - TXu002097880 (Version 2.0.0)

The names of its contributors may not be used to endorse or promote products derived from this software without specific prior written permission

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE

The documentation for this class was generated from the following file:

- platforms/SEC/include/jsgt.h

7.112 PlatformPP::jsgt Class Reference

```
#include <jsgt.h>
```

Public Member Functions

- [sgt_t \(\)](#)
- [jsgt \(sgt_t *sgt\)](#)
- virtual [~jsgt \(\)](#)
- void [get \(unsigned int size, uint8_t *data, uint8_t *sgtout\)](#)

Static Public Member Functions

- static void [read \(unsigned int sgtsize, uint8_t *sgtin, uint8_t *data\)](#)

Public Attributes

- struct [sgt_t](#)
- std::string [chunkszie](#)
- unsigned int [addrszie](#)
- unsigned int [extend](#)
- unsigned int [sgtfinal](#)
- unsigned int [length](#)
- unsigned int [bpid](#)
- unsigned int [offset](#)
- unsigned int [entrysize](#)
- std::vector< std::string > [order](#)

7.112.1 Constructor & Destructor Documentation

7.112.1.1 PlatformPP::jsgt::jsgt (sgt_t * sgt)

7.112.1.2 virtual PlatformPP::jsgt::~jsgt () [inline], [virtual]

7.112.2 Member Function Documentation

```

7.112.2.1 uint64_t jsgt::get( unsigned int size, uint8_t * data, uint8_t * sgtout )

7.112.2.2 static void jsgt::read( unsigned int sgtsize, uint8_t * sgtin, uint8_t * data ) [static]

7.112.2.3 PlatformPP::jsgt::sgt_t( ) [inline]

```

7.112.3 Member Data Documentation

```

7.112.3.1 unsigned int PlatformPP::jsgt::addrsize

7.112.3.2 unsigned int PlatformPP::jsgt::bpid

7.112.3.3 std::string PlatformPP::jsgt::chunksize

7.112.3.4 unsigned int PlatformPP::jsgt::entrysize

7.112.3.5 unsigned int PlatformPP::jsgt::extend

7.112.3.6 unsigned int PlatformPP::jsgt::length

7.112.3.7 unsigned int PlatformPP::jsgt::offset

7.112.3.8 std::vector<std::string> PlatformPP::jsgt::order

7.112.3.9 struct PlatformPP::jsgt::sgt_t

```

Initial value:

```
{
    platform_t platform
}
```

7.112.3.10 unsigned int PlatformPP::jsgt::sgtfinal

The documentation for this class was generated from the following file:

- platforms/SEC/include/jsgt.h

7.113 jsnow3g Class Reference

```
#include "include/jsnow3g.h"
```

7.113.1 Detailed Description

7.113.2 Snow3G Protocol

Code derived from the 3GPP specification found below

ETSI/SAGE Specification

Version: 1.1 Date: 6th September 2006

Specification of the 3GPP Confidentiality and Integrity Algorithms UEA2 & UIA2. Document 1: UEA2 and UIA2 Specification

Protocol++ written by : John Peter Greninger © John Peter Greninger 2015-2019 All Rights Reserved

All copyrights and trademarks are the property of their respective owners

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer
- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution
- Any and all modifications must be returned to John Peter Greninger at GitHub.com <https://github.com/jpgreninger/protocolpp> for evaluation. Inclusion of modifications in the source code shall be determined solely by John Peter Greninger. Failure to provide modifications shall render this license NULL and VOID and revoke any rights to use of Protocol++
- Commercial use requires a fee-based license obtainable at www.protocolpp.com
- Academic use requires written and notarized permission from John Peter and Sheila Rocha Greninger
- US Copyrights at <https://www.copyright.gov/>
 - TXu002059872 (Version 1.0.0)
 - TXu002066632 (Version 1.2.7)
 - TXu002082674 (Version 1.4.0)
 - TXu002097880 (Version 2.0.0)

The names of its contributors may not be used to endorse or promote products derived from this software without specific prior written permission

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE

The documentation for this class was generated from the following file:

- [include/jsnow3g.h](#)

7.114 ProtocolPP::jsnow3g Class Reference

```
#include <jsnow3g.h>
```

Public Types

- enum [dir_t](#) { [UPLINK](#), [DOWNLINK](#) }

Public Member Functions

- `jsnow3g (dir_t dir, uint8_t *key, unsigned int keylen, uint32_t count, uint8_t bearer)`
 - `jsnow3g (dir_t dir, uint8_t *key, unsigned int keylen, uint32_t count, uint32_t fresh)`
 - virtual `~jsnow3g ()`
- Standard deconstructor.*
- void `ProcessData (const uint8_t *input, uint8_t *output, int length)`
 - void `ProcessData (const uint8_t *input, uint64_t length)`
 - void `context (uint32_t *context)`
 - void `result (uint8_t *icv)`
 - unsigned int `result_size ()`

7.114.1 Member Enumeration Documentation

7.114.1.1 enum ProtocolPP::jsnow3g::dir_t

Direction of processing for SNOW3G UPLINK - User to tower DOWNLINK - Tower to user

Enumerator

UPLINK

DOWNLINK

7.114.2 Constructor & Destructor Documentation

7.114.2.1 ProtocolPP::jsnow3g::jsnow3g (`dir_t dir, uint8_t *key, unsigned int keylen, uint32_t count, uint8_t bearer`)

Interface for Snow-3G encryption

Parameters

<code>dir</code>	- Direction of the data
<code>key</code>	- Key for encryption of the data
<code>keylen</code>	- Key length for encryption of the data
<code>count</code>	- Count value to generate key stream
<code>bearer</code>	- Bearer value for initialize the cipher

7.114.2.2 ProtocolPP::jsnow3g::jsnow3g (`dir_t dir, uint8_t *key, unsigned int keylen, uint32_t count, uint32_t fresh`)

Interface for Snow-3G authentication

Parameters

<code>dir</code>	- Direction of the data
<code>key</code>	- Key for authentication of the data
<code>keylen</code>	- Key length for authentication of the data
<code>count</code>	- Count value to generate key stream
<code>fresh</code>	- Fresh value for initialize the authenticator

7.114.2.3 virtual ProtocolPP::jsnow3g::~jsnow3g () [inline], [virtual]

Standard deconstructor.

7.114.3 Member Function Documentation

7.114.3.1 void ProtocolPP::jsnow3g::context (*uint32_t * context*)

Retrieves the context of the engine

Returns

icv - result of the authentication process

7.114.3.2 void ProtocolPP::jsnow3g::ProcessData (*const uint8_t * input, uint8_t * output, int length*)

Calculates either the ciphertext or plaintext based on dir using a bitlength that is passed to the function

Parameters

<i>input</i>	- either the ciphertext or plaintext to be calculated
<i>output</i>	- result of encryption process
<i>length</i>	- bit length

7.114.3.3 void ProtocolPP::jsnow3g::ProcessData (*const uint8_t * input, uint64_t length*)

Authenticates the input

Parameters

<i>input</i>	- data to authenticate
<i>length</i>	- bit length

7.114.3.4 void ProtocolPP::jsnow3g::result (*uint8_t * icv*)

Retrieves the result of the authentication

Returns

icv - result of the authentication process

7.114.3.5 unsigned int ProtocolPP::jsnow3g::result_size () [inline]

Size of the result

Returns

size of the result or ICV

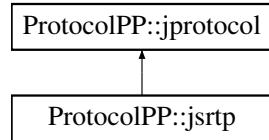
The documentation for this class was generated from the following file:

- [include/jsnow3g.h](#)

7.115 ProtocolPP::jsrtp Class Reference

```
#include <jsrtp.h>
```

Inheritance diagram for ProtocolPP::jsrtp:



Public Member Functions

- `jsrtp (std::shared_ptr< jrand > &rand, std::shared_ptr< jsrtpsa > &security)`
- virtual `~jsrtp ()`
Standard deconstructor.
- void `encap_packet (std::shared_ptr< jarray< uint8_t >> &input, std::shared_ptr< jarray< uint8_t >> &output)`
- void `decap_packet (std::shared_ptr< jarray< uint8_t >> &input, std::shared_ptr< jarray< uint8_t >> &output)`
- void `set_hdr (jarray< uint8_t > &hdr)`
- void `set_exthdr (jarray< uint8_t > &exthdr)`
- void `set_field (field_t field, uint64_t value)`
- void `set_field (field_t field, uint64_t value, unsigned int index)`
- `jarray< uint8_t > get_hdr ()`
- `jarray< uint8_t > get_exthdr ()`
- `uint64_t get_field (field_t field, jarray< uint8_t > &header)`
- `uint64_t get_field (field_t field, jarray< uint8_t > &header, unsigned int index)`
- void `to_xml (tinyxml2::XMLPrinter &myxml, direction_t direction)`

Additional Inherited Members

7.115.1 Constructor & Destructor Documentation

7.115.1.1 `ProtocolPP::jsrtp::jsrtp (std::shared_ptr< jrand > & rand, std::shared_ptr< jsrtpsa > & security)`

constructor for SRTP

Parameters

<code>rand</code>	- random data generator for padding
<code>security</code>	- security association for SRTP

7.115.1.2 `virtual ProtocolPP::jsrtp::~jsrtp () [inline], [virtual]`

Standard deconstructor.

7.115.2 Member Function Documentation

7.115.2.1 `void ProtocolPP::jsrtp::decap_packet (std::shared_ptr< jarray< uint8_t >> & input, std::shared_ptr< jarray< uint8_t >> & output) [virtual]`

This function is for use with the constructor without a file handle. Decap will produce a payload from the packet passed in.

Parameters

<i>input</i>	- SRTP encapsulated packet to decapsulate
<i>output</i>	- decapsulated payload

Implements [ProtocolPP::jprotocol](#).

7.115.2.2 void ProtocolPP::jsrtp::encap_packet (std::shared_ptr< jarray< uint8_t >> &*input*, std::shared_ptr< jarray< uint8_t >> &*output*) [virtual]

This function is for use with the constructor without a file handle. Encap will produce a packet with the payload passed in

Parameters

<i>input</i>	- payload to encapsulate with SRTP
<i>output</i>	- SRTP encapsulated packet

Implements [ProtocolPP::jprotocol](#).

7.115.2.3 jarray<uint8_t> ProtocolPP::jsrtp::get_exthdr ()

Returns the complete extension header

Returns

current extension header

7.115.2.4 uint64_t ProtocolPP::jsrtp::get_field (field_t *field*, jarray< uint8_t > &*header*) [virtual]

Returns the field of the SRTP header

Parameters

<i>field</i>	- field to retrieve from the SRTP header
<i>header</i>	- SRTP header to retrieve the field from

Returns

field of the SRTP header

Implements [ProtocolPP::jprotocol](#).

7.115.2.5 uint64_t ProtocolPP::jsrtp::get_field (field_t *field*, jarray< uint8_t > &*header*, unsigned int *index*)

Returns the field of the SRTP header

Parameters

<i>field</i>	- field to retrieve from the SRTP header
<i>header</i>	- SRTP header to retrieve the field from
<i>index</i>	- Only used for CSRC field to retrieve additional CSRC fields past the first one

Returns

field of the SRTP header

7.115.2.6 `jarray<uint8_t> ProtocolPP::jsrtp::get_hdr() [virtual]`

Returns the complete SRTP header

Returns

current SRTP header

Implements [ProtocolPP::jprotocol](#).

7.115.2.7 `void ProtocolPP::jsrtp::set_exthdr(jarray< uint8_t > & exthdr)`

Allows the user to update the SRTP header with a single extension header

Parameters

<i>exthdr</i>	- extension header to use
---------------	---------------------------

7.115.2.8 `void ProtocolPP::jsrtp::set_field(field_t field, uint64_t value) [virtual]`

Allows the user to update the field of the SRTP header

Parameters

<i>field</i>	- field to update the SRTP header with
<i>value</i>	- value to update the SRTP header with

Implements [ProtocolPP::jprotocol](#).

7.115.2.9 `void ProtocolPP::jsrtp::set_field(field_t field, uint64_t value, unsigned int index)`

Allows the user to update the field of the SRTP header

Parameters

<i>field</i>	- field to update the SRTP header with
<i>value</i>	- value to update the SRTP header with
<i>index</i>	- Index into the CSRC field of the SRTP header

7.115.2.10 `void ProtocolPP::jsrtp::set_hdr(jarray< uint8_t > & hdr) [virtual]`

Allows the user to update the SRTP header with their own header Note : This function will NOT save the previous header. If it is desired to keep the current header the user would need to use the [get_hdr\(\)](#) function and save the current header before updating the SRTP header with a custom header. The header would then need to be restored when the custom header is no longer needed

Parameters

<i>hdr</i>	- new SRTP header to use
------------	--------------------------

Implements [ProtocolPP::jprotocol](#).

7.115.2.11 `void ProtocolPP::jsrtp::to_xml(tinyxml2::XMLPrinter & myxml, direction_t direction) [virtual]`

Prints the protocol and security objects to XML

Parameters

<i>myxml</i>	- XMLPrinter object
<i>direction</i>	- randomization

Implements [ProtocolPP::jprotocol](#).

The documentation for this class was generated from the following file:

- [include/jsrtp.h](#)

7.116 jsrtp Class Reference

```
#include "include/jsrtp.h"
```

7.116.1 Detailed Description

7.116.2 Secure Real-Time Protocol (SRTP)

SRTP Framework

RTP is the Real-time Transport Protocol [RFC3550]. We define SRTP as a profile of RTP. This profile is an extension to the RTP Audio/Video Profile [RFC3551]. Except where explicitly noted, all aspects of that profile apply, with the addition of the SRTP security features. Conceptually, we consider SRTP to be a "bump in the stack" implementation which resides between the RTP application and the transport layer. SRTP intercepts RTP packets and then forwards an equivalent SRTP packet on the sending side, and intercepts SRTP packets and passes an equivalent RTP packet up the stack on the receiving side.

Secure RTCP (SRTCP) provides the same security services to RTCP as SRTP does to RTP. SRTCP message authentication is MANDATORY and thereby protects the RTCP fields to keep track of membership, provide feedback to RTP senders, or maintain packet sequence counters. SRTCP is described in Section 3.4.

Secure RTP (see IETF specifications RFC3711, RFC7741)

The format of an SRTP packet is illustrated in Figure 1 [1]

Bit	+0..7				+8..15		+16..23		+24..31							
0	V	P	X	CC	M	Payload Type				Sequence number						
32	Timestamp															
64	Synchronization source (SSRC) identifier															
+32	Contributing source (CSRC) identifier (optional)															
+32	RTP extension (optional)															
+N	Encrypted RTP payload ...															
...	SRTP MKI (optional) + Authentication Tag (optional)															

Figure 7.104: Secure Real-Time Protocol (SRTP) Packet Structure

The first twelve octets are present in every RTP packet, while the list of CSRC identifiers is present only when inserted by a mixer. The fields have the following meaning:

- version (V): 2 bits

This eld identifies the version of RTP. The version dened by this specication is two (2). (The value 1 is used by the rst draft version of RTP and the value 0 is used by the protocol initially implemented in the “vat” audio tool.)

- padding (P): 1 bit

If the padding bit is set, the packet contains one or more additional padding octets at the end which are not part of the payload. The last octet of the padding contains a count of how many padding octets should be ignored, including itself. Padding may be needed by some encryption algorithms with xed block sizes or for carrying several RTP packets in a lower-layer protocol data unit.

- extension (X): 1 bit

If the extension bit is set, the xed header must be followed by exactly one header extension, with a format dened in Section 5.3.1.

- CSRC count (CC): 4 bits

The CSRC count contains the number of CSRC identiers that follow the xed header.

- marker (M): 1 bit

The interpretation of the marker is dened by a prole. It is intended to allow signicant events such as frame boundaries to be marked in the packet stream. A prole may dene additional marker bits or specify that there is no marker bit by changing the number of bits in the payload type eld (see Section 5.3).

- payload type (PT): 7 bits

This eld identies the format of the RTP payload and determines its interpretation by the application. A prole may specify a default static mapping of payload type codes to payload formats. Additional payload type codes may be dened dynamically through non-RTP means (see Section 3). A set of default mappings for audio and video is specied in the companion RFC 3551 [1]. An RTP source may change the payload type during a session, but this eld should not be used for multiplexing separate media streams (see Section 5.2). A receiver must ignore packets with payload types that it does not understand.

- sequence number: 16 bits

The sequence number increments by one for each RTP data packet sent, and may be used by the receiver to detect packet loss and to restore packet sequence. The initial value of the sequence number should be random (unpredictable) to make known-plaintext attacks on encryption more dicult, even if the source itself does not encrypt according to the method in Section 9.1, because the packets may ow through a translator that does. Techniques for choosing unpredictable numbers are discussed in [17].

- timestamp: 32 bits

The timestamp reects the sampling instant of the rst octet in the RTP data packet. The sampling instant must be derived from a clock that increments monotonically and linearly in time to allow synchronization and jitter calculations (see Section 6.4.1). The resolution of the clock must be suient for the desired synchronization accuracy and for measuring packet arrival jitter (one tick per video frame is typically not suient). The clock frequency is dependent on the format of data carried as payload and is specied statically in the prole or payload format specication that denes the format, or may be specied dynamically for payload formats dened through non-RTP means. If RTP packets are generated periodically, the nominal sampling instant as determined from the sampling clock is to be used, not a reading of the system clock. As an example, for xed-rate audio the timestamp clock would likely increment by one for each sampling period. If an audio application reads blocks covering 160 sampling periods from the input device, the timestamp would be increased by 160 for each such block, regardless of whether the block is transmitted in a packet or dropped as silent. The initial value of the timestamp should be random, as for the sequence number. Several consecutive RTP packets will have equal timestamps if they are (logically) generated at once, e.g., belong to the same video frame. Consecutive RTP packets may contain timestamps that are not monotonic if the data is not transmitted in the order it was sampled, as in the case of MPEG interpolated video frames. (The sequence numbers of the packets as transmitted will still be monotonic.) RTP timestamps from dierent media streams may advance at dierent rates and usually have independent, random oses. Therefore, although these timestamps are suient to reconstruct the timing of a single stream, directly comparing RTP timestamps from dierent media is not efective for synchronization. Instead, for each medium the RTP timestamp is related to the sampling instant by pairing it with a timestamp from a reference clock (wallclock) that represents the time when the data corresponding to the RTP timestamp was sampled. The reference clock is shared by all media to be synchronized.

The timestamp pairs are not transmitted in every data packet, but at a lower rate in RTCP SR packets as described in Section 6.4. The sampling instant is chosen as the point of reference for the RTP timestamp because it is known to the transmitting endpoint and has a common definition for all media, independent of encoding delays or other processing. The purpose is to allow synchronized presentation of all media sampled at the same time. Applications transmitting stored data rather than data sampled in real time typically use a virtual presentation timeline derived from wallclock time to determine when the next frame or other unit of each medium in the stored data should be presented. In this case, the RTP timestamp would reflect the presentation time for each unit. That is, the RTP timestamp for each unit would be related to the wallclock time at which the unit becomes current on the virtual presentation timeline. Actual presentation occurs some time later as determined by the receiver. An example describing live audio narration of prerecorded video illustrates the significance of choosing the sampling instant as the reference point. In this scenario, the video would be presented locally for the narrator to view and would be simultaneously transmitted using RTP. The "sampling instant" of a video frame transmitted in RTP would be established by referencing its timestamp to the wallclock time when that video frame was presented to the narrator. The sampling instant for the audio RTP packets containing the narrator's speech would be established by referencing the same wallclock time when the audio was sampled. The audio and video may even be transmitted by different hosts if the reference clocks on the two hosts are synchronized by some means such as NTP. A receiver can then synchronize presentation of the audio and video packets by relating their RTP timestamps using the timestamp pairs in RTCP SR packets.

- SSRC: 32 bits

The SSRC field identifies the synchronization source. This identifier should be chosen randomly, with the intent that no two synchronization sources within the same RTP session will have the same SSRC identifier. An example algorithm for generating a random identifier is presented in Appendix A.6. Although the probability of multiple sources choosing the same identifier is low, all RTP implementations must be prepared to detect and resolve collisions. Section 8 describes the probability of collision along with a mechanism for resolving collisions and detecting RTP-level forwarding loops based on the uniqueness of the SSRC identifier. If a source changes its source transport address, it must also choose a new SSRC identifier to avoid being interpreted as a looped source (see Section 8.2).

- CSRC list: 0 to 15 items, 32 bits each

The CSRC list identifies the contributing sources for the payload contained in this packet. The number of identifiers is given by the CC field. If there are more than 15 contributing sources, only 15 can be identified. CSRC identifiers are inserted by mixers (see Section 7.1), using the SSRC identifiers of contributing sources. For example, for audio packets the SSRC identifiers of all sources that were mixed together to create a packet are listed, allowing correct talker indication at the receiver.

The "Encrypted Portion" of an SRTP packet consists of the encryption of the RTP payload (including RTP padding when present) of the equivalent RTP packet. The Encrypted Portion MAY be the exact size of the plaintext or MAY be larger. Figure 1 shows the RTP payload including any possible padding for RTP [RFC3550].

None of the pre-defined encryption transforms uses any padding; for these, the RTP and SRTP payload sizes match exactly. New transforms added to SRTP (following Section 6) may require padding, and may hence produce larger payloads. RTP provides its own padding format (as seen in Fig. 1), which due to the padding indicator in the RTP header has merits in terms of compactness relative to paddings using prefix-free codes. This RTP padding SHALL be the default method for transforms requiring padding. Transforms MAY specify other padding methods, and MUST then specify the amount, format, and processing of their padding. It is important to note that encryption transforms that use padding are vulnerable to subtle attacks, especially when message authentication is not used [V02]. Each specification for a new encryption transform needs to carefully consider and describe the security implications of the padding that it uses. Message authentication codes define their own padding, so this default does not apply to authentication transforms.

The OPTIONAL MKI and the RECOMMENDED authentication tag are the only fields defined by SRTP that are not in RTP. Only 8-bit alignment is assumed.

- MKI (Master Key Identifier): configurable length, OPTIONAL. The MKI is defined, signaled, and used by key management. The MKI identifies the master key from which the session key(s) were derived that authenticate and/or encrypt the particular packet. Note that the MKI SHALL NOT identify the SRTP cryptographic context, which is identified according to Section 3.2.3. The MKI MAY be used by key management for the purposes of re-keying, identifying a particular master key within the cryptographic context (Section 3.2.1).

- Authentication tag: configurable length, RECOMMENDED. The authentication tag is used to carry message authentication data. The Authenticated Portion of an SRTP packet consists of the RTP header followed by the Encrypted Portion of the SRTP packet. Thus, if both encryption and authentication are applied, encryption SHALL be applied before authentication on the sender side and conversely on the receiver side. The authentication tag provides authentication of the RTP header and payload, and it indirectly provides replay protection by authenticating the sequence number. Note that the MKI is not integrity protected as this does not provide any extra protection.

SRTP Cryptographic Contexts

Each SRTP stream requires the sender and receiver to maintain cryptographic state information. This information is called the "cryptographic context". SRTP uses two types of keys: session keys and master keys. By a "session key", we mean a key which is used directly in a cryptographic transform (e.g., encryption or message authentication), and by a "master key", we mean a random bit string (given by the key management protocol) from which session keys are derived in a cryptographically secure way. The master key(s) and other parameters in the cryptographic context are provided by key management mechanisms external to SRTP, see Section 8.

Transform-independent parameters

Transform-independent parameters are present in the cryptographic context independently of the particular encryption or authentication transforms that are used. The transform-independent parameters of the cryptographic context for SRTP consist of:

- a 32-bit unsigned rollover counter (ROC), which records how many times the 16-bit RTP sequence number has been reset to zero after passing through 65,535. Unlike the sequence number (SEQ), which SRTP extracts from the RTP packet header, the ROC is maintained by SRTP as described in Section 3.3.1.

We define the index of the SRTP packet corresponding to a given ROC and RTP sequence number to be the 48-bit quantity

$$i = 2^{16} \oplus ROC + SEQ$$

- for the receiver only, a 16-bit sequence number s_l, which can be thought of as the highest received RTP sequence number (see Section 3.3.1 for its handling), which SHOULD be authenticated since message authentication is RECOMMENDED,
- an identifier for the encryption algorithm, i.e., the cipher and its mode of operation,
- an identifier for the message authentication algorithm,
- a replay list, maintained by the receiver only (when authentication and replay protection are provided), containing indices of recently received and authenticated SRTP packets,
- an MKI indicator (0/1) as to whether an MKI is present in SRTP and SRTCP packets,
- if the MKI indicator is set to one, the length (in octets) of the MKI field, and (for the sender) the actual value of the currently active MKI (the value of the MKI indicator and length MUST be kept fixed for the lifetime of the context),
- the master key(s), which MUST be random and kept secret,
- for each master key, there is a counter of the number of SRTP packets that have been processed (sent) with that master key (essential for security, see Sections 3.3.1 and 9),
- non-negative integers n_e, and n_a, determining the length of the session keys for encryption, and message authentication.

In addition, for each master key, an SRTP stream MAY use the following associated values:

- a master salt, to be used in the key derivation of session keys. This value, when used, MUST be random, but MAY be public. Use of master salt is strongly RECOMMENDED, see Section 9.2. A "NULL" salt is treated as 0...0.

- an integer in the set {1,2,4,..., 2^{24} }, the "key_derivation_rate", where an unspecified value is treated as zero. The constraint to be a power of 2 simplifies the session-key derivation implementation, see Section 4.3.
- an MKI value,
- <From, To> values, specifying the lifetime for a master key, expressed in terms of the two 48-bit index values inside whose range (including the range end-points) the master key is valid. For the use of <From, To>, see Section 8.1.1. <From, To> is an alternative to the MKI and assumes that a master key is in one to-one correspondence with the SRTP session key on which the <From, To> range is defined.

' SRTCP SHALL by default share the crypto context with SRTP, except:

- no rollover counter and s_l-value need to be maintained as the RTCP index is explicitly carried in each SRTCP packet,
- a separate replay list is maintained (when replay protection is provided),
- SRTCP maintains a separate counter for its master key (even if the master key is the same as that for SRTP, see below), as a means to maintain a count of the number of SRTCP packets that have been processed with that key.

Note in particular that the master key(s) MAY be shared between SRTP and the corresponding SRTCP, if the pre-defined transforms (including the key derivation) are used but the session key(s) MUST NOT be so shared.

In addition, there can be cases (see Sections 8 and 9.1) where several SRTP streams within a given RTP session, identified by their synchronization source (SSRCs, which is part of the RTP header), share most of the crypto context parameters (including possibly master and session keys). In such cases, just as in the normal SRTP/S-RTCP parameter sharing above, separate replay lists and packet counters for each stream (SSRC) MUST still be maintained.

Also, separate SRTP indices MUST then be maintained. A summary of parameters, pre-defined transforms, and default values for the above parameters (and other SRTP parameters) can be found in Sections 5 and 8.2.

7.116.2.1 SRTP Packet Processing

Assuming initialization of the cryptographic context(s) has taken place via key management, the sender SHALL do the following to construct an SRTP packet:

1. Determine which cryptographic context to use as described in Section 3.2.3.
2. Determine the index of the SRTP packet using the rollover counter, the highest sequence number in the cryptographic context, and the sequence number in the RTP packet, as described in Section 3.3.1.
3. Determine the master key and master salt. This is done using the index determined in the previous step or the current MKI in the cryptographic context, according to Section 8.1.
4. Determine the session keys and session salt (if they are used by the transform) as described in Section 4.3, using master key, master salt, key_derivation_rate, and session key-lengths in the cryptographic context with the index, determined in Steps 2 and 3.
5. Encrypt the RTP payload to produce the Encrypted Portion of the packet (see Section 4.1, for the defined ciphers). This step uses the encryption algorithm indicated in the cryptographic context, the session encryption key and the session salt (if used) found in Step 4 together with the index found in Step 2.
6. If the MKI indicator is set to one, append the MKI to the packet.
7. For message authentication, compute the authentication tag for the Authenticated Portion of the packet, as described in Section 4.2. This step uses the current rollover counter, the authentication algorithm indicated in the cryptographic context, and the session authentication key found in Step 4. Append the authentication tag to the packet.
8. If necessary, update the ROC as in Section 3.3.1, using the packet index determined in Step 2.

To authenticate and decrypt an SRTP packet, the receiver SHALL do the following:

1. Determine which cryptographic context to use as described in Section 3.2.3.
2. Run the algorithm in Section 3.3.1 to get the index of the SRTP packet. The algorithm uses the rollover counter and highest sequence number in the cryptographic context with the sequence number in the SRTP packet, as described in Section 3.3.1.
3. Determine the master key and master salt. If the MKI indicator in the context is set to one, use the MKI in the SRTP packet, otherwise use the index from the previous step, according to Section 8.1.
4. Determine the session keys, and session salt (if used by the transform) as described in Section 4.3, using master key, master salt, key_derivation_rate and session key-lengths in the cryptographic context with the index, determined in Steps 2 and 3.
5. For message authentication and replay protection, first check if the packet has been replayed (Section 3.3.2), using the Replay List and the index as determined in Step 2. If the packet is judged to be replayed, then the packet MUST be discarded, and the event SHOULD be logged.

Next, perform verification of the authentication tag, using the rollover counter from Step 2, the authentication algorithm indicated in the cryptographic context, and the session authentication key from Step 4. If the result is "AUTHENTICATION FAILURE" (see Section 4.2), the packet MUST be discarded from further processing and the event SHOULD be logged.

6. Decrypt the Encrypted Portion of the packet (see Section 4.1, for the defined ciphers), using the decryption algorithm indicated in the cryptographic context, the session encryption key and salt (if used) found in Step 4 with the index from Step 2.
7. Update the rollover counter and highest sequence number, s_l , in the cryptographic context as in Section 3.3.1, using the packet index estimated in Step 2. If replay protection is provided, also update the Replay List as described in Section 3.3.2.
8. When present, remove the MKI and authentication tag fields from the packet.

Packet Index Determination, and ROC, s_l Update

SRTP implementations use an "implicit" packet index for sequencing, i.e., not all of the index is explicitly carried in the SRTP packet. For the pre-defined transforms, the index i is used in replay protection (Section 3.3.2), encryption (Section 4.1), message authentication (Section 4.2), and for the key derivation (Section 4.3).

When the session starts, the sender side MUST set the rollover counter, ROC, to zero. Each time the RTP sequence number, SEQ, wraps modulo 2^{16} , the sender side MUST increment ROC by one, modulo 2^{32} (see security aspects below). The sender's packet index is then defined as

$$i = 2^{16} \oplus ROC + SEQ$$

Receiver-side implementations use the RTP sequence number to determine the correct index of a packet, which is the location of the packet in the sequence of all SRTP packets. A robust approach for the proper use of a rollover counter requires its handling and use to be well defined. In particular, out-of-order RTP packets with sequence numbers close to 2^{16} or zero must be properly handled.

The index estimate is based on the receiver's locally maintained ROC and s_l values. At the setup of the session, the ROC MUST be set to zero. Receivers joining an on-going session MUST be given the current ROC value using out-of-band signaling such as key-management signaling. Furthermore, the receiver SHALL initialize s_l to the RTP sequence number (SEQ) of the first observed SRTP packet (unless the initial value is provided by out of band signaling such as key management).

On consecutive SRTP packets, the receiver SHOULD estimate the index as

$$i = 2^{16} \oplus v + SEQ$$

where v is chosen from the set { ROC-1, ROC, ROC+1 } (modulo 2^{32}) such that i is closest (in modulo 2^{48} sense) to the value $2^{16} \oplus ROC + s_l$ (see Appendix A for pseudocode).

After the packet has been processed and authenticated (when enabled for SRTP packets for the session), the receiver MUST use v to conditionally update its s_l and ROC variables as follows. If $v = (ROC - 1) \bmod 2^{32}$, then there is no update to s_l or ROC. If $v=ROC$, then s_l is set to SEQ if and only if SEQ is larger than the current s_l ; there is no change to ROC. If $v = (ROC + 1) \bmod 2^{32}$, then s_l is set to SEQ and ROC is set to v .

After a re-keying occurs (changing to a new master key), the rollover counter always maintains its sequence of values, i.e., it MUST NOT be reset to zero.

As the rollover counter is 32 bits long and the sequence number is 16 bits long, the maximum number of packets belonging to a given SRTP stream that can be secured with the same key is 2^{48} using the predefined transforms. After that number of SRTP packets have been sent with a given (master or session) key, the sender MUST NOT send any more packets with that key. (There exists a similar limit for SRTCP, which in practice may be more restrictive, see Section 9.2.) This limitation enforces a security benefit by providing an upper bound on the amount of traffic that can pass before cryptographic keys are changed. Re-keying (see Section 8.1) MUST be triggered, before this amount of traffic, and MAY be triggered earlier, e.g., for increased security and access control to media. Recurring key derivation by means of a non-zero key_derivation_rate (see Section 4.3), also gives stronger security but does not change the above absolute maximum value.

On the receiver side, there is a caveat to updating s_l and ROC: if message authentication is not present, neither the initialization of s_l , nor the ROC update can be made completely robust. The receiver's "implicit index" approach works for the pre-defined transforms as long as the reorder and loss of the packets are not too great and bit-errors do not occur in unfortunate ways. In particular, 2^{15} packets would need to be lost, or a packet would need to be 2^{15} packets out of sequence before synchronization is lost. Such drastic loss or reorder is likely to disrupt the RTP application itself.

The algorithm for the index estimate and ROC update is a matter of implementation, and should take into consideration the environment (e.g., packet loss rate) and the cases when synchronization is likely to be lost, e.g., when the initial sequence number (randomly chosen by RTP) is not known in advance (not sent in the key management protocol) but may be near to wrap modulo 2^{16} .

A more elaborate and more robust scheme than the one given above is the handling of RTP's own "rollover counter", see Appendix A.1 of [RFC3550].

Replay Protection

Secure replay protection is only possible when integrity protection is present. It is RECOMMENDED to use replay protection, both for RTP and RTCP, as integrity protection alone cannot assure security against replay attacks.

A packet is "replayed" when it is stored by an adversary, and then re-injected into the network. When message authentication is provided, SRTP protects against such attacks through a Replay List. Each SRTP receiver maintains a Replay List, which conceptually contains the indices of all of the packets which have been received and authenticated. In practice, the list can use a "sliding window" approach, so that a fixed amount of storage suffices for replay protection. Packet indices which lag behind the packet index in the context by more than SRTP-WINDOW-SIZE can be assumed to have been received, where SRTP-WINDOW-SIZE is a receiver-side, implementation dependent parameter and MUST be at least 64, but which MAY be set to a higher value.

The receiver checks the index of an incoming packet against the replay list and the window. Only packets with index ahead of the window, or, inside the window but not already received, SHALL be accepted.

After the packet has been authenticated (if necessary the window is first moved ahead), the replay list SHALL be updated with the new index. The Replay List can be efficiently implemented by using a bitmap to represent which packets have been received, as described in the Security Architecture for IP [RFC2401].

7.116.2.2 Generic AEAD Processing

Types of Input Data

- Associated Data: This is data that is to be authenticated but not encrypted
- Plaintext: Data that is to be both encrypted and authenticated
- Raw Data: Data that is to be neither encrypted nor authenticated

Which portions of SRTP/SRTCP packets that are to be treated as associated data, which are to be treated as plaintext, and which are to be treated as raw data

AEAD Invocation Inputs and Outputs

Encrypt Mode

Inputs:

- Encryption_key : Octet string, either 16 or 32 octets long
- Initialization_Vector : Octet string, 12 octets long
- Associated_Data : Octet string of variable length
- Plaintext : Octet string of variable length

Outputs:

- Ciphertext : Octet string, length = length(Plaintext)+tag_length

(*): In AEAD the authentication tag is embedded in the cipher text. When GCM is being used the ciphertext consists of the encrypted plain text followed by the authentication tag.

Decrypt Mode

Inputs:

- Encryption_key : Octet string, either 16 or 32 octets long
- Initialization_Vector : Octet string, 12 octets long
- Associated_Data : Octet string of variable length
- Ciphertext : Octet string of variable length

Outputs:

- Plaintext : Octet string, length = length(Ciphertext)-tag_length
- Validity_Flag : Boolean, TRUE if valid, FALSE otherwise

Handling of AEAD Authentication

AEAD requires that all incoming packets MUST pass AEAD authentication before any other action takes place. Plaintext and associated data MUST NOT be released until the AEAD authentication tag has been validated. Further the ciphertext MUST NOT be decrypted until the AEAD tag has been validated.

Should the AEAD tag prove to be invalid, the packet in question is to be discarded and a Validation Error flag raised. Local policy determines how this flag is to be handled and is outside the scope of this document.

7.116.2.3 Counter Mode Encryption

Each outbound packet uses a 12-octet IV and an encryption key to form two outputs, a 16-octet first_key_block which is used in forming the authentication tag and a key stream of octets, formed in blocks of 16-octets each. The first 16-octet block of key is saved for use in forming the authentication tag, and the remainder of the key stream is XORed to the plaintext to form cipher. This key stream is formed one block at a time by inputting the concatenation of a 12-octet IV (see sections 8.1 and 9.1) with a 4-octet block to AES. The pseudo-code below illustrates this process:

```

*   def GCM_keystream( Plaintext_len, IV, Encryption_key ):
*       assert Plaintext_len <= (2**36) - 32 ## measured in octets
*       key_stream = ""
*       block_counter = 1
*       first_key_block = AES_ENC( data=IV||block_counter,
*                                   key=Encryption_key           )
*       while len(key_stream) < Plaintext_len:
*           block_counter = block_counter + 1
*           key_block = AES_ENC( data=IV||block_counter,
*                                 key=Encryption_key         )
*           key_stream = key_stream || key_block
*           key_stream = truncate( key_stream, Plaintext_len )
*       return (first_key_block, key_stream)
*

```

In theory this keystream generation process allows for the encryption of up to $2^{36} - 32$ octets per invocation (i.e. per packet), far longer than is actually required.

With any counter mode, if the same (IV, Encryption_key) pair is used twice, precisely the same keystream is formed. As explained in section 9.1 of RFC 3711, this is a cryptographic disaster. For GCM the consequences are even worse since such a reuse compromises GCM's integrity mechanism not only for the current packet stream but for all future uses of the current encryption_key.

Unneeded SRTP/SRTCP Fields

AEAD counter mode encryption removes the need for certain existing SRTP/SRTCP mechanisms.

SRTP/SRTCP Authentication Field

The AEAD message authentication mechanism MUST be the primary message authentication mechanism for AEAD SRTP/SRTCP. Additional SRTP/SRTCP authentication mechanisms SHOULD NOT be used with any AEAD algorithm and the optional SRTP/SRTCP Authentication Tags are NOT RECOMMENDED and SHOULD NOT be present. Note that this contradicts section 3.4 of [RFC3711] which makes the use of the SRTCP Authentication field mandatory, but the presence of the AEAD authentication renders the older authentication methods redundant.

Rationale - Some applications use the SRTP/SRTCP Authentication Tag as a means of conveying additional information, notably [RFC4771]. This document retains the Authentication Tag field primarily to preserve compatibility with these applications.

RTP Padding

AES-GCM does not require that the data be padded out to a specific block size, reducing the need to use the padding mechanism provided by RTP. It is RECOMMENDED that the RTP padding mechanism not be used unless it is necessary to disguise the length of the underlying plaintext.

7.116.2.4 AES-GCM processing for SRTP

SRTP IV formation for AES-GCM

```

*   0   0   0   0   0   0   0   0   0   0   1   1
*   0   1   2   3   4   5   6   7   8   9   0   1
*   +-+-----+-----+-----+-----+-----+
*   |00|00|      SSRC    |      ROC    | SEQ |---+
*   +-+-----+-----+-----+-----+-----+-----+ |
*   |          |
*   +-+-----+-----+-----+-----+-----+-----+ |
*   |          Encryption Salt        | -> (+)
*   +-+-----+-----+-----+-----+-----+-----+ |
*   |          |
*   +-+-----+-----+-----+-----+-----+-----+ |
*   |          Initialization Vector | <-+
*   +-+-----+-----+-----+-----+-----+-----+
*
*   Figure 1: AES-GCM SRTP Initialization
*               Vector Formation
*

```

Vector formation - The 12 octet initialization vector used by AES-GCM SRTP is formed by first concatenating 2 octets of zeroes, the 4-octet SSRC, the 4-octet Rollover Counter (ROC) and the 2-octet sequence number SEQ. The resulting 12-octet value is then XORed to the 12-octet salt to form the 12-octet IV.

Summary of AES-GCM in SRTP/SRTCP

For convenience, much of the information about the use of AES-GCM family of algorithms in SRTP is collected in the tables contained in this section. The AES-GCM family of AEAD algorithms is built around the AES block cipher algorithm. AES-GCM uses AES counter mode for encryption and Galois Message Authentication Code (GMAC) for authentication. A detailed description of the AES-GCM family can be found in [RFC5116]. The following members of the AES-GCM family may be used with SRTP/SRTCP:

Name	Key Size	AEAD Tag Size	Reference
<hr/>			
AEAD_AES_128_GCM_8	16 octets	8 octets	[RFC5282]
AEAD_AES_128_GCM	16 octets	16 octets	[RFC5116]
AEAD_AES_256_GCM	32 octets	16 octets	[RFC5116]

Table 1: AES-GCM algorithms [for SRTP/SRTCP](#)

Any implementation of AES-GCM SRTP MUST support both AEAD_AES_128_GCM and AEAD_AES_256_GCM (the versions with 16 octet AEAD authentication tags), and it MAY support AEAD_AES_128_GCM_8. Below we summarize parameters associated with these three GCM algorithms:

Parameter	Value
Master key length	128 bits
Master salt length	96 bits
Key Derivation Function	AES_CM_PRF [RFC3711]
Maximum key lifetime (SRTP)	2^{48} packets
Maximum key lifetime (SRTCP)	2^{31} packets
Cipher (for SRTP and SRTCP)	AEAD_AES_128_GCM_8
AEAD authentication tag length	64 bits

Table 2: The AEAD_AES_128_GCM_8 Crypto Suite

Parameter	Value
Master key length	128 bits
Master salt length	96 bits
Key Derivation Function	AES_CM_PRF [RFC3711]
Maximum key lifetime (SRTP)	2^{48} packets
Maximum key lifetime (SRTCP)	2^{31} packets
Cipher (for SRTP and SRTCP)	AEAD_AES_128_GCM
AEAD authentication tag length	128 bits

Table 3: The AEAD_AES_128_GCM Crypto Suite

Parameter	Value
Master key length	256 bits
Master salt length	96 bits
Key Derivation Function	AES_256_CM_PRF [RFC6188]
Maximum key lifetime (SRTP)	2^{48} packets
Maximum key lifetime (SRTCP)	2^{31} packets
Cipher (for SRTP and SRTCP)	AEAD_AES_256_GCM
AEAD authentication tag length	128 bits

Table 4: The AEAD_AES_256_GCM Crypto Suite

7.116.2.5 AES-CCM for SRTP/SRTCP

AES-CCM is another family of AEAD algorithms built around the AES block cipher algorithm. AES-CCM uses AES counter mode for encryption and AES Cipher Block Chaining Message Authentication Code (CBC-MAC) for authentication. A detailed description of the AES-CCM family can be found in [RFC5116]. Four of the six CCM algorithms used in this document are defined in previous RFCs, while two, AEAD_AES_128_CCM_12 and AEAD_AES_256_CCM_12, are defined in section 7 of this document.

Any implementation of AES-CCM SRTP/SRTCP MUST support both AEAD_AES_128_CCM and AEAD_AES_256_CCM (the versions with 16 octet AEAD authentication tags), and MAY support the other four variants.

Name	Key Size	AEAD Tag Size	Reference
<hr/>			
AEAD_AES_128_CCM	128 bits	16 octets	[RFC5116]
AEAD_AES_256_CCM	256 bits	16 octets	[RFC5116]

```

*   AEAD_AES_128_CCM_12 128 bits    12 octets      see section 7
*   AEAD_AES_256_CCM_12 256 bits    12 octets      see section 7
*   AEAD_AES_128_CCM_8 128 bits     8 octets       [RFC6655]
*   AEAD_AES_256_CCM_8 256 bits     8 octets       [RFC6655]
*
*           Table 6: AES-CCM algorithms for SRTP/SRTCP
*
```

In addition to the flag octet used in counter mode encryption, AES-CCM authentications also uses a flag octet that conveys information about the length of the authentication tag, length of the block counter, and presence of additional authenticated data (see section 2.2 of [RFC3610]). For AES-CCM in SRTP/SRTCP, the flag octet has the hex value 5A if an 8-octet AEAD authentication tag is used, 6A if a 12-octet AEAD authentication tag is used, and 7A if a 16-octet AEAD authentication tag is used. The flag octet is one of the inputs to AES during the counter mode encryption of the plaintext.

```

* +-----+-----+
* | Parameter          | Value
* +-----+-----+
* | Master key length | 128 bits
* | Master salt length| 96 bits
* | Key Derivation Function | AES_CM_PRF [RFC3711]
* | Maximum key lifetime (SRTP) | 2^48 packets
* | Maximum key lifetime (SRTCP) | 2^31 packets
* | Cipher (for SRTP and SRTCP) | AEAD_AES_128_CCM_8
* | AEAD authentication tag length | 64 bits
* +-----+-----+
*
```

Table 7: The [AEAD_AES_128_CCM_8](#) Crypto Suite

```

* +-----+-----+
* | Parameter          | Value
* +-----+-----+
* | Master key length | 128 bits
* | Master salt length| 96 bits
* | Key Derivation Function | AES_CM_PRF [RFC3711]
* | Maximum key lifetime (SRTP) | 2^48 packets
* | Maximum key lifetime (SRTCP) | 2^31 packets
* | Cipher (for SRTP and SRTCP) | AEAD_AES_128_CCM_12
* | AEAD authentication tag length | 96 bits
* +-----+-----+
*
```

Table 8: The [AEAD_AES_128_CCM_12](#) Crypto Suite

```

* +-----+-----+
* | Parameter          | Value
* +-----+-----+
* | Master key length | 128 bits
* | Master salt length| 96 bits
* | Key Derivation Function | AES_CM_PRF [RFC3711]
* | Maximum key lifetime (SRTP) | 2^48 packets
* | Maximum key lifetime (SRTCP) | 2^31 packets
* | Cipher (for SRTP and SRTCP) | AEAD_AES_128_CCM
* | AEAD authentication tag length | 128 bits
* +-----+-----+
*
```

Table 9: The [AEAD_AES_128_CCM](#) Crypto Suite

```

* +-----+-----+
* | Parameter          | Value
* +-----+-----+
* | Master key length | 256 bits
* | Master salt length| 96 bits
* | Key Derivation Function | AES_256_CM_PRF [RFC6188]
* | Maximum key lifetime (SRTP) | 2^48 packets
* | Maximum key lifetime (SRTCP) | 2^31 packets
* | Cipher (for SRTP and SRTCP) | AEAD_AES_256_CCM_8
* | AEAD authentication tag length | 64 bits
* +-----+-----+
*
```

Table 10: The [AEAD_AES_256_CCM_8](#) Crypto Suite

```

* +-----+-----+
* | Parameter          | Value
* +-----+-----+
* | Master key length | 256 bits
* | Master salt length| 96 bits
* | Key Derivation Function | AES_256_CM_PRF [RFC6188]
* | Maximum key lifetime (SRTP) | 2^48 packets
* | Maximum key lifetime (SRTCP) | 2^31 packets
* +-----+-----+
*
```

```

* | Cipher (for SRTP and SRTCP) | AEAD_AES_256_CCM_12 |
* | AEAD authentication tag length | 96 bits |
* +-----+-----+
*
*      Table 11: The AEAD_AES_256_CCM_12 Crypto Suite
*
*
* +-----+-----+
* | Parameter | Value |
* +-----+-----+
* | Master key length | 256 bits |
* | Master salt length | 96 bits |
* | Key Derivation Function | AES_256_CM_PRF [RFC6188] |
* | Maximum key lifetime (SRTP) | 2^48 packets |
* | Maximum key lifetime (SRTCP) | 2^31 packets |
* | Cipher (for SRTP and SRTCP) | AEAD_AES_256_CCM |
* | AEAD authentication tag length | 128 bits |
* +-----+-----+
*
*      Table 12: The AEAD_AES_256_CCM Crypto Suite
*

```

7.116.2.6 Use of the ARIA Block cipher with SRTP

ARIA is a general-purpose block cipher algorithm developed by Korean cryptographers in 2003. It is an iterated block cipher with 128-, 192-, and 256-bit keys and encrypts 128-bit blocks in 12, 14, and 16 rounds, depending on the key size. It is secure and suitable for most software and hardware implementations on 32-bit and 8-bit processors. It was established as a Korean standard block cipher algorithm in 2004 [ARIAKS] and has been widely used in Korea, especially for government-to-public services. It was included in PKCS #11 in 2007 [ARIAPKCS]. The algorithm specification and object identifiers are described in [RFC5794].

ARIA-CTR

Section 4.1.1 of [RFC3711] defines AES-128 counter mode encryption, which it refers to as "AES_CM". Section 2 of [RFC6188] defines "AES_192_CM" and "AES_256_CM" in SRTP. ARIA counter modes are defined in the same manner except that each invocation of AES is replaced by that of ARIA [RFC5794], and are denoted by ARIA_128_CTR, ARIA_192_CTR and ARIA_256_CTR respectively, according to the key lengths. The plaintext inputs to the block cipher are formed as in AES-CTR(AES_CM, AES_192_CM, AES_256_CM) and the block cipher outputs are processed as in AES-CTR. Note that, ARIA-CTR MUST be used only in conjunction with an authentication transform.

Section 3.2 of [RFC6904] defines AES-CTR for SRTP header extension keystream generation. When ARIA-CTR is used, the header extension keystream SHALL be generated in the same manner except that each invocation of AES is replaced by that of ARIA [RFC5794].

ARIA-GCM

GCM (Galois Counter Mode) [GCM][RFC5116] is an AEAD (Authenticated Encryption with Associated Data) block cipher mode. A detailed description of ARIA-GCM is defined similarly as AES-GCM found in [RFC5116][RFC5282].

The document [I-D.ietf-avtcore-srtp-aes-gcm] describes the use of AES-GCM with SRTP [RFC3711][RFC6904]. The use of ARIA-GCM with SRTP is defined the same as that of AES-GCM except that each invocation of AES is replaced by ARIA [RFC5794]. When [RFC6904] is in use, a separate keystream to encrypt selected RTP header extension elements MUST be generated in the same manner defined in [I-D.ietf-avtcore-srtp-aes-gcm] except that AES-CTR is replaced by ARIA-CTR.

Protection Profiles

This section defines SRTP Protection Profiles that use the ARIA transforms and key derivation functions defined in this document. The following list indicates the SRTP transform parameters for each protection profile. Those are described for use with DTLS-SRTP [RFC5764].

The parameters cipher_key_length, cipher_salt_length, auth_key_length, and auth_tag_length express the number of bits in the values to which they refer. The maximum_lifetime parameter indicates the maximum number of packets that can be protected with each single set of keys when the parameter profile is in use. All of these parameters apply to both RTP and RTCP, unless the RTCP parameters are separately specified.

SRTP_ARIA_128_CTR_HMAC_SHA1_80

- cipher: ARIA_128_CTR

- cipher_key_length: 128 bits
- cipher_salt_length: 112 bits
- key derivation function: ARIA_128_CTR_PRF
- auth_function: HMAC-SHA1
- auth_key_length: 160 bits
- auth_tag_length: 80 bits
- maximum_lifetime: at most 2^{31} SRTCP packets and at most 2^{48} SRTP packets

SRTP_ARIA_128_CTR_HMAC_SHA1_32

- cipher: ARIA_128_CTR
- cipher_key_length: 128 bits
- cipher_salt_length: 112 bits
- key derivation function: ARIA_128_CTR_PRF
- auth_function: HMAC-SHA1
- auth_key_length: 160 bits
- SRTP auth_tag_length: 32 bits
- SRTCP auth_tag_length: 80 bits
- maximum_lifetime: at most 2^{31} SRTCP packets and at most 2^{48} SRTP packets

SRTP_ARIA_192_CTR_HMAC_SHA1_80

- cipher: ARIA_192_CTR
- cipher_key_length: 192 bits
- cipher_salt_length: 112 bits
- key derivation function: ARIA_192_CTR_PRF
- auth_function: HMAC-SHA1
- auth_key_length: 160 bits
- auth_tag_length: 80 bits
- maximum_lifetime: at most 2^{31} SRTCP packets and at most 2^{48} SRTP packets

SRTP_ARIA_192_CTR_HMAC_SHA1_32

- cipher: ARIA_192_CTR
- cipher_key_length: 192 bits
- cipher_salt_length: 112 bits
- key derivation function: ARIA_192_CTR_PRF
- auth_function: HMAC-SHA1
- auth_key_length: 160 bits
- SRTP auth_tag_length: 32 bits

- SRTCP auth_tag_length: 80 bits
- maximum_lifetime: at most 2^{31} SRTCP packets and at most 2^{48} SRTP packets

SRTP_ARIA_256_CTR_HMAC_SHA1_80

- cipher: ARIA_256_CTR
- cipher_key_length: 256 bits
- cipher_salt_length: 112 bits
- key derivation function: ARIA_256_CTR_PRF
- auth_function: HMAC-SHA1
- auth_key_length: 160 bits
- auth_tag_length: 80 bits
- maximum_lifetime: at most 2^{31} SRTCP packets and at most 2^{48} SRTP packets

SRTP_ARIA_256_CTR_HMAC_SHA1_32

- cipher: ARIA_256_CTR
- cipher_key_length: 256 bits
- cipher_salt_length: 112 bits
- key derivation function: ARIA_256_CTR_PRF
- auth_function: HMAC-SHA1
- auth_key_length: 160 bits
- SRTP auth_tag_length: 32 bits
- SRTCP auth_tag_length: 80 bits
- maximum_lifetime: at most 2^{31} SRTCP packets and at most 2^{48} SRTP packets

SRTP_AEAD_ARIA_128_GCM

- cipher: ARIA_128_GCM
- cipher_key_length: 128 bits
- cipher_salt_length: 96 bits
- aead_auth_tag_length: 128 bits
- auth_function: NULL
- auth_key_length: N/A
- auth_tag_length: N/A
- key derivation function: ARIA_128_CTR_PRF
- maximum_lifetime: at most 2^{31} SRTCP packets and at most 2^{48} SRTP packets

SRTP_AEAD_ARIA_256_GCM

- cipher: ARIA_256_GCM
- cipher_key_length: 256 bits

- cipher_salt_length: 96 bits
- aead_auth_tag_length: 128 bits
- auth_function: NULL
- auth_key_length: N/A
- auth_tag_length: N/A
- key derivation function: ARIA_256_CTR_PRF
- maximum_lifetime: at most 2^{31} SRTCP packets and at most 2^{48} SRTP packets

The ARIA-CTR protection profiles use the same authentication transform that is mandatory to implement in SRTP, HMAC-SHA1 with a 160-bit key.

Note that SRTP Protection Profiles which use AEAD algorithms do not specify an auth_function, auth_key_length, or auth_tag_length, since they do not use a separate auth_function, auth_key, or auth_tag. The term aead_auth_tag_length is used to emphasize that this refers to the authentication tag provided by the AEAD algorithm and that this tag is not located in the authentication tag field provided by SRTP/SRTCP.

[1] <http://blog.csdn.net/fanbird2008/article/details/18623141>

For API Documentation:

See Also

ProtocolPP::jprotocol

For Additional Documentation:

See Also

jprotocol

Protocol++ written by : John Peter Greninger © John Peter Greninger 2015-2019 All Rights Reserved

All copyrights and trademarks are the property of their respective owners

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer
- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution
- Any and all modifications must be returned to John Peter Greninger at GitHub.com <https://github.com/jpgreninger/protocolpp> for evaluation. Inclusion of modifications in the source code shall be determined solely by John Peter Greninger. Failure to provide modifications shall render this license NULL and VOID and revoke any rights to use of Protocol++
- Commercial use requires a fee-based license obtainable at www.protocolpp.com
- Academic use requires written and notarized permission from John Peter and Sheila Rocha Greninger
- US Copyrights at <https://www.copyright.gov/>
 - TXu002059872 (Version 1.0.0)
 - TXu002066632 (Version 1.2.7)
 - TXu002082674 (Version 1.4.0)

- TXu002097880 (Version 2.0.0)

The names of its contributors may not be used to endorse or promote products derived from this software without specific prior written permission

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE

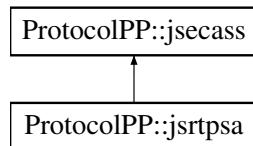
The documentation for this class was generated from the following file:

- [include/jsrtpsa.h](#)

7.117 ProtocolPP::jsrtpsa Class Reference

```
#include <jsrtpsa.h>
```

Inheritance diagram for ProtocolPP::jsrtpsa:



Public Member Functions

- [jsrtpsa \(\)](#)
- [jsrtpsa \(direction_t dir, protocol_t mode, srpcipher_t cipher, unsigned int icvlen, unsigned int blksize, uint16_t seqnum, uint32_t roc, uint32_t ssrc, uint8_t ver, uint8_t type, bool pad, bool ext, std::shared_ptr< jarray< uint8_t >> exthdr, bool marker, bool mki, unsigned int ckeylen, std::shared_ptr< jarray< uint8_t >> cipherkey, unsigned int akeylen, std::shared_ptr< jarray< uint8_t >> authkey, unsigned int saltlen, std::shared_ptr< jarray< uint8_t >> salt, unsigned int mkilen, std::shared_ptr< jarray< uint8_t >> mkidata, unsigned int cc, std::shared_ptr< jarray< uint32_t >> csrc, unsigned int arlen, jarray< uint8_t > arwin\)](#)
- [jsrtpsa \(jsrtpsa &rhs\)](#)
- [jsrtpsa \(std::shared_ptr< jsrtpsa > &rhs\)](#)
- [virtual ~jsrtpsa \(\)](#)
 - Standard deconstructor.*
- template<typename T>
 [void set_field \(field_t field, T fieldval\)](#)
- template<typename T>
 [T get_field \(field_t field\)](#)
- [void to_xml \(tinyxml2::XMLPrinter &myxml, direction_t direction\)](#)

7.117.1 Constructor & Destructor Documentation

7.117.1.1 ProtocolPP::jsrtpsa::jsrtpsa ()

Standard constructor with defaults

~~7.117.1.2 ProtocolPP::jsrtpsa::jsrtpsa (direction_t dir, protocol_t mode, srpcipher_t cipher, unsigned int icvlen, unsigned int blksize, uint16_t seqnum, uint32_t roc, uint32_t ssrc, uint8_t ver, uint8_t type, bool pad, bool ext, std::shared_ptr< jarray< uint8_t >> exthdr, bool marker, bool mki, unsigned int ckeylen, std::shared_ptr< jarray< uint8_t >> cipherkey, unsigned int akeylen, std::shared_ptr< jarray< uint8_t >> authkey, unsigned int saltlen, std::shared_ptr< jarray< uint8_t >> salt, unsigned int mkilen, std::shared_ptr< jarray< uint8_t >> mkidata, unsigned int cc, std::shared_ptr< jarray< uint32_t >> csrc, unsigned int arlen, jarray< uint8_t > arwin)~~

Parameters

<i>dir</i>	- Direction of processing (ENCAP or DECAP)
<i>mode</i>	- Mode of operation either SRTP or SCRTP
<i>cipher</i>	- indicates the cipher used for SRTP encryption
<i>icvlen</i>	- ICV length
<i>blksize</i>	- Size of the encryption data size
<i>seqnum</i>	- Sequence number
<i>roc</i>	- Rollover counter (ROC)
<i>ssrc</i>	- Synchronization source identifier (SSRC)
<i>ver</i>	- Version of SRTP (currently 2)
<i>type</i>	- Type of the payload data
<i>pad</i>	- Indicates padding is present in the payload
<i>ext</i>	- If asserted, a single extension header follows the SRTP header
<i>exthdr</i>	- Extension header data if present
<i>marker</i>	- Marker for debug
<i>mki</i>	- MKI flag
<i>ckeylen</i>	- Cipher key length
<i>cipherkey</i>	- the session encryption key
<i>akeylen</i>	- Authentication key length
<i>authkey</i>	- the session authentication key for HMAC-SHA1
<i>saltlen</i>	- Salt length
<i>salt</i>	- the session salting key
<i>mkilen</i>	- MKI length
<i>mkidata</i>	- MKI data
<i>cc</i>	- Number of CSRC identifiers
<i>csrc</i>	- Contributing source identifier (CSRC)
<i>arlen</i>	- Number of packets to track in replay window
<i>arwin</i>	- Anti-replay window

7.117.1.3 ProtocolPP::jsrtpsa::jsrtpsa (*jsrtpsa & rhs*)

constructor for SRTP Security Association

Parameters

<i>rhs</i>	- security association for SRTP
------------	---------------------------------

7.117.1.4 ProtocolPP::jsrtpsa::jsrtpsa (std::shared_ptr<*jsrtpsa* > & *rhs*)

constructor for SRTP Security Association

Parameters

<i>rhs</i>	- security association for SRTP
------------	---------------------------------

7.117.1.5 virtual ProtocolPP::jsrtpsa::~jsrtpsa() [inline], [virtual]

Standard deconstructor.

7.117.2 Member Function Documentation

	field name	Example
Returns Parameters	DIRECTION MODE	direction_t mydir = get_field<-ProtocolPP::direction_t>(ProtocolPP::field_t::DIRECTION) protocol_t mymode = get_field<ProtocolPP::protocol_t>(ProtocolPP::field_t::MODE)
Returns	CIPHER	srtpcipher_t mycipher = get_field<ProtocolPP::srtpcipher_t>(ProtocolPP::field_t::CIPHER)
7.117.2.2	ICVLEN	uint32_t myicvlen = get_field<uint32_t>(ProtocolPP::field_t::ICVLEN)
Allows the user to update the field of the SRTP security association.	BLKSIZE	uint32_t myblksize = get_field<uint32_t>(ProtocolPP::field_t::BLKSIZE)
Parameters	ROC	uint32_t myroc = get_field<uint32_t>(ProtocolPP::field_t::ROC)
	SSR	fieldval - value to update the SRTP security association
		get_field<uint32_t>(ProtocolPP::field_t::SSRC)
7.117.2.3	CKEYLEN	uint32_t myckeylen = get_field<uint32_t>(ProtocolPP::field_t::CKEYLEN)
Prints the protocol and security objects to XML.	AKEYLEN	uint32_t myakeylen = get_field<uint32_t>(ProtocolPP::field_t::AKEYLEN)
Parameters	myxml	- XMLPrinter object
	SALTLEN	uint32_t mysaltlen = get_field<uint32_t>(ProtocolPP::field_t::SALTLEN)
Implements ProtocolPP::jsecass .	direction	- randomization
MKILEN		uint32_t mymkilen = get_field<uint32_t>(ProtocolPP::field_t::MKILEN)
The documentation for this class was generated from the following file:	• include/jsrtpsa.h	get_field<uint32_t>(ProtocolPP::field_t::CC)
ARLEN		uint32_t myarlen = get_field<uint32_t>(ProtocolPP::field_t::ARLEN)
7.118 jsrtpsa Class Reference		
#include "include/jsrtpsa.h"	SEQNUM	uint16_t myseqnum = get_field<uint16_t>(ProtocolPP::field_t::SEQNUM)
7.118.1	VERSION	uint8_t myver = get_field<uint8_t>(ProtocolPP::field_t::VERSION)
7.118.2	Secure Real-Time Protocol Security Association (SRTP)	uint8_t mytype = get_field<uint8_t>(ProtocolPP::field_t::TYPE)
RTP is an extension to the RTP Transport Protocol [RFC3550]. It defines SRTP as a profile of RTP. This profile is an extension to the RTP Audio/Video Profile [RFC3551].	PAD	bool mypad = get_field<bool>(ProtocolPP::field_t::PAD)
SRTP Framework	EXTENSION	ProtocolPP::ProfileEXTENSION aspects of that profile apply, with the additional MARKER RTP security features. Conceptually, one can consider SRTP to be a "bump in the stack" implementation which resides between the RTP application and the transport layer. SRTP intercepts RTP packets and then forwards an equivalent SRTP packet on the sending side; MARKER intercepts SRTP packets and passes an equivalent RTP packet up the stack on the receiving side.
Secure RTP (see IETF specifications RFC3711, RFC7741)	ARWIN	bool mymki = get_field<bool>(ProtocolPP::field_t::MKI). Protects the RTP fields to keep track of membership, provide feedback to RTP senders, or maintain packet sequence counters. SRTP is described in Section 3.4.
The format of an SRTP packet is illustrated in Figure 7.11.	EXTHDR	std::shared_ptr<ProtocolPP::jarray<uint8_t>> myexthdr = std::shared_ptr<ProtocolPP::jarray<uint8_t>> myexthdr =

Parameters

<i>name</i>	- string to populate the array with
<i>dir</i>	- direction of processing
<i>type</i>	- type of security association
<i>postprocess</i>	- if using a random IV, postprocess the descriptor to decap the payload and verify
<i>security</i>	- Security association for this stream
<i>engine</i>	- Encryption engine for this stream

7.120.1.2 `ProtocolPP::jstream::jstream (std::string name, direction_t dir, protocol_t type, bool postprocess, std::shared_ptr<jsecass> security) [inline]`

Constructor for jstream

Parameters

<i>name</i>	- string to populate the array with
<i>dir</i>	- direction of processing
<i>type</i>	- type of security association
<i>postprocess</i>	- if using a random IV, postprocess the descriptor to decap the payload and verify
<i>security</i>	- Security association for this stream

7.120.1.3 `ProtocolPP::jstream::jstream (std::string name, direction_t dir, protocol_t type, bool postprocess, std::shared_ptr<jprotocol> engine) [inline]`

Constructor for jstream

Parameters

<i>name</i>	- string to populate the array with
<i>dir</i>	- direction of processing
<i>type</i>	- type of security association
<i>postprocess</i>	- if using a random IV, postprocess the descriptor to decap the payload and verify
<i>engine</i>	- Encryption engine for this stream

7.120.1.4 `virtual ProtocolPP::jstream::~jstream () [inline], [virtual]`

Standard deconstructor.

7.120.2 Member Function Documentation

7.120.2.1 `void ProtocolPP::jstream::clean () [inline]`

Erase the encryption engine if not needed. Encryption engine is only needed to generate random packets. If running W.A.S.P, the encryption engine will be deleted after generating random packets to reduce the memory footprint of the stream object

7.120.2.2 `direction_t ProtocolPP::jstream::get_dir () [inline]`

Get the direction for this packet

Returns

- direction of processing

7.120.2.3 void ProtocolPP::jstream::get_engine(std::shared_ptr<jprotocol> & newengine) [inline]

Get the encryption engine for this stream

Returns

- pointer to the encryption engine

7.120.2.4 std::string ProtocolPP::jstream::get_name() [inline]

Get the name for this packet

Returns

- string containing name of the stream

7.120.2.5 bool ProtocolPP::jstream::get_post() [inline]

Get the postprocessing flag for this flow

Returns

- postprocessing flag

7.120.2.6 template<typename T> void ProtocolPP::jstream::get_security(std::shared_ptr<T> & security) [inline]

Get the security association for this stream

Returns

- pointer to the security association NOTE : User must cast the security association to its correct type

7.120.2.7 bool ProtocolPP::jstream::get_setup() [inline]

Get the setup flag

Returns

- true/false on whether stream is set up

7.120.2.8 unsigned int ProtocolPP::jstream::get_size() [inline]

Get the name for this packet

Returns

- string containing name of the stream

7.120.2.9 protocol_t ProtocolPP::jstream::get_type() [inline]

Get the type of this security association

Returns

- type of security association

7.120.2.10 void ProtocolPP::jstream::set_dir(direction_t *dir*) [inline]

Set the direction for this stream

Parameters

<i>dir</i>	- direction of processing
------------	---------------------------

7.120.2.11 void ProtocolPP::jstream::set_engine (std::shared_ptr<jprotocol> *engine*) [inline]

Set the encryption engine for this stream

Parameters

<i>engine</i>	- encryption engine
---------------	---------------------

7.120.2.12 void ProtocolPP::jstream::set_name (std::string *name*) [inline]

Set the name for this stream

Parameters

<i>name</i>	- name of the stream
-------------	----------------------

7.120.2.13 void ProtocolPP::jstream::set_post (bool *postprocess*) [inline]

Set the postprocessing flag for this stream

Parameters

<i>postprocess</i>	- flag to indicate postprocessing
--------------------	-----------------------------------

7.120.2.14 void ProtocolPP::jstream::set_security (std::shared_ptr<jsecass> & *security*) [inline]

Set the security association for this stream

Parameters

<i>security</i>	- pointer to security association
-----------------	-----------------------------------

7.120.2.15 void ProtocolPP::jstream::set_setup (bool *set*) [inline]

Set the setup flag to indicate the stream information has been setup and passed to the device such that packets can then be issued

Parameters

<i>set</i>	- true/false on whether the streams is set up
------------	---

7.120.2.16 void ProtocolPP::jstream::set_size (unsigned int *size*) [inline]

Set the size for the stream

Parameters

<code>size</code>	- size of the stream
-------------------	----------------------

7.120.2.17 void ProtocolPP::jstream::set_type (`protocol_t type`) [inline]

Set the type for this stream

Parameters

<code>type</code>	- type of the security association
-------------------	------------------------------------

The documentation for this class was generated from the following file:

- [include/jstream.h](#)

7.121 jtcp Class Reference

```
#include "include/jtcp.h"
```

7.121.1 Detailed Description

7.121.2 Transport Control Protocol (TCP)

See https://en.wikipedia.org/wiki/Transmission_Control_Protocol

The Transmission Control Protocol (TCP) is a core protocol of the Internet protocol suite. It originated in the initial network implementation in which it complemented the Internet Protocol (IP). Therefore, the entire suite is commonly referred to as TCP/IP. TCP provides reliable, ordered, and error-checked delivery of a stream of octets between applications running on hosts communicating over an IP network. Major Internet applications such as the World Wide Web, email, remote administration and file transfer rely on TCP. Applications that do not require reliable data stream service may use the User Datagram Protocol (UDP), which provides a connectionless datagram service that emphasizes reduced latency over reliability.

TCP segments are sent as internet datagrams. The Internet Protocol header carries several information fields, including the source and destination host addresses [2]. A TCP header follows the internet header, supplying information specific to the TCP protocol. This division allows for the existence of host level protocols other than TCP (See RFC 793)

TCP Header Format [1]

TCP Segment Header Format										
Bit #	0	7	8	15	16	23	24	31		
0	Source Port				Destination Port					
32	Sequence Number									
64	Acknowledgment Number									
96	Data Offset	Res	Flags		Window Size					
128	Header and Data Checksum				Urgent Pointer					
160...	Options									

Figure 7.106: Transport Control Protocol (TCP) Header Format

- Source Port: 16 bits

The source port number

- Destination Port: 16 bits

The destination port number

- Sequence Number: 32 bits

The sequence number of the first data octet in this segment (except when SYN is present). If SYN is present the sequence number is the initial sequence number (ISN) and the first data octet is ISN+1

- Acknowledgment Number: 32 bits

If the ACK control bit is set this field contains the value of the next sequence number the sender of the segment is expecting to receive. Once a connection is established this is always sent

- Data Offset: 4 bits

The number of 32 bit words in the TCP Header. This indicates where the data begins. The TCP header (even one including options) is an integral number of 32 bits long.

- Reserved: 6 bits

Reserved for future use. Must be zero

- Flags: 6 bits (from left to right):

- URG: Urgent Pointer field significant
- ACK: Acknowledgment field significant
- PSH: Push Function
- RST: Reset the connection
- SYN: Synchronize sequence numbers
- FIN: No more data from sender

- Window: 16 bits

The number of data octets beginning with the one indicated in the acknowledgment field which the sender of this segment is willing to accept

- Checksum: 16 bits

The checksum field is the 16 bit one's complement of the one's complement sum of all 16 bit words in the header and text. If a segment contains an odd number of header and text octets to be checksummed, the last octet is padded on the right with zeros to form a 16 bit word for checksum purposes. The pad is not transmitted as part of the segment. While computing the checksum, the checksum field itself is replaced with zeros

The checksum also covers a 96 bit pseudo header conceptually prefixed to the TCP header. This pseudo header contains the Source Address, the Destination Address, the Protocol, and TCP length. This gives the TCP protection against misrouted segments. This information is carried in the Internet Protocol and is transferred across the TCP/Network interface in the arguments or results of calls by the TCP on the IP

```
*
*      +-----+-----+-----+-----+
*      |           Source Address          |
*      +-----+-----+-----+-----+
*      |           Destination Address       |
*      +-----+-----+-----+-----+
*      | zero  | PTCL  |   TCP Length   |
*      +-----+-----+-----+-----+
*
```

The TCP Length is the TCP header length plus the data length in octets (this is not an explicitly transmitted quantity, but is computed), and it does not count the 12 octets of the pseudo header

- Urgent Pointer: 16 bits

This field communicates the current value of the urgent pointer as a positive offset from the sequence number in this segment. The urgent pointer points to the sequence number of the octet following the urgent data. This field is only be interpreted in segments with the URG control bit set

- Options (Variable 0–320 bits, divisible by 32)

The length of this field is determined by the data offset field. Options have up to three fields: Option-Kind (1 byte), Option-Length (1 byte), Option-Data (variable). The Option-Kind field indicates the type of option, and is the only field that is not optional. Depending on what kind of option we are dealing with, the next two fields may be set: the Option-Length field indicates the total length of the option, and the Option-Data field contains the value of the option, if applicable. For example, an Option-Kind byte of 0x01 indicates that this is a No-Op option used only for padding, and does not have an Option-Length or Option-Data byte following it. An Option-Kind byte of 0 is the End Of Options option, and is also only one byte. An Option-Kind byte of 0x02 indicates that this is the Maximum Segment Size option, and will be followed by a byte specifying the length of the MSS field (should be 0x04). Note that this length is the total length of the given options field, including Option-Kind and Option-Length bytes. So while the MSS value is typically expressed in two bytes, the length of the field will be 4 bytes (+2 bytes of kind and length). In short, an MSS option field with a value of 0x05B4 will show up as (0x02 0x04 0x05B4) in the TCP options section

Some options may only be sent when SYN is set; they are indicated below as [SYN]. Option-Kind and standard lengths given as (Option-Kind, Option-Length)

- 0 (8 bits) – End of options list
- 1 (8 bits) – No operation (NOP, Padding) This may be used to align option fields on 32-bit boundaries for better performance
- 2,4,SS (32 bits) – Maximum segment size (see maximum segment size) [SYN]
- 3,3,S (24 bits) – Window scale (see window scaling for details) [SYN][6]
- 4,2 (16 bits) – Selective Acknowledgement permitted. [SYN] (See selective acknowledgments for details)[7]
- 5,N,BBBB,EEEE,... (variable bits, N is either 10, 18, 26, or 34)- Selective ACKnowledgement (SACK)[8] These first two bytes are followed by a list of 1–4 blocks being selectively acknowledged, specified as 32-bit begin/end pointers
- 8,10,TTTT,EEEE (80 bits)- Timestamp and echo of previous timestamp (see TCP timestamps for details)[9]

(The remaining options are historical, obsolete, experimental, not yet standardized, or unassigned)

Options: variable

Options may occupy space at the end of the TCP header and are a multiple of 8 bits in length. All options are included in the checksum. An option may begin on any octet boundary. There are two cases for the format of an option:

- Case 1: A single octet of option-kind
- Case 2: An octet of option-kind, an octet of option-length, and the actual option-data octets.

The option-length counts the two octets of option-kind and option-length as well as the option-data octets

Note that the list of options may be shorter than the data offset field might imply. The content of the header beyond the End-of-Option option must be header padding (i.e., zero)

A TCP must implement all options.

Currently defined options include (kind indicated in octal):

```

*
*      Kind      Length      Meaning
*      ----      -----      -----
*      0          -          End of option list
*      1          -          No-Operation
*      2          4          Maximum Segment Size.
*
*
```

Specific Option Definitions

```

*
*      End of Option List
*
*      +-----+
*      |00000000|
*      +-----+
*      Kind=0
*
*
```

This option code indicates the end of the option list. This might not coincide with the end of the TCP header according to the Data Offset field. This is used at the end of all options not the end of each option, and need only be used if the end of the options would not otherwise coincide with the end of the TCP header

```

*
*      No-Operation
*
*      +-----+
*      |00000001|
*      +-----+
*      Kind=1
*
*
```

This option code may be used between options, for example, to align the beginning of a subsequent option on a word boundary. There is no guarantee that senders will use this option, so receivers must be prepared to process options even if they do not begin on a word boundary

```

*
*      Maximum Segment Size
*
*      +-----+-----+-----+
*      |00000010|00000100| max seg size   |
*      +-----+-----+-----+
*      Kind=2    Length=4
*
*
```

- Maximum Segment Size Option Data: 16 bits

If this option is present, then it communicates the maximum receive segment size at the TCP which sends this segment. This field must only be sent in the initial connection request (i.e., in segments with the SYN control bit set). If this option is not used, any segment size is allowed

Padding

The TCP header padding is used to ensure that the TCP header ends and data begins on a 32 bit boundary. The padding is composed of zeros[10]

Protocol operation

TCP protocol operations may be divided into three phases. Connections must be properly established in a multi-step handshake process (connection establishment) before entering the data transfer phase. After data transmission is completed, the connection termination closes established virtual circuits and releases all allocated resources

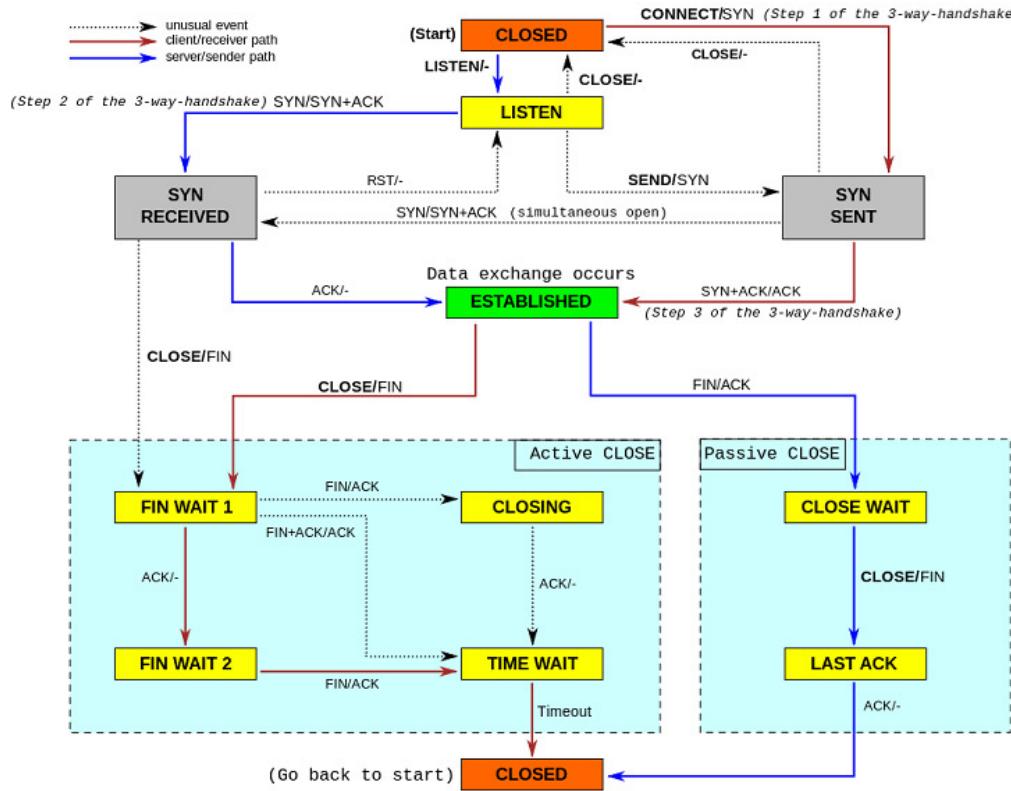


Figure 7.107: Transmission Control Protocol (TCP) State Transition Diagram [2]

A TCP connection is managed by an operating system through a programming interface that represents the local end-point for communications, the Internet socket. During the lifetime of a TCP connection the local end-point undergoes a series of state changes:[11]

LISTEN

- (server) represents waiting for a connection request from any remote TCP and port

SYN-SENT

- (client) represents waiting for a matching connection request after having sent a connection request

SYN-RECEIVED

- (server) represents waiting for a confirming connection request acknowledgment after having both received and sent a connection request

ESTABLISHED

- (both server and client) represents an open connection, data received can be delivered to the user. The normal state for the data transfer phase of the connection

FIN-WAIT-1

- (both server and client) represents waiting for a connection termination request from the remote TCP, or an acknowledgment of the connection termination request previously sent

FIN-WAIT-2

- (both server and client) represents waiting for a connection termination request from the remote TCP

CLOSE-WAIT

- (both server and client) represents waiting for a connection termination request from the local user

CLOSING

- (both server and client) represents waiting for a connection termination request acknowledgment from the remote TCP

LAST-ACK

- (both server and client) represents waiting for an acknowledgment of the connection termination request previously sent to the remote TCP (which includes an acknowledgment of its connection termination request)

TIME-WAIT

- (either server or client) represents waiting for enough time to pass to be sure the remote TCP received the acknowledgment of its connection termination request. [According to RFC 793 a connection can stay in TIME-WAIT for a maximum of four minutes known as two MSL (maximum segment lifetime).]

CLOSED

- (both server and client) represents no connection state at all

Connection establishment

To establish a connection, TCP uses a three-way handshake. Before a client attempts to connect with a server, the server must first bind to and listen at a port to open it up for connections: this is called a passive open. Once the passive open is established, a client may initiate an active open. To establish a connection, the three-way (or 3-step) handshake occurs:

- 1.SYN: The active open is performed by the client sending a SYN to the server. The client sets the segment's sequence number to a random value A
- 2.SYN-ACK: In response, the server replies with a SYN-ACK. The acknowledgement number is set to one more than the received sequence number i.e. A+1, and the sequence number that the server chooses for the packet is another random number, B
- 3.ACK: Finally, the client sends an ACK back to the server. The sequence number is set to the received acknowledgement value i.e. A+1, and the acknowledgement number is set to one more than the received sequence number i.e. B+1

At this point, both the client and server have received an acknowledgment of the connection. The steps 1, 2 establish the connection parameter (sequence number) for one direction and it is acknowledged. The steps 2, 3 establish the connection parameter (sequence number) for the other direction and it is acknowledged. With these, a full-duplex communication is established

Connection termination

The connection termination phase uses a four-way handshake, with each side of the connection terminating independently. When an endpoint wishes to stop its half of the connection, it transmits a FIN packet, which the other end acknowledges with an ACK. Therefore, a typical tear-down requires a pair of FIN and ACK segments from each TCP endpoint. After the side that sent the first FIN has responded with the final ACK, it waits for a timeout before finally closing the connection, during which time the local port is unavailable for new connections; this prevents confusion due to delayed packets being delivered during subsequent connections

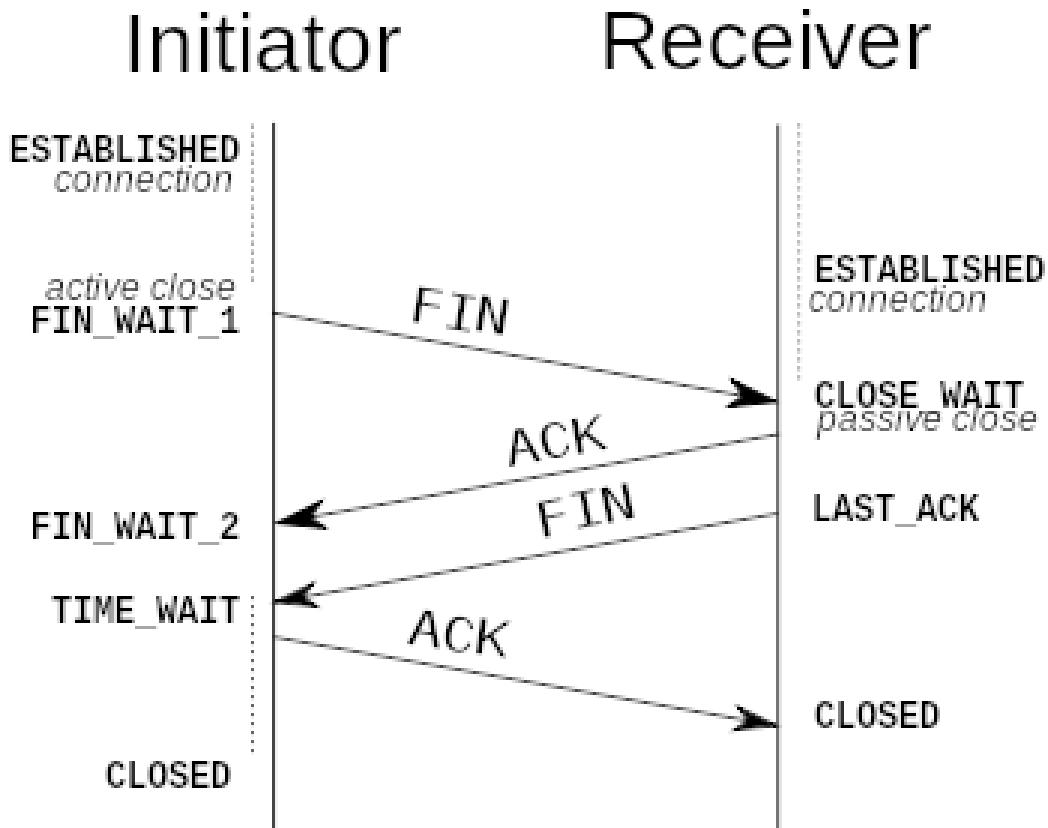


Figure 7.108: Transmission Control Protocol (TCP) Closing Procedure [3]

A connection can be "half-open", in which case one side has terminated its end, but the other has not. The side that has terminated can no longer send any data into the connection, but the other side can. The terminating side should continue reading the data until the other side terminates as well

It is also possible to terminate the connection by a 3-way handshake, when host A sends a FIN and host B replies with a FIN & ACK (merely combines 2 steps into one) and host A replies with an ACK.[12]

Some host TCP stacks may implement a half-duplex close sequence, as Linux or HP-UX do. If such a host actively closes a connection but still has not read all the incoming data the stack already received from the link, this host sends a RST instead of a FIN (Section 4.2.2.13 in RFC 1122). This allows a TCP application to be sure the remote application has read all the data the former sent—waiting the FIN from the remote side, when it actively closes the connection. But the remote TCP stack cannot distinguish between a Connection Aborting RST and Data Loss RST. Both cause the remote stack to lose all the data received

TCP checksum for IPv4

When TCP runs over IPv4, the method used to compute the checksum is defined in RFC 793:

The checksum field is the 16 bit one's complement of the one's complement sum of all 16-bit words in the header and text. If a segment contains an odd number of header and text octets to be checksummed, the last octet is padded on the right with zeros to form a 16-bit word for checksum purposes. The pad is not transmitted as part of the segment. While computing the checksum, the checksum field itself is replaced with zeros.

In other words, after appropriate padding, all 16-bit words are added using one's complement arithmetic. The sum is then bitwise complemented and inserted as the checksum field. A pseudo-header that mimics the IPv4 packet header used in the checksum computation is shown in the table below.

TCP pseudo-header for checksum computation (IPv4) [3]

TCP pseudo-header for checksum computation (IPv4)						
Bit offset	0–3	4–7	8–15	16–31		
0	Source address					
32	Destination address					
64	Zeros		Protocol	TCP length		
96	Source port		Destination port			
128	Sequence number					
160	Acknowledgement number					
192	Data offset	Reserved	Flags	Window		
224	Checksum			Urgent pointer		
256	Options (optional)					
256/288+	Data					

Figure 7.109: Transmission Control Protocol (TCP) Psuedo Header for IPv4 Checksum Calculation

- The source and destination addresses are those of the IPv4 header
- The protocol value is 6 for TCP (cf. List of IP protocol numbers)
- The TCP length field is the length of the TCP header and data (measured in octets).

TCP checksum for IPv6

When TCP runs over IPv6, the method used to compute the checksum is changed, as per RFC 2460:

Any transport or other upper-layer protocol that includes the addresses from the IP header in its checksum computation must be modified for use over IPv6, to include the 128-bit IPv6 addresses instead of 32-bit IPv4 addresses.

A pseudo-header that mimics the IPv6 header for computation of the checksum is shown below

TCP pseudo-header for checksum computation (IPv6) [4]

TCP pseudo-header for checksum computation (IPv6)						
Bit offset	0–7	8–15	16–23	24–31		
0	Source address					
32						
64						
96						
128	Destination address					
160						
192						
224						
256	TCP length					
288	Zeros			Next header		
320	Source port		Destination port			
352	Sequence number					
384	Acknowledgement number					
416	Data offset	Reserved	Flags	Window		
448	Checksum			Urgent pointer		
480	Options (optional)					
480/512+	Data					

Figure 7.110: Transmission Control Protocol (TCP) Psuedo Header for IPv6 Checksum Calculation

- Source address – the one in the IPv6 header
- Destination address – the final destination; if the IPv6 packet doesn't contain a Routing header, TCP uses the destination address in the IPv6 header, otherwise, at the originating node, it uses the address in the last element of the Routing header, and, at the receiving node, it uses the destination address in the IPv6 header
- TCP length – the length of the TCP header and data
- Next Header – the protocol value for TCP

Checksum offload

Many TCP/IP software stack implementations provide options to use hardware assistance to automatically compute the checksum in the network adapter prior to transmission onto the network or upon reception from the network for validation. This may relieve the OS from using precious CPU cycles calculating the checksum. Hence, overall network performance is increased

This feature may cause packet analyzers detecting outbound network traffic upstream of the network adapter that are unaware or uncertain about the use of checksum offload to report invalid checksum in outbound packets

[1] <http://microchip.wikitdot.com/tcpip:tcp-vs-udp>

[2] https://en.wikipedia.org/wiki/Transmission_Control_Protocol

[3] https://en.wikipedia.org/wiki/Transmission_Control_Protocol

For API Documentation:

See Also

[ProtocolPP::jprotocol](#)
[ProtocolPP::jarray](#)
[ProtocolPP::jrand](#)

For Additional Documentation:**See Also**

[jprotocol](#)
[jarray](#)
[jrand](#)

Protocol++ written by : John Peter Greninger © John Peter Greninger 2015-2019 All Rights Reserved

All copyrights and trademarks are the property of their respective owners

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer
- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution
- Any and all modifications must be returned to John Peter Greninger at GitHub.com <https://github.com/jpgreninger/protocolpp> for evaluation. Inclusion of modifications in the source code shall be determined solely by John Peter Greninger. Failure to provide modifications shall render this license NULL and VOID and revoke any rights to use of Protocol++
- Commercial use requires a fee-based license obtainable at www.protocolpp.com
- Academic use requires written and notarized permission from John Peter and Sheila Rocha Greninger
- US Copyrights at <https://www.copyright.gov/>
 - TXU002059872 (Version 1.0.0)
 - TXU002066632 (Version 1.2.7)
 - TXU002082674 (Version 1.4.0)
 - TXU002097880 (Version 2.0.0)

The names of its contributors may not be used to endorse or promote products derived from this software without specific prior written permission

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE

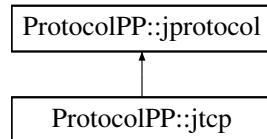
The documentation for this class was generated from the following file:

- [include/jtcp.h](#)

7.122 ProtocolPP::jtcp Class Reference

```
#include <jtcp.h>
```

Inheritance diagram for ProtocolPP::jtcp:



Public Types

- enum `tcpopt_t` {
 `END, NOP, MSS, WINSCALE,`
`SELACK, SACK, TIMESTAMP` }

Options that TCP may add to the standard header.

- enum `tcpstate_t` {
 `LISTEN, SYN_SENT, SYN_RCVD, ESTABLISHED,`
`FIN_WAIT_1, FIN_WAIT_2, CLOSE_WAIT, LAST_ACK,`
`TIME_WAIT, CLOSED` }

TCP states of operation.

Public Member Functions

- `jtcp (tcpstate_t state, std::shared_ptr< jtcpsta > &security)`
- `jtcp (tcpstate_t state, std::shared_ptr< jtcpsta > &security, std::string &file)`
- virtual `~jtcp ()`
Standard deconstructor.
- void `encap_packet (std::shared_ptr< jarray< uint8_t >> &input, std::shared_ptr< jarray< uint8_t >> &output)`
- void `decap_packet (std::shared_ptr< jarray< uint8_t >> &input, std::shared_ptr< jarray< uint8_t >> &output)`
- void `set_field (field_t field, uint64_t value)`
- void `set_hdr (jarray< uint8_t > &hdr)`
- void `add_option (tcpopt_t OPT, jarray< uint8_t > &value)`
- `uint64_t get_field (field_t field, jarray< uint8_t > &header)`
- `uint64_t get_field (field_t field)`
- `jarray< uint8_t > get_hdr ()`
- void `to_xml (tinyxml2::XMLPrinter &myxml, direction_t direction)`

Additional Inherited Members

7.122.1 Member Enumeration Documentation

7.122.1.1 enum ProtocolPP::jtcp::tcpopt_t

Options that TCP may add to the standard header.

Enumerator

END This option code indicates the end of the option list. This might not coincide with the end of the TCP header according to the Data Offset field. This is used at the end of all options not the end of each option,

and need only be used if the end of the options would not otherwise coincide with the end of the TCP header.

NOP This option code may be used between options, for example, to align the beginning of a subsequent option on a word boundary. There is no guarantee that senders will use this option, so receivers must be prepared to process options even if they do not begin on a word boundary.

MSS If this option is present, then it communicates the maximum receive segment size at the TCP which sends this segment. This field must only be sent in the initial connection request (i.e., in segments with the SYN control bit set). If this option is not used, any segment size is allowed.

WINSCALE Resize the window.

SELACK Select acknowledge.

SACK Acknowledge.

TIMESTAMP Timestamp.

7.122.1.2 enum ProtocolPP::jtcp::tcpstate_t

TCP states of operation.

Enumerator

LISTEN (Server) Waiting for connection request

SYN_SENT (Client) Connection request sent, waiting for server request

SYN_RCVD (Server) waiting for a confirming connection request acknowledgment having both received and sent a connection request

ESTABLISHED (BOTH) Open connection with data flowing. Normal state for a data connection

FIN_WAIT_1 (BOTH) represents waiting for a connection termination request from the remote TCP, or an acknowledgment of the connection termination request previously sent

FIN_WAIT_2 (BOTH) represents waiting for a connection termination request from the remote TCP

CLOSE_WAIT (BOTH) represents waiting for a connection termination request from the local user

LAST_ACK (BOTH) represents waiting for an acknowledgment of the connection termination request previously sent to the remote TCP (which includes an acknowledgment of its connection termination request)

TIME_WAIT (BOTH) represents waiting for enough time to pass to be sure the remote TCP received the acknowledgment of its connection termination request. [According to RFC 793 a connection can stay in TIME-WAIT for a maximum of four minutes known as two MSL (maximum segment lifetime).]

CLOSED (BOTH) represents no connection state at all

7.122.2 Constructor & Destructor Documentation

7.122.2.1 ProtocolPP::jtcp::jtcp (`tcpstate_t state, std::shared_ptr<jtcpsa> & security`)

constructor for normal TCP containing all necessary fields to setup and run a TCP connection. All fields are accessible with the [set_field\(\)](#) and [get_field\(\)](#) functions

Parameters

<code>state</code>	- Initial state of the engine (either LISTEN for SERVER or SYN_SENT for CLIENT). Can be ESTABLISHED
--------------------	---

<i>security</i>	- Parameters to setup a TCP flow
-----------------	----------------------------------

7.122.2.2 ProtocolPP::jtcp::jtcp (*tcpstate_t state*, std::shared_ptr<jtcpsa> &*security*, std::string &*file*)

constructor for normal TCP with file input containing all necessary fields to setup and run a TCP connection. All fields are accessible with the [set_field\(\)](#) and [get_field\(\)](#) functions

Parameters

<i>state</i>	- Initial state of the engine (either LISTEN for SERVER or SYN_SENT for CLIENT). Can be ESTABLISHED
<i>security</i>	- Parameters to setup a TCP flow
<i>file</i>	- path to file to use to source packet payloads

7.122.2.3 virtual ProtocolPP::jtcp::~jtcp() [inline], [virtual]

Standard deconstructor.

7.122.3 Member Function Documentation

7.122.3.1 void ProtocolPP::jtcp::add_option (*tcpopt_t OPT*, jarray<uint8_t> &*value*)

Allows the user to add options to the TCP header

Parameters

<i>OPT</i>	- option to add to the TCP header
<i>value</i>	- value of the option to add to the TCP header

7.122.3.2 void ProtocolPP::jtcp::decap_packet (std::shared_ptr<jarray<uint8_t>> &*input*, std::shared_ptr<jarray<uint8_t>> &*output*) [virtual]

This function is for use with the constructor without a file handle. Decap will produce a payload from the packet passed in check the flags, and update the window

Parameters

<i>input</i>	- TCP packet to decapsulate
<i>output</i>	- decapsulated payload

Implements [ProtocolPP::jprotocol](#).

7.122.3.3 void ProtocolPP::jtcp::encap_packet (std::shared_ptr<jarray<uint8_t>> &*input*, std::shared_ptr<jarray<uint8_t>> &*output*) [virtual]

This function is for use with the constructor without a file handle. Encap will produce a packet from the payload passed in

Parameters

<i>input</i>	- payload to encapsulate with TCP
--------------	-----------------------------------

<i>output</i>	- encapsulated TCP packet
---------------	---------------------------

Implements [ProtocolPP::jprotocol](#).

7.122.3.4 uint64_t ProtocolPP::jtcp::get_field (*field_t field, jarray<uint8_t> & header*) [virtual]

Retrieves the field from the TCP header

Parameters

<i>field</i>	- field from TCP header to retrieve
<i>header</i>	- TCP header to retrieve field from

Returns

field from the TCP header

Implements [ProtocolPP::jprotocol](#).

7.122.3.5 uint64_t ProtocolPP::jtcp::get_field (*field_t field*) [virtual]

Retrieves the field from the TCP security association

Parameters

<i>field</i>	- field from TCP security association to retrieve
--------------	---

Returns

field from the TCP association

Reimplemented from [ProtocolPP::jprotocol](#).

7.122.3.6 jarray<uint8_t> ProtocolPP::jtcp::get_hdr () [virtual]

Retrieves the complete TCP header

Returns

TCP header

Implements [ProtocolPP::jprotocol](#).

7.122.3.7 void ProtocolPP::jtcp::set_field (*field_t field, uint64_t value*) [virtual]

Allows the user to update a field in the header

Parameters

<i>field</i>	- field to update in the TCP header
<i>value</i>	- new value to add to the TCP header

Implements [ProtocolPP::jprotocol](#).

7.122.3.8 void ProtocolPP::jtcp::set_hdr (*jarray<uint8_t> & hdr*) [virtual]

Allows the user to update complete TCP header

Parameters

<i>hdr</i>	- new header to add to the TCP header
------------	---------------------------------------

Implements [ProtocolPP::jprotocol](#).

7.122.3.9 void [ProtocolPP::jtcp::to_xml](#)([tinyxml2::XMLPrinter](#) & *myxml*, [direction_t](#) *direction*) [virtual]

Writes the XML protocol and security objects to XML

Parameters

<i>myxml</i>	- XMLPrinter object
<i>direction</i>	- randomization

Implements [ProtocolPP::jprotocol](#).

The documentation for this class was generated from the following file:

- [include/jtcp.h](#)

7.123 jtcsa Class Reference

```
#include "include/jtcsa.h"
```

7.123.1 Detailed Description

7.123.2 Transport Control Protocol Security Association (TCP)

See https://en.wikipedia.org/wiki/Transmission_Control_Protocol

The Transmission Control Protocol (TCP) is a core protocol of the Internet protocol suite. It originated in the initial network implementation in which it complemented the Internet Protocol (IP). Therefore, the entire suite is commonly referred to as TCP/IP. TCP provides reliable, ordered, and error-checked delivery of a stream of octets between applications running on hosts communicating over an IP network. Major Internet applications such as the World Wide Web, email, remote administration and file transfer rely on TCP. Applications that do not require reliable data stream service may use the User Datagram Protocol (UDP), which provides a connectionless datagram service that emphasizes reduced latency over reliability.

TCP segments are sent as internet datagrams. The Internet Protocol header carries several information fields, including the source and destination host addresses [2]. A TCP header follows the internet header, supplying information specific to the TCP protocol. This division allows for the existence of host level protocols other than TCP (See RFC 793)

TCP Header Format [1]

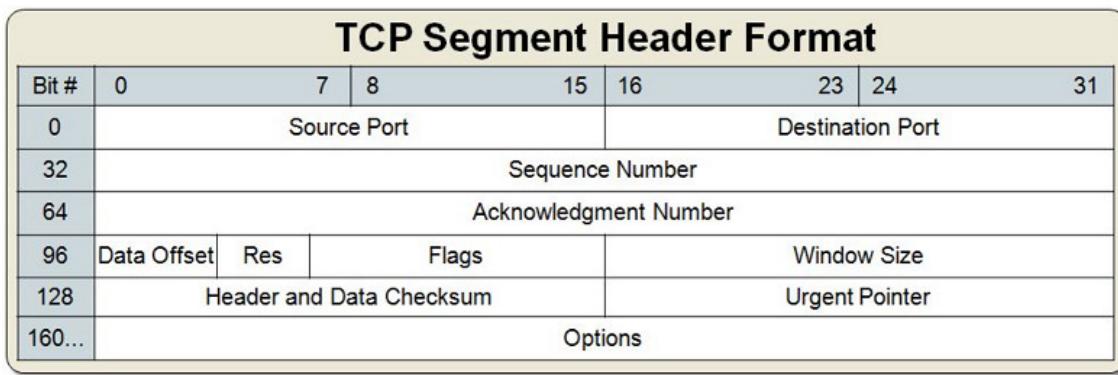


Figure 7.111: Transmission Control Protocol (TCP) Header Format

- Source Port: 16 bits
The source port number
- Destination Port: 16 bits
The destination port number
- Sequence Number: 32 bits
The sequence number of the first data octet in this segment (except when SYN is present). If SYN is present the sequence number is the initial sequence number (ISN) and the first data octet is ISN+1
- Acknowledgment Number: 32 bits
If the ACK control bit is set this field contains the value of the next sequence number the sender of the segment is expecting to receive. Once a connection is established this is always sent
- Data Offset: 4 bits
The number of 32 bit words in the TCP Header. This indicates where the data begins. The TCP header (even one including options) is an integral number of 32 bits long.
- Reserved: 6 bits
Reserved for future use. Must be zero
- Flags: 6 bits (from left to right):
 - URG: Urgent Pointer field significant
 - ACK: Acknowledgment field significant
 - PSH: Push Function
 - RST: Reset the connection
 - SYN: Synchronize sequence numbers
 - FIN: No more data from sender
- Window: 16 bits
The number of data octets beginning with the one indicated in the acknowledgment field which the sender of this segment is willing to accept
- Checksum: 16 bits
The checksum field is the 16 bit one's complement of the one's complement sum of all 16 bit words in the header and text. If a segment contains an odd number of header and text octets to be checksummed, the last octet is padded on the right with zeros to form a 16 bit word for checksum purposes. The pad is not transmitted as part of the segment. While computing the checksum, the checksum field itself is replaced with zeros

The checksum also covers a 96 bit pseudo header conceptually prefixed to the TCP header. This pseudo header contains the Source Address, the Destination Address, the Protocol, and TCP length. This gives the TCP protection against misrouted segments. This information is carried in the Internet Protocol and is transferred across the TCP/Network interface in the arguments or results of calls by the TCP on the IP

For API Documentation:**See Also**

[ProtocolPP::jprotocol](#)
[ProtocolPP::jarray](#)
[ProtocolPP::jrand](#)

For Additional Documentation:**See Also**

[jprotocol](#)
[jarray](#)
[jrand](#)

Protocol++ written by : John Peter Greninger © John Peter Greninger 2015-2019 All Rights Reserved

All copyrights and trademarks are the property of their respective owners

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

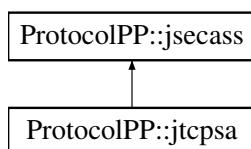
- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer
- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution
- Any and all modifications must be returned to John Peter Greninger at GitHub.com <https://github.com/jpgreninger/protocolpp> for evaluation. Inclusion of modifications in the source code shall be determined solely by John Peter Greninger. Failure to provide modifications shall render this license NULL and VOID and revoke any rights to use of Protocol++
- Commercial use requires a fee-based license obtainable at www.protocolpp.com
- Academic use requires written and notarized permission from John Peter and Sheila Rocha Greninger
- US Copyrights at <https://www.copyright.gov/>
 - TXu002059872 (Version 1.0.0)
 - TXu002066632 (Version 1.2.7)
 - TXu002082674 (Version 1.4.0)
 - TXu002097880 (Version 2.0.0)

The names of its contributors may not be used to endorse or promote products derived from this software without specific prior written permission

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY,

	Default Value	AY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE
• include/jtcpsa.h	1500 0xDEAD 0xBEEF 0x00000001 0x00000001 0	The documentation for this class was generated from the following file: #include <jtcpsa.h> 0
7.124 ProtocolPP::jtcpsa Class Reference	1500	
		Inheritance diagram for ProtocolPP::jtcpsa:

Table 7.62: TCP Defaults



Public Member Functions

- `jtcpsa()`
- `jtcpsa(direction_t dir, unsigned int mtu, uint16_t src, uint16_t dst, uint32_t seqnum, uint32_t acknum, uint8_t doffset, uint8_t flags, uint16_t window, uint16_t urgpointer, unsigned int timeout)`
- `jtcpsa(jtcpsa &rhs)`

Standard copy constructor.
- `jtcpsa(std::shared_ptr<jtcpsa> &rhs)`

Standard copy constructor from shared pointer.
- template<typename T>
void `set_field(field_t field, T fieldval)`
- template<typename T>
T `get_field(field_t field)`
- void `to_xml(tinyxml2::XMLPrinter &myxml, direction_t direction)`

7.124.1 Constructor & Destructor Documentation

7.124.1.1 ProtocolPP::jtcpsa::jtcpsa()

Standard constructor using default values

7.124.1.2 ProtocolPP::jtcpsa::jtcpsa(direction_t dir, unsigned int mtu, uint16_t src, uint16_t dst, uint32_t seqnum, uint32_t acknum, uint8_t doffset, uint8_t flags, uint16_t window, uint16_t urgpointer, unsigned int timeout)

TCP security association needed to setup a flow

Parameters

<code>dir</code>	- direction. Because TCP is bi-directional in all cases, this parameter is mainly used to decide if a file is being read from or written to (ENCAP or DECAP) and it keeps the interface consistent with other protocols
------------------	---

<i>mtu</i>	- Maximum Transmission Unit
<i>src</i>	- Source Port
<i>dst</i>	- Destination Port
<i>seqnum</i>	- Sequence Number
<i>acknum</i>	- Acknowledgement Number
<i>doffset</i>	- offset to where data begins
<i>flags</i>	- header flags in following order (RSVD, RSVD, URG, ACK, PSH, RST, SYN, FIN)
<i>window</i>	- size of the receive window
<i>urgpointer</i>	- points to the octet following urgent data
<i>timeout</i>	- Time to wait before closing idle socket

7.124.1.3 ProtocolPP::jtcpса::jtcpса (*jtcpса & rhs*)

Standard copy constructor.

7.124.1.4 ProtocolPP::jtcpса::jtcpса (std::shared_ptr<*jtcpса* > & *rhs*)

Standard copy constructor from shared pointer.

7.124.2 Member Function Documentation

7.124.2.1 template<typename T > T ProtocolPP::jtcpса::get_field (*field_t field*)

Retrieves a boolean value from the field requested. If the function is called with a field that is not a boolean, this function prints an error to the screen

Due to their dynamic nature, some fields are only available in jtcp which include the following fields

- LENGTH
- CHECKSUM
- STATE
 - LISTEN
 - SYN_SENT
 - SYN_RECV
 - FIN_WAIT_1
 - FIN_WAIT_2
 - CLOSE_WAIT
 - LAST_ACK
 - TIME_WAIT
 - CLOSED
- SNDUNA
- SNDNXT
- SNDWND
- SNDUP
- SNDW1
- SNDW2
- ISS
- RCVNXT
- RCVWND

	field name	Example
	DIRECTION <i>field</i> - field to set	set_field<ProtocolPP::direction_t>(ProtocolPP::field_t::DIRECTION, ProtocolPP::ENCAP)
Returns		
	MTU - boolean value from the field	set_field<uint32_t>(ProtocolPP::field_t::MTU, 1500)
7.124.2.2 template<typename T > void ProtocolPP::Ncpsa::set_field(field_t field, T fieldval) 0x00000001)	SEQNUM	set_field<uint32_t>(ProtocolPP::field_t::SEQNUM,
	Sets a boolean field in the security association. prints an error to the screen	set_field<uint32_t>(ProtocolPP::field_t::ACKNUM, 0x00000001)
Due to the dynamic nature, some fields are only available TCP which include the following fields	• LENGTH	set_field<uint32_t>(ProtocolPP::field_t::TIMEOUT, 0x00000001)
	• SOURCE	set_field<uint16_t>(ProtocolPP::field_t::SOURCE, 0xDEAD)
	• CHECKSUM	
	• STATE	
	DESTINATION – LISTEN	set_field<uint16_t>(ProtocolPP::field_t::DESTINATION, 0xBEEF)
	– SYN_SENT	
	WINDOW – SYN_RCVD	set_field<uint16_t>(ProtocolPP::field_t::WINDOW, 0x1000)
	– FIN_WAIT_1	
	URGPOINTER – FIN_WAIT_2	set_field<uint16_t>(ProtocolPP::field_t::URGPOINTER, 0x1000)
	– CLOSE_WAIT	
	DOFFSET – TIME_WAIT	set_field<uint8_t>(ProtocolPP::field_t::DOFFSET, 0x10)
	FLAGS – CLOSED	set_field<uint8_t>(ProtocolPP::field_t::FLAGS, 0x10)
	• SNDUNA	
	• SNDNXT Table 7.64: TCP Set Fields	
	• SNDWND	
	• SNDUP	
	• SNDW1	
	• SNDW2	
	• ISS	
	• RCVNXT	
	• RCVWND	
	• RCVUP	
	• IRS	
Parameters		

<i>field</i>	- field to set
<i>fieldval</i>	- boolean value for the field

7.124.2.3 void ProtocolPP::jtcpssa::to_xml (*tinyxml2::XMLPrinter & myxml*, *direction_t direction*) [virtual]

print the protocol and security objects as XML

Parameters

<i>myxml</i>	- object to print to
<i>direction</i>	- randomization

Implements [ProtocolPP::jsecass](#).

The documentation for this class was generated from the following file:

- [include/jtcpssa.h](#)

7.125 jtestbench Class Reference

```
#include "include/jtestbench.h"
```

7.125.1 Detailed Description

7.125.2 Testbench for Protocol++

See Also

[jring](#)
[jdata](#)
[jpacket](#)
[jstream](#)
[jprotocolpp](#)

Protocol++ written by : John Peter Greninger © John Peter Greninger 2015-2019 All Rights Reserved

All copyrights and trademarks are the property of their respective owners

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer
- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution
- Any and all modifications must be returned to John Peter Greninger at GitHub.com <https://github.com/jpgreninger/protocolpp> for evaluation. Inclusion of modifications in the source code shall be determined solely by John Peter Greninger. Failure to provide modifications shall render this license NULL and VOID and revoke any rights to use of Protocol++
- Commercial use requires a fee-based license obtainable at www.protocolpp.com
- Academic use requires written and notarized permission from John Peter and Sheila Rocha Greninger
- US Copyrights at [https://www.copyright.gov](http://www.copyright.gov)

- TXu002059872 (Version 1.0.0)
- TXu002066632 (Version 1.2.7)
- TXu002082674 (Version 1.4.0)
- TXu002097880 (Version 2.0.0)

The names of its contributors may not be used to endorse or promote products derived from this software without specific prior written permission

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE

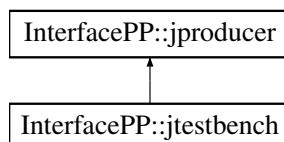
The documentation for this class was generated from the following file:

- [jtestbench/include/jtestbench.h](#)

7.126 InterfacePP::jtestbench Class Reference

```
#include <jtestbench.h>
```

Inheritance diagram for InterfacePP::jtestbench:



Public Member Functions

- [`jtestbench \(ProtocolPP::platform_t platform, unsigned long seed, int responders, std::shared_ptr< jmmu > &mmu, std::shared_ptr< jlogger > &logger, std::shared_ptr< ProtocolPP::jdata > &indata, std::shared_ptr< jrинг< ringflow >> &fring, std::shared_ptr< jrинг< ringin >> &iring, std::shared_ptr< jrинг< ringout >> &oring, ProtocolPP::endian_t endianess=ProtocolPP::BIG\)`](#)
- virtual [`~jtestbench \(\)`](#)
standard deconstructor
- [`uint64_t get_mem \(std::string packet, unsigned int length\)`](#)
- [`void write \(uint64_t address, uint32_t data\)`](#)
- [`void write \(uint64_t address, uint64_t data\)`](#)
- [`uint64_t write \(std::string packet, uint8_t *data, unsigned int length\)`](#)
- [`uint64_t write \(std::string packet, uint32_t *data, unsigned int length\)`](#)
- [`uint32_t read \(uint64_t address\)`](#)
- [`void read \(uint64_t address, uint8_t *data, unsigned int length\)`](#)
- [`void read \(uint64_t address, uint32_t *data, unsigned int length\)`](#)

Additional Inherited Members

7.126.1 Constructor & Destructor Documentation

```
7.126.1.1 InterfacePP::jtestbench::jtestbench ( ProtocolPP::platform_t platform, unsigned long seed, int responders,  
std::shared_ptr< jmmu > & mmu, std::shared_ptr< jlogger > & logger, std::shared_ptr< ProtocolPP::jdata  
> & indata, std::shared_ptr< jring< ringflow >> & fring, std::shared_ptr< jring< ringin >> & iring,  
std::shared_ptr< jring< ringout >> & oring, ProtocolPP::endian_t endianess = ProtocolPP::BIG )
```

jtestbench uses protocol++ to drive packets into a ring which can then be read by any device that supports a ring

Parameters

<i>platform</i>	- Platform to connect testbench to (W.A.S.P or SEC)
<i>seed</i>	- seed for the testbench
<i>responders</i>	- number of responders used
<i>mmu</i>	- track memory
<i>logger</i>	- object to print output
<i>indata</i>	- JDATA object containing flows and packets
<i>fring</i>	- Software ring for the flows
<i>iring</i>	- Software ring for the input packets
<i>oring</i>	- Software ring for the output packets
<i>endianess</i>	- Endianess of the platform to support

7.126.1.2 `virtual InterfacePP::jtestbench::~jtestbench() [inline], [virtual]`

standard deconstructor

7.126.2 Member Function Documentation

7.126.2.1 `uint64_t InterfacePP::jtestbench::get_mem(std::string packet, unsigned int length) [virtual]`

Retrieve pointer to uninitialized memory

Parameters

<i>packet</i>	- Name of packet to write data for
<i>length</i>	- Length in bytes for memory location

Returns

address to memory

Implements [InterfacePP::jproducer](#).

7.126.2.2 `uint32_t InterfacePP::jtestbench::read(uint64_t address) [virtual]`

Read function to overload in your testbench

Parameters

<i>address</i>	- Address to register
----------------	-----------------------

Returns

value of the register

Implements [InterfacePP::jproducer](#).

7.126.2.3 `void InterfacePP::jtestbench::read(uint64_t address, uint8_t * data, unsigned int length) [virtual]`

Read function to overload in your testbench

Parameters

<i>address</i>	- Address to read data from
<i>data</i>	- Retrieved data
<i>length</i>	- Length of data to read

Implements [InterfacePP::jproducer](#).

7.126.2.4 void InterfacePP::jtestbench::read (uint64_t *address*, uint32_t * *data*, unsigned int *length*) [virtual]

Read function to overload in your testbench

Parameters

<i>address</i>	- Address to read data from
<i>data</i>	- Retrieved data
<i>length</i>	- Length of data to read

Implements [InterfacePP::jproducer](#).

7.126.2.5 void InterfacePP::jtestbench::write (uint64_t *address*, uint32_t *data*) [virtual]

Write function to overload in your testbench

Parameters

<i>address</i>	- Address to write data to
<i>data</i>	- Data to write to memory

Implements [InterfacePP::jproducer](#).

7.126.2.6 void InterfacePP::jtestbench::write (uint64_t *address*, uint64_t *data*) [virtual]

Write function to overload in your testbench

Parameters

<i>address</i>	- Address to write data to
<i>data</i>	- Data to write to memory

Implements [InterfacePP::jproducer](#).

7.126.2.7 uint64_t InterfacePP::jtestbench::write (std::string *packet*, uint8_t * *data*, unsigned int *length*) [virtual]

Write function to overload in your testbench

Parameters

<i>packet</i>	- Name of packet to write data for
<i>data</i>	- Data to write to memory
<i>length</i>	- length of data to write

Implements [InterfacePP::jproducer](#).

7.126.2.8 uint64_t InterfacePP::jtestbench::write (std::string *packet*, uint32_t * *data*, unsigned int *length*) [virtual]

Write function to overload in your testbench

Parameters

<i>packet</i>	- Name of packet to write data for
<i>data</i>	- Word oriented data to write
<i>length</i>	- length of data to write

Implements [InterfacePP::jproducer](#).

The documentation for this class was generated from the following file:

- jtestbench/include/jtestbench.h

7.127 jtls Class Reference

```
#include "include/jtls.h"
```

7.127.1 Detailed Description

7.127.2 Transport Layer Security (TLS)

For additional information see http://en.wikipedia.org/wiki/Transport_Layer_Security

The TLS protocol exchanges records—which encapsulate the data to be exchanged in a specific format (see below). Each record can be compressed, padded, appended with a message authentication code (MAC), or encrypted, all depending on the state of the connection. Each record has a content type field that designates the type of data encapsulated, a length field and a TLS version field. The data encapsulated may be control or procedural messages of the TLS itself, or simply the application data needed to be transferred by TLS. The specifications (cipher suite, keys etc.) required to exchange application data by TLS, are agreed upon in the "TLS handshake" between the client requesting the data and the server responding to requests. The protocol therefore defines both the structure of payloads transferred in TLS and the procedure to establish and monitor the transfer

TLS handshake

When the connection starts, the record encapsulates a "control" protocol—the handshake messaging protocol (content type 22). This protocol is used to exchange all the information required by both sides for the exchange of the actual application data by TLS. It defines the messages formatting or containing this information and the order of their exchange. These may vary according to the demands of the client and server—i.e., there are several possible procedures to set up the connection. This initial exchange results in a successful TLS connection (both parties ready to transfer application data with TLS) or an alert message (as specified below)

7.127.2.1 Basic TLS handshake

A typical connection example follows, illustrating a handshake where the server (but not the client) is authenticated by its certificate:

1. Negotiation phase: A client sends a ClientHello message specifying the highest TLS protocol version it supports, a random number, a list of suggested cipher suites and suggested compression methods. If the client is attempting to perform a resumed handshake, it may send a session ID. The server responds with a ServerHello message, containing the chosen protocol version, a random number, CipherSuite and compression method from the choices offered by the client. To confirm or allow resumed handshakes the server may send a session ID. The chosen protocol version should be the highest that both the client and server support. For example, if the client supports TLS version 1.1 and the server supports version 1.2, version 1.1 should be selected; version 1.0 should not be selected. The server sends its Certificate message (depending on the selected cipher suite, this may be omitted by the server).[257] The server sends its ServerKeyExchange message (depending on the selected cipher suite, this may be omitted by the server). This message is sent for all DHE and DH_anon ciphersuites.[1] The server sends a ServerHelloDone message, indicating it is done with handshake negotiation. The client responds with a ClientKeyExchange message, which may contain a

PreMasterSecret, public key, or nothing. (Again, this depends on the selected cipher.) This PreMasterSecret is encrypted using the public key of the server certificate. The client and server then use the random numbers and PreMasterSecret to compute a common secret, called the "master secret". All other key data for this connection is derived from this master secret (and the client- and server-generated random values), which is passed through a carefully designed pseudorandom function.

2. The client now sends a ChangeCipherSpec record, essentially telling the server, "Everything I tell you from now on will be authenticated (and encrypted if encryption parameters were present in the server certificate)." The ChangeCipherSpec is itself a record-level protocol with content type of 20. Finally, the client sends an authenticated and encrypted Finished message, containing a hash and MAC over the previous handshake messages. The server will attempt to decrypt the client's Finished message and verify the hash and MAC. If the decryption or verification fails, the handshake is considered to have failed and the connection should be torn down.
3. Finally, the server sends a ChangeCipherSpec, telling the client, "Everything I tell you from now on will be authenticated (and encrypted, if encryption was negotiated)." The server sends its authenticated and encrypted Finished message. The client performs the same decryption and verification.
4. Application phase: at this point, the "handshake" is complete and the application protocol is enabled, with content type of 23. Application messages exchanged between client and server will also be authenticated and optionally encrypted exactly like in their Finished message. Otherwise, the content type will return 25 and the client will not authenticate.

Client-authenticated TLS handshake

The following full example shows a client being authenticated (in addition to the server as in the example above) via TLS using certificates exchanged between both peers.

1. Negotiation Phase: A client sends a ClientHello message specifying the highest TLS protocol version it supports, a random number, a list of suggested cipher suites and compression methods. The server responds with a ServerHello message, containing the chosen protocol version, a random number, cipher suite and compression method from the choices offered by the client. The server may also send a session id as part of the message to perform a resumed handshake. The server sends its Certificate message (depending on the selected cipher suite, this may be omitted by the server).[257] The server sends its ServerKeyExchange message (depending on the selected cipher suite, this may be omitted by the server). This message is sent for all DHE and DH_anon ciphersuites.[1] The server requests a certificate from the client, so that the connection can be mutually authenticated, using a CertificateRequest message. The server sends a ServerHelloDone message, indicating it is done with handshake negotiation. The client responds with a Certificate message, which contains the client's certificate. The client sends a ClientKeyExchange message, which may contain a PreMasterSecret, public key, or nothing. (Again, this depends on the selected cipher.) This PreMasterSecret is encrypted using the public key of the server certificate. The client sends a CertificateVerify message, which is a signature over the previous handshake messages using the client's certificate's private key. This signature can be verified by using the client's certificate's public key. This lets the server know that the client has access to the private key of the certificate and thus owns the certificate. The client and server then use the random numbers and PreMasterSecret to compute a common secret, called the "master secret". All other key data for this connection is derived from this master secret (and the client- and server-generated random values), which is passed through a carefully designed pseudorandom function.
2. The client now sends a ChangeCipherSpec record, essentially telling the server, "Everything I tell you from now on will be authenticated (and encrypted if encryption was negotiated)." The ChangeCipherSpec is itself a record-level protocol and has type 20 and not 22. Finally, the client sends an encrypted Finished message, containing a hash and MAC over the previous handshake messages. The server will attempt to decrypt the client's Finished message and verify the hash and MAC. If the decryption or verification fails, the handshake is considered to have failed and the connection should be torn down.
3. Finally, the server sends a ChangeCipherSpec, telling the client, "Everything I tell you from now on will be authenticated (and encrypted if encryption was negotiated)." The server sends its own encrypted Finished message. The client performs the same decryption and verification.
4. Application phase: at this point, the "handshake" is complete and the application protocol is enabled, with content type of 23. Application messages exchanged between client and server will also be encrypted exactly like in their Finished message.

Resumed TLS handshake

Public key operations (e.g., RSA) are relatively expensive in terms of computational power. TLS provides a secure shortcut in the handshake mechanism to avoid these operations: resumed sessions. Resumed sessions are implemented using session IDs or session tickets.

Apart from the performance benefit, resumed sessions can also be used for Single sign-on, as it guarantees that both the original session and any resumed session originate from the same client. This is of particular importance for the FTP over TLS/SSL protocol, which would otherwise suffer from a man-in-the-middle attack in which an attacker could intercept the contents of the secondary data connections.[258]

Session IDs

In an ordinary full handshake, the server sends a session id as part of the ServerHello message. The client associates this session id with the server's IP address and TCP port, so that when the client connects again to that server, it can use the session id to shortcut the handshake. In the server, the session id maps to the cryptographic parameters previously negotiated, specifically the "master secret". Both sides must have the same "master secret" or the resumed handshake will fail (this prevents an eavesdropper from using a session id). The random data in the ClientHello and ServerHello messages virtually guarantee that the generated connection keys will be different from in the previous connection. In the RFCs, this type of handshake is called an abbreviated handshake. It is also described in the literature as a restart handshake.

1. Negotiation phase: A client sends a ClientHello message specifying the highest TLS protocol version it supports, a random number, a list of suggested cipher suites and compression methods. Included in the message is the session id from the previous TLS connection. The server responds with a ServerHello message, containing the chosen protocol version, a random number, cipher suite and compression method from the choices offered by the client. If the server recognizes the session id sent by the client, it responds with the same session id. The client uses this to recognize that a resumed handshake is being performed. If the server does not recognize the session id sent by the client, it sends a different value for its session id. This tells the client that a resumed handshake will not be performed. At this point, both the client and server have the "master secret" and random data to generate the key data to be used for this connection.
2. The server now sends a ChangeCipherSpec record, essentially telling the client, "Everything I tell you from now on will be encrypted." The ChangeCipherSpec is itself a record-level protocol and has type 20 and not 22. Finally, the server sends an encrypted Finished message, containing a hash and MAC over the previous handshake messages. The client will attempt to decrypt the server's Finished message and verify the hash and MAC. If the decryption or verification fails, the handshake is considered to have failed and the connection should be torn down.
3. Finally, the client sends a ChangeCipherSpec, telling the server, "Everything I tell you from now on will be encrypted." The client sends its own encrypted Finished message. The server performs the same decryption and verification.
4. Application phase: at this point, the "handshake" is complete and the application protocol is enabled, with content type of 23. Application messages exchanged between client and server will also be encrypted exactly like in their Finished message.

Session tickets

RFC 5077 extends TLS via use of session tickets, instead of session IDs. It defines a way to resume a TLS session without requiring that session-specific state is stored at the TLS server. When using session tickets, the TLS server stores its session-specific state in a session ticket and sends the session ticket to the TLS client for storing. The client resumes a TLS session by sending the session ticket to the server, and the server resumes the TLS session according to the session-specific state in the ticket. The session ticket is encrypted and authenticated by the server, and the server verifies its validity before using its contents.

One particular weakness of this method with OpenSSL is that it always limits encryption and authentication security of the transmitted TLS session ticket to AES128-CBC-SHA256, no matter what other TLS parameters were negotiated for the actual TLS session.[248] This means that the state information (the TLS session ticket) is not as well protected as the TLS session itself. Of particular concern is OpenSSL's storage of the keys in an application-wide context (SSL_CTX), i.e. for the life of the application, and not allowing for re-keying of the AES128-CBC-SHA256 TLS session tickets without resetting the application-wide OpenSSL context (which is uncommon, error-prone and often requires manual administrative intervention).[249][247]

TLS Records

This is the general format for all TLS records (packets) [1]

Byte	+0	+1	+2	+3
0	Content type			
1..4	Version		Length	
5..n		Payload		
n..m		MAC		
m..p		Padding (block ciphers only)		

Major Version
0x03
0x03
0x03
0x03
0xFF

Code
0
1
2
4
11
12
13
14
15
16
20

Figure 7.112: General Message Format of TLS Records (Packets)

Content Type

This field identifies the Record Layer Protocol Type contained in this Record

Version

This field identifies the major and minor version of TLS for the contained message. For a ClientHello message, this need not be the highest version supported by the client

Length

The length of Protocol message(s), MAC and Padding, not to exceed 2^{14} bytes (16KB)

Protocol Message(s)

One or more messages identified by the Protocol field. Note that this field may be encrypted depending on the state of the connection

MAC and Padding

A message authentication code computed over the Protocol message, with additional key material included. Note that this field may be encrypted, or not included entirely, depending on the state of the connection. No MAC or Padding can be present at end of TLS records before all cipher algorithms and parameters have been negotiated and handshaked and then confirmed by sending a CipherStateChange record (see below) for signaling that these parameters will take effect in all further records sent by the same peer

Handshake protocol

Most messages exchanged during the setup of the TLS session are based on this record, unless an error or warning occurs and needs to be signaled by an Alert protocol record (see below), or the encryption mode of the session is modified by another record (see ChangeCipherSpec protocol below)

Message type

This field identifies the handshake message type

Handshake message data length

This is a 3-byte field indicating the length of the handshake data, not including the header. Note that multiple handshake messages may be combined within one record

Alert protocol

This record should normally not be sent during normal handshaking or application exchanges. However, this message can be sent at any time during the handshake and up to the closure of the session. If this is used to signal a fatal error, the session will be closed immediately after sending this record, so this record is used to give a reason for this closure. If the alert level is flagged as a warning, the remote can decide to close the session if it decides that the session is not reliable enough for its needs (before doing so, the remote may also send its own signal)

Level

Byte0	Byte1	Byte2	Byte3	
0x17	XXXXXX	XXXXXX	XXXXXX	the session immediately. Otherwise, the recipient may decide to terminate the session itself, by sending its own fatal alert and closing the session itself immediately after sending it. The use of Alert records is optional, however if it is missing before the session closure, the session may be resumed automatically (with its handshakes)
	Normal closure of a session after termination of the transported application should preferably be alerted with at least the Close notify Alert type (with a simple warning level) to prevent such automatic resume of a new session.	Version	Epoch	Signal explicitly the normal closure of a secure session before effectively closing its transport layer is useful to prevent or detect attacks (like attempts to truncate the securely transported data, if it intrinsically does not have a predetermined length or duration that the recipient of the secured data may expect)
	Seqnum	Seqnum	Seqnum	ChangeCipherSpec protocol
	CCS protocol type			
	Currently Seqnum		Length	
	Application protocol			
	Length	Application Data		
		Length of the application data (excluding the protocol header and including the MAC and padding trailers)		
	MAC	MAC (optional)		
		20 bytes for the SHA1 based HMAC, 16 bytes for the MD5 based HMAC		
	Padding	Padding (block ciphers only)		
		Variable length; last byte contains the padding length		

Table 7.72: DTLS Datagram
7.127.2.2 ChaCha20-Poly1305 Cipher Suites for Transport Layer Security (TLS)

This document describes the use of the ChaCha stream cipher and Poly1305 authenticator in version 1.2 or later of the Transport Layer Security (TLS) [RFC5246] protocol, as well as version 1.2 or later of the Datagram Transport Layer Security (DTLS) protocol [RFC6347].

ChaCha [CHACHA] is a stream cipher developed by D.J. Bernstein in 2008. It is a refinement of Salsa20, which is one of the selected ciphers in the eSTREAM portfolio [ESTREAM], and was used as the core of the SHA-3 finalist, BLAKE.

The variant of ChaCha used in this document has 20 rounds, a 96-bit nonce and a 256-bit key, and will be referred to as ChaCha20. This is the conservative variant (with respect to security) of the ChaCha family and is described in [RFC7539].

Poly1305 [POLY1305] is a Wegman-Carter, one-time authenticator designed by D.J. Bernstein. Poly1305 takes a 256-bit, one-time key and a message, and produces a 16-byte tag that authenticates the message such that an attacker has a negligible chance of producing a valid tag for an inauthentic message. It is also described in [RFC7539].

ChaCha and Poly1305 have both been designed for high performance in software implementations. They typically admit a compact implementation that uses few resources and inexpensive operations, which makes them suitable on a wide range of architectures. They have also been designed to minimize leakage of information through side channels.

Recent attacks [CBC-ATTACK] have indicated problems with the CBC-mode cipher suites in TLS and DTLS, as well as issues with the only supported stream cipher (RC4) [RC4-ATTACK]. While the existing AEAD cipher suites (based on AES-GCM) address some of these issues, there are concerns about their performance and ease of software implementation. Therefore, a new stream cipher to replace RC4 and address all the previous issues is needed. It is the purpose of this document to describe a secure stream cipher for both TLS and DTLS that is comparable to RC4 in speed on a wide range of platforms and can be implemented easily without being vulnerable to software side-channel attacks.

ChaCha20 Cipher Suites

The ChaCha20 and Poly1305 primitives are built into an AEAD algorithm [RFC5116], AEAD_CHACHA20_POLY1305, as described in [RFC7539]. This AEAD is incorporated into TLS and DTLS as specified in section 6.2.3.3

of [RFC5246]. AEAD_CHACHA20_POLY1305 requires a 96-bit nonce, which is formed as follows:

1. The 64-bit record sequence number is serialized as an 8-byte, big-endian value and padded on the left with four 0x00 bytes.
2. The padded sequence number is XORed with the client_write_IV (when the client is sending) or server_write_IV (when the server is sending). In DTLS, the 64-bit seq_num is the 16-bit epoch concatenated with the 48-bit seq_num. This nonce construction is different from the one used with AES-GCM in TLS 1.2 but matches the scheme expected to be used in TLS 1.3. The nonce is constructed from the record sequence number and shared secret, both of which are known to the recipient. The advantage is that no per-record, explicit nonce need be transmitted, which saves eight bytes per record and prevents implementations from mistakenly using a random nonce. Thus, in the terms of [RFC5246], SecurityParameters.fixed_iv_length is twelve bytes and SecurityParameters.record_iv_length is zero bytes.

The following cipher suites are defined.

- TLS_ECDHE_RSA_WITH_CHACHA20_POLY1305_SHA256 = {0xTBD, 0xTBD}
- TLS_ECDHE_ECDSA_WITH_CHACHA20_POLY1305_SHA256 = {0xTBD, 0xTBD}
- TLS_DHE_RSA_WITH_CHACHA20_POLY1305_SHA256 = {0xTBD, 0xTBD}
- TLS_PSK_WITH_CHACHA20_POLY1305_SHA256 = {0xTBD, 0xTBD}
- TLS_ECDHE_PSK_WITH_CHACHA20_POLY1305_SHA256 = {0xTBD, 0xTBD}
- TLS_DHE_PSK_WITH_CHACHA20_POLY1305_SHA256 = {0xTBD, 0xTBD}
- TLS_RSA_PSK_WITH_CHACHA20_POLY1305_SHA256 = {0xTBD, 0xTBD}

The DHE_RSA, ECDHE_RSA, ECDHE_ECDSA, PSK, ECDHE_PSK, DHE_PSK and RSA_PSK key exchanges for these cipher suites are unaltered and thus are performed as defined in [RFC5246], [RFC4492], and [RFC5489]. The pseudorandom function (PRF) for all the cipher suites defined in this document is the TLS PRF with SHA-256 as the hash function.

IANA Considerations

IANA is requested to add the following entries in the TLS Cipher Suite Registry:

- TLS_ECDHE_RSA_WITH_CHACHA20_POLY1305_SHA256 = {0xTBD, 0xTBD} {0xCC, 0xA8}
- TLS_ECDHE_ECDSA_WITH_CHACHA20_POLY1305_SHA256 = {0xTBD, 0xTBD} {0xCC, 0xA9}
- TLS_DHE_RSA_WITH_CHACHA20_POLY1305_SHA256 = {0xTBD, 0xTBD} {0xCC, 0xAA}
- TLS_PSK_WITH_CHACHA20_POLY1305_SHA256 = {0xTBD, 0xTBD} {0xCC, 0xAB}
- TLS_ECDHE_PSK_WITH_CHACHA20_POLY1305_SHA256 = {0xTBD, 0xTBD} {0xCC, 0xAC}
- TLS_DHE_PSK_WITH_CHACHA20_POLY1305_SHA256 = {0xTBD, 0xTBD} {0xCC, 0xAD}
- TLS_RSA_PSK_WITH_CHACHA20_POLY1305_SHA256 = {0xTBD, 0xTBD} {0xCC, 0xAE}

The cipher suite numbers listed in the second column are numbers used for cipher suite interoperability testing and it's suggested that IANA use these values for assignment.

7.127.2.3 Addition of Camellia Cipher Suites to Transport Layer Security (TLS)

Camellia was selected as a recommended cryptographic primitive by the EU NESSIE (New European Schemes for Signatures, Integrity and Encryption) project [NESSIE] and included in the list of cryptographic techniques for Japanese e-Government systems, which were selected by the Japan CRYPTREC (Cryptography Research and Evaluation Committees) [CRYPTREC]. Camellia is also included in specification of the TV-Anytime Forum [TV-ANYTIME]. The TV-Anytime Forum is an association of organizations that seeks to develop specifications to enable audio-visual and other services based on mass-market high-volume digital storage in consumer platforms. Camellia is specified as Cipher Suite in TLS used by Phase 1 S-7 (Bi-directional Metadata Delivery Protection) specification and S-5 (TV-Anytime Rights Management and Protection Information for Broadcast Applications) specification. Camellia has been submitted to other several standardization bodies such as ISO (ISO/IEC 18033) and IETF S/MIME Mail Security Working Group [Camellia-CMS].

Camellia supports 128-bit block size and 128-, 192-, and 256-bit key sizes; i.e., the same interface specifications as the Advanced Encryption Standard (AES) [AES]. Camellia was jointly developed by NTT and Mitsubishi Electric Corporation in 2000 [CamelliaTech]. It was carefully designed to withstand all known cryptanalytic attacks and even to have a sufficiently large security leeway. It has been scrutinized by worldwide cryptographic experts.

Camellia was also designed to be suitable for both software and hardware implementations and to cover all possible encryption applications, from low-cost smart cards to high-speed network systems. Compared to the AES, Camellia offers at least comparable encryption speed in software and hardware. In addition, a distinguishing feature is its small hardware design. Camellia perfectly meets one of the current TLS market requirements, for which low power consumption is mandatory.

The algorithm specification and object identifiers are described in [Camellia-Desc]. The Camellia homepage, <http://info.isl.ntt.co.jp/camellia/>, contains a wealth of information about camellia, including detailed specification, security analysis, performance figures, reference implementation, and test vectors.

Proposed Cipher Suites

The new cipher suites proposed here have the following definitions:

- CipherSuite TLS_RSA_WITH_CAMELLIA_128_CBC_SHA = { 0x00,0x41 };
- CipherSuite TLS_DH_DSS_WITH_CAMELLIA_128_CBC_SHA = { 0x00,0x42 };
- CipherSuite TLS_DH_RSA_WITH_CAMELLIA_128_CBC_SHA = { 0x00,0x43 };
- CipherSuite TLS_DHE_DSS_WITH_CAMELLIA_128_CBC_SHA = { 0x00,0x44 };
- CipherSuite TLS_DHE_RSA_WITH_CAMELLIA_128_CBC_SHA = { 0x00,0x45 };
- CipherSuite TLS_DH_anon_WITH_CAMELLIA_128_CBC_SHA = { 0x00,0x46 };
- CipherSuite TLS_RSA_WITH_CAMELLIA_256_CBC_SHA = { 0x00,0x84 };
- CipherSuite TLS_DH_DSS_WITH_CAMELLIA_256_CBC_SHA = { 0x00,0x85 };
- CipherSuite TLS_DH_RSA_WITH_CAMELLIA_256_CBC_SHA = { 0x00,0x86 };
- CipherSuite TLS_DHE_DSS_WITH_CAMELLIA_256_CBC_SHA = { 0x00,0x87 };
- CipherSuite TLS_DHE_RSA_WITH_CAMELLIA_256_CBC_SHA = { 0x00,0x88 };
- CipherSuite TLS_DH_anon_WITH_CAMELLIA_256_CBC_SHA = { 0x00,0x89 };

Cipher

All the cipher suites described here use Camellia in cipher block chaining (CBC) mode as a bulk cipher algorithm. Camellia is a 128 bit block cipher with 128-, 192-, and 256-bit key sizes; i.e., it supports the same block and key sizes as the Advanced Encryption Standard (AES). However, this document only defines cipher suites for 128- and 256-bit keys as well as AES cipher suites for TLS [AES-TLS]. These cipher suites are efficient and practical enough for most uses, including high-security applications.

Hash

All the cipher suites described here use SHA-1 [SHA-1] in a Hashed Message Authentication Code (HMAC) construction, as described in section 5 of [TLS].

Key Exchange

The cipher suites defined here differ in the type of certificate and key exchange method. They use the following options:

Cipher Suite Key Exchange Algorithm

- TLS_RSA_WITH_CAMELLIA_128_CBC_SHA RSA
- TLS_DH_DSS_WITH_CAMELLIA_128_CBC_SHA DH_DSS
- TLS_DH_RSA_WITH_CAMELLIA_128_CBC_SHA DH_RSA
- TLS_DHE_DSS_WITH_CAMELLIA_128_CBC_SHA DHE_DSS
- TLS_DHE_RSA_WITH_CAMELLIA_128_CBC_SHA DHE_RSA
- TLS_DH_anon_WITH_CAMELLIA_128_CBC_SHA DH_anon
- TLS_RSA_WITH_CAMELLIA_256_CBC_SHA RSA
- TLS_DH_DSS_WITH_CAMELLIA_256_CBC_SHA DH_DSS
- TLS_DH_RSA_WITH_CAMELLIA_256_CBC_SHA DH_RSA
- TLS_DHE_DSS_WITH_CAMELLIA_256_CBC_SHA DHE_DSS
- TLS_DHE_RSA_WITH_CAMELLIA_256_CBC_SHA DHE_RSA
- TLS_DH_anon_WITH_CAMELLIA_256_CBC_SHA DH_anon

7.127.2.4 Addition of the ARIA Cipher Suites to Transport Layer Security (TLS)

ARIA

ARIA is a general-purpose block cipher algorithm developed by Korean cryptographers in 2003. It is an iterated block cipher with 128-, 192-, and 256-bit keys and encrypts 128-bit blocks in 12, 14, and 16 rounds, depending on the key size. It is secure and suitable for most software and hardware implementations on 32-bit and 8-bit processors. It was established as a Korean standard block cipher algorithm in 2004 [ARIAKS] and has been widely used in Korea, especially for government-to-public services. It was included in PKCS #11 in 2007 [ARIAPKCS]. The algorithm specification and object identifiers are described in [RFC5794].

Proposed Cipher Suites

The first twenty cipher suites use ARIA [RFC5794] in Cipher Block Chaining (CBC) mode with a SHA-2 family Hashed Message Authentication Code (HMAC). Eight out of twenty use elliptic curves.

- CipherSuite TLS_RSA_WITH_ARIA_128_CBC_SHA256 = { 0xC0,0x3C };
- CipherSuite TLS_RSA_WITH_ARIA_256_CBC_SHA384 = { 0xC0,0x3D };
- CipherSuite TLS_DH_DSS_WITH_ARIA_128_CBC_SHA256 = { 0xC0,0x3E };
- CipherSuite TLS_DH_DSS_WITH_ARIA_256_CBC_SHA384 = { 0xC0,0x3F };
- CipherSuite TLS_DH_RSA_WITH_ARIA_128_CBC_SHA256 = { 0xC0,0x40 };
- CipherSuite TLS_DH_RSA_WITH_ARIA_256_CBC_SHA384 = { 0xC0,0x41 };
- CipherSuite TLS_DHE_DSS_WITH_ARIA_128_CBC_SHA256 = { 0xC0,0x42 };
- CipherSuite TLS_DHE_DSS_WITH_ARIA_256_CBC_SHA384 = { 0xC0,0x43 };
- CipherSuite TLS_DHE_RSA_WITH_ARIA_128_CBC_SHA256 = { 0xC0,0x44 };

- CipherSuite TLS_DHE_RSA_WITH_ARIA_256_CBC_SHA384 = { 0xC0,0x45 };
- CipherSuite TLS_DH_anon_WITH_ARIA_128_CBC_SHA256 = { 0xC0,0x46 };
- CipherSuite TLS_DH_anon_WITH_ARIA_256_CBC_SHA384 = { 0xC0,0x47 };
- CipherSuite TLS_ECDHE_ECDSA_WITH_ARIA_128_CBC_SHA256 = { 0xC0,0x48 };
- CipherSuite TLS_ECDHE_ECDSA_WITH_ARIA_256_CBC_SHA384 = { 0xC0,0x49 };
- CipherSuite TLS_ECDH_ECDSA_WITH_ARIA_128_CBC_SHA256 = { 0xC0,0x4A };
- CipherSuite TLS_ECDH_ECDSA_WITH_ARIA_256_CBC_SHA384 = { 0xC0,0x4B };
- CipherSuite TLS_ECDHE_RSA_WITH_ARIA_128_CBC_SHA256 = { 0xC0,0x4C };
- CipherSuite TLS_ECDHE_RSA_WITH_ARIA_256_CBC_SHA384 = { 0xC0,0x4D };
- CipherSuite TLS_ECDH_RSA_WITH_ARIA_128_CBC_SHA256 = { 0xC0,0x4E };
- CipherSuite TLS_ECDH_RSA_WITH_ARIA_256_CBC_SHA384 = { 0xC0,0x4F };

GCM-Based Cipher Suites

The next twenty cipher suites use the same asymmetric algorithms as those in the previous section but use the authenticated encryption modes defined in TLS 1.2 with the ARIA in Galois Counter Mode (GCM) [GCM].

- CipherSuite TLS_RSA_WITH_ARIA_128_GCM_SHA256 = { 0xC0,0x50 };
- CipherSuite TLS_RSA_WITH_ARIA_256_GCM_SHA384 = { 0xC0,0x51 };
- CipherSuite TLS_DHE_RSA_WITH_ARIA_128_GCM_SHA256 = { 0xC0,0x52 };
- CipherSuite TLS_DHE_RSA_WITH_ARIA_256_GCM_SHA384 = { 0xC0,0x53 };
- CipherSuite TLS_DH_RSA_WITH_ARIA_128_GCM_SHA256 = { 0xC0,0x54 };
- CipherSuite TLS_DH_RSA_WITH_ARIA_256_GCM_SHA384 = { 0xC0,0x55 };
- CipherSuite TLS_DHE_DSS_WITH_ARIA_128_GCM_SHA256 = { 0xC0,0x56 };
- CipherSuite TLS_DHE_DSS_WITH_ARIA_256_GCM_SHA384 = { 0xC0,0x57 };
- CipherSuite TLS_DH_DSS_WITH_ARIA_128_GCM_SHA256 = { 0xC0,0x58 };
- CipherSuite TLS_DH_DSS_WITH_ARIA_256_GCM_SHA384 = { 0xC0,0x59 };
- CipherSuite TLS_DH_anon_WITH_ARIA_128_GCM_SHA256 = { 0xC0,0x5A };
- CipherSuite TLS_DH_anon_WITH_ARIA_256_GCM_SHA384 = { 0xC0,0x5B };
- CipherSuite TLS_ECDHE_ECDSA_WITH_ARIA_128_GCM_SHA256 = { 0xC0,0x5C };
- CipherSuite TLS_ECDHE_ECDSA_WITH_ARIA_256_GCM_SHA384 = { 0xC0,0x5D };
- CipherSuite TLS_ECDH_ECDSA_WITH_ARIA_128_GCM_SHA256 = { 0xC0,0x5E };
- CipherSuite TLS_ECDH_ECDSA_WITH_ARIA_256_GCM_SHA384 = { 0xC0,0x5F };
- CipherSuite TLS_ECDHE_RSA_WITH_ARIA_128_GCM_SHA256 = { 0xC0,0x60 };
- CipherSuite TLS_ECDHE_RSA_WITH_ARIA_256_GCM_SHA384 = { 0xC0,0x61 };
- CipherSuite TLS_ECDH_RSA_WITH_ARIA_128_GCM_SHA256 = { 0xC0,0x62 };
- CipherSuite TLS_ECDH_RSA_WITH_ARIA_256_GCM_SHA384 = { 0xC0,0x63 };

PSK Cipher Suites

The next fourteen cipher suites describe PSK cipher suites. Eight cipher suites use an HMAC and six cipher suites use the ARIA Galois Counter Mode.

- CipherSuite TLS_PSK_WITH_ARIA_128_CBC_SHA256 = { 0xC0,0x64 };
- CipherSuite TLS_PSK_WITH_ARIA_256_CBC_SHA384 = { 0xC0,0x65 };
- CipherSuite TLS_DHE_PSK_WITH_ARIA_128_CBC_SHA256 = { 0xC0,0x66 };
- CipherSuite TLS_DHE_PSK_WITH_ARIA_256_CBC_SHA384 = { 0xC0,0x67 };
- CipherSuite TLS_RSA_PSK_WITH_ARIA_128_CBC_SHA256 = { 0xC0,0x68 };
- CipherSuite TLS_RSA_PSK_WITH_ARIA_256_CBC_SHA384 = { 0xC0,0x69 };
- CipherSuite TLS_PSK_WITH_ARIA_128_GCM_SHA256 = { 0xC0,0x6A };
- CipherSuite TLS_PSK_WITH_ARIA_256_GCM_SHA384 = { 0xC0,0x6B };
- CipherSuite TLS_DHE_PSK_WITH_ARIA_128_GCM_SHA256 = { 0xC0,0x6C };
- CipherSuite TLS_DHE_PSK_WITH_ARIA_256_GCM_SHA384 = { 0xC0,0x6D };
- CipherSuite TLS_RSA_PSK_WITH_ARIA_128_GCM_SHA256 = { 0xC0,0x6E };
- CipherSuite TLS_RSA_PSK_WITH_ARIA_256_GCM_SHA384 = { 0xC0,0x6F };
- CipherSuite TLS_ECDHE_PSK_WITH_ARIA_128_CBC_SHA256 = { 0xC0,0x70 };
- CipherSuite TLS_ECDHE_PSK_WITH_ARIA_256_CBC_SHA384 = { 0xC0,0x71 };

Cipher

The ARIA_128_CBC cipher suites use ARIA [RFC5794] in CBC mode with a 128-bit key and 128-bit Initialization Vector (IV); the ARIA_256_CBC cipher suites use a 256-bit key and 128-bit IV. AES-authenticated encryption with additional data algorithms, AEAD_AES_128_GCM, and AEAD_AES_256_GCM are described in [RFC5116]. AES GCM cipher suites for TLS are described in [RFC5288]. AES and ARIA share common characteristics, including key sizes and block length. ARIA_128_GCM and ARIA_256_GCM are defined according to those characteristics of AES.

IANA Considerations

IANA has allocated the following numbers in the TLS Cipher Suite Registry:

- CipherSuite TLS_RSA_WITH_ARIA_128_CBC_SHA256 = { 0xC0,0x3C };
- CipherSuite TLS_RSA_WITH_ARIA_256_CBC_SHA384 = { 0xC0,0x3D };
- CipherSuite TLS_DH_DSS_WITH_ARIA_128_CBC_SHA256 = { 0xC0,0x3E };
- CipherSuite TLS_DH_DSS_WITH_ARIA_256_CBC_SHA384 = { 0xC0,0x3F };
- CipherSuite TLS_DH_RSA_WITH_ARIA_128_CBC_SHA256 = { 0xC0,0x40 };
- CipherSuite TLS_DH_RSA_WITH_ARIA_256_CBC_SHA384 = { 0xC0,0x41 };
- CipherSuite TLS_DHE_DSS_WITH_ARIA_128_CBC_SHA256 = { 0xC0,0x42 };
- CipherSuite TLS_DHE_DSS_WITH_ARIA_256_CBC_SHA384 = { 0xC0,0x43 };
- CipherSuite TLS_DHE_RSA_WITH_ARIA_128_CBC_SHA256 = { 0xC0,0x44 };
- CipherSuite TLS_DHE_RSA_WITH_ARIA_256_CBC_SHA384 = { 0xC0,0x45 };
- CipherSuite TLS_DH_anon_WITH_ARIA_128_CBC_SHA256 = { 0xC0,0x46 };

- CipherSuite TLS_DH_anon_WITH_ARIA_256_CBC_SHA384 = { 0xC0,0x47 };
- CipherSuite TLS_ECDHE_ECDSA_WITH_ARIA_128_CBC_SHA256 = { 0xC0,0x48 };
- CipherSuite TLS_ECDHE_ECDSA_WITH_ARIA_256_CBC_SHA384 = { 0xC0,0x49 };
- CipherSuite TLS_ECDH_ECDSA_WITH_ARIA_128_CBC_SHA256 = { 0xC0,0x4A };
- CipherSuite TLS_ECDH_ECDSA_WITH_ARIA_256_CBC_SHA384 = { 0xC0,0x4B };
- CipherSuite TLS_ECDHE_RSA_WITH_ARIA_128_CBC_SHA256 = { 0xC0,0x4C };
- CipherSuite TLS_ECDHE_RSA_WITH_ARIA_256_CBC_SHA384 = { 0xC0,0x4D };
- CipherSuite TLS_ECDH_RSA_WITH_ARIA_128_CBC_SHA256 = { 0xC0,0x4E };
- CipherSuite TLS_ECDH_RSA_WITH_ARIA_256_CBC_SHA384 = { 0xC0,0x4F };
- CipherSuite TLS_RSA_WITH_ARIA_128_GCM_SHA256 = { 0xC0,0x50 };
- CipherSuite TLS_RSA_WITH_ARIA_256_GCM_SHA384 = { 0xC0,0x51 };
- CipherSuite TLS_DHE_RSA_WITH_ARIA_128_GCM_SHA256 = { 0xC0,0x52 };
- CipherSuite TLS_DHE_RSA_WITH_ARIA_256_GCM_SHA384 = { 0xC0,0x53 };
- CipherSuite TLS_DH_RSA_WITH_ARIA_128_GCM_SHA256 = { 0xC0,0x54 };
- CipherSuite TLS_DH_RSA_WITH_ARIA_256_GCM_SHA384 = { 0xC0,0x55 };
- CipherSuite TLS_DHE_DSS_WITH_ARIA_128_GCM_SHA256 = { 0xC0,0x56 };
- CipherSuite TLS_DHE_DSS_WITH_ARIA_256_GCM_SHA384 = { 0xC0,0x57 };
- CipherSuite TLS_DH_DSS_WITH_ARIA_128_GCM_SHA256 = { 0xC0,0x58 };
- CipherSuite TLS_DH_DSS_WITH_ARIA_256_GCM_SHA384 = { 0xC0,0x59 };
- CipherSuite TLS_DH_anon_WITH_ARIA_128_GCM_SHA256 = { 0xC0,0x5A };
- CipherSuite TLS_DH_anon_WITH_ARIA_256_GCM_SHA384 = { 0xC0,0x5B };
- CipherSuite TLS_ECDHE_ECDSA_WITH_ARIA_128_GCM_SHA256 = { 0xC0,0x5C };
- CipherSuite TLS_ECDHE_ECDSA_WITH_ARIA_256_GCM_SHA384 = { 0xC0,0x5D };
- CipherSuite TLS_ECDH_ECDSA_WITH_ARIA_128_GCM_SHA256 = { 0xC0,0x5E };
- CipherSuite TLS_ECDH_ECDSA_WITH_ARIA_256_GCM_SHA384 = { 0xC0,0x5F };
- CipherSuite TLS_ECDHE_RSA_WITH_ARIA_128_GCM_SHA256 = { 0xC0,0x60 };
- CipherSuite TLS_ECDHE_RSA_WITH_ARIA_256_GCM_SHA384 = { 0xC0,0x61 };
- CipherSuite TLS_ECDH_RSA_WITH_ARIA_128_GCM_SHA256 = { 0xC0,0x62 };
- CipherSuite TLS_ECDH_RSA_WITH_ARIA_256_GCM_SHA384 = { 0xC0,0x63 };
- CipherSuite TLS_PSK_WITH_ARIA_128_CBC_SHA256 = { 0xC0,0x64 };
- CipherSuite TLS_PSK_WITH_ARIA_256_CBC_SHA384 = { 0xC0,0x65 };
- CipherSuite TLS_DHE_PSK_WITH_ARIA_128_CBC_SHA256 = { 0xC0,0x66 };
- CipherSuite TLS_DHE_PSK_WITH_ARIA_256_CBC_SHA384 = { 0xC0,0x67 };
- CipherSuite TLS_RSA_PSK_WITH_ARIA_128_CBC_SHA256 = { 0xC0,0x68 };
- CipherSuite TLS_RSA_PSK_WITH_ARIA_256_CBC_SHA384 = { 0xC0,0x69 };
- CipherSuite TLS_PSK_WITH_ARIA_128_GCM_SHA256 = { 0xC0,0x6A };

- CipherSuite TLS_PSK_WITH_ARIA_256_GCM_SHA384 = { 0xC0,0x6B };
- CipherSuite TLS_DHE_PSK_WITH_ARIA_128_GCM_SHA256 = { 0xC0,0x6C };
- CipherSuite TLS_DHE_PSK_WITH_ARIA_256_GCM_SHA384 = { 0xC0,0x6D };
- CipherSuite TLS_RSA_PSK_WITH_ARIA_128_GCM_SHA256 = { 0xC0,0x6E };
- CipherSuite TLS_RSA_PSK_WITH_ARIA_256_GCM_SHA384 = { 0xC0,0x6F };
- CipherSuite TLS_ECDHE_PSK_WITH_ARIA_128_CBC_SHA256 = { 0xC0,0x70 };
- CipherSuite TLS_ECDHE_PSK_WITH_ARIA_256_CBC_SHA384 = { 0xC0,0x71 };

7.127.2.5 Addition of SEED Cipher Suites to Transport Layer Security (TLS)

SEED is a symmetric encryption algorithm that was developed by Korea Information Security Agency (KISA) and a group of experts, beginning in 1998. The input/output block size of SEED is 128-bit and the key length is also 128-bit. SEED has the 16-round Feistel structure. A 128-bit input is divided into two 64-bit blocks and the right 64-bit block is an input to the round function with a 64-bit subkey generated from the key scheduling.

SEED is easily implemented in various software and hardware because it is designed to increase the efficiency of memory storage and the simplicity of generating keys without degrading the security of the algorithm. In particular, it can be effectively adopted in a computing environment that has a restricted resources such as mobile devices, smart cards, and so on.

SEED is a national industrial association standard [TTASSEED] and is widely used in South Korea for electronic commerce and financial services operated on wired & wireless PKI.

The algorithm specification and object identifiers are described in [SEED-ALG]. The SEED homepage, http://www.kisa.or.kr/seed/seed_eng.html, contains a wealth of information about SEED, including detailed specification, evaluation report, test vectors, and so on.

Proposed Cipher Suites

The new cipher suites proposed here have the following definitions:

- CipherSuite TLS_RSA_WITH_SEED_CBC_SHA = { 0x00, 0x96 };
- CipherSuite TLS_DH_DSS_WITH_SEED_CBC_SHA = { 0x00, 0x97 };
- CipherSuite TLS_DH_RSA_WITH_SEED_CBC_SHA = { 0x00, 0x98 };
- CipherSuite TLS_DHE_DSS_WITH_SEED_CBC_SHA = { 0x00, 0x99 };
- CipherSuite TLS_DHE_RSA_WITH_SEED_CBC_SHA = { 0x00, 0x9A };
- CipherSuite TLS_DH_anon_WITH_SEED_CBC_SHA = { 0x00, 0x9B };

Key Exchange

The cipher suites defined here differ in the type of certificate and key exchange method. They use the following options:

CipherSuite Key Exchange Algorithm

- TLS_RSA_WITH_SEED_CBC_SHA RSA
- TLS_DH_DSS_WITH_SEED_CBC_SHA DH_DSS
- TLS_DH_RSA_WITH_SEED_CBC_SHA DH_RSA
- TLS_DHE_DSS_WITH_SEED_CBC_SHA DHE_DSS
- TLS_DHE_RSA_WITH_SEED_CBC_SHA DHE_RSA
- TLS_DH_anon_WITH_SEED_CBC_SHA DH_anon

7.127.2.6 Encrypt-then-MAC for Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS) [2]

This document describes a means of negotiating the use of the encrypt-then-MAC security mechanism in place of the existing MAC then-encrypt mechanism in Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS). The MAC-then-encrypt mechanism has been the subject of a number of security vulnerabilities over a period of many years

Negotiating Encrypt-then-MAC

The use of encrypt-then-MAC is negotiated via TLS/DTLS extensions as defined in TLS. On connecting, the client includes the `encrypt_then_mac` extension in its `client_hello` if it wishes to use encrypt-then-MAC rather than the default MAC-then-encrypt. If the server is capable of meeting this requirement, it responds with an `encrypt_then_mac` in its `server_hello`. The "extension_type" value for this extension SHALL be 22 (0x16), and the "extension_data" field of this extension SHALL be empty. The client and server MUST NOT use encrypt-then-MAC unless both sides have successfully exchanged `encrypt_then_mac` extensions

Rationale

The use of TLS/DTLS extensions to negotiate an overall switch is preferable to defining new ciphersuites because the latter would result in a Cartesian explosion of suites, potentially requiring duplicating every single existing suite with a new one that uses encrypt-then-MAC. In contrast, the approach presented here requires just a single new extension type with a corresponding minimal-length extension sent by client and server

Another possibility for introducing encrypt-then-MAC would be to make it part of TLS 1.3; however, this would require the implementation and deployment of all of TLS 1.2 just to support a trivial code change in the order of encryption and MAC'ing. In contrast, deploying encrypt-then-MAC via the TLS/DTLS extension mechanism required changing less than a dozen lines of code in one implementation (not including the handling for the new extension type, which was a further 50 or so lines of code)

The use of extensions precludes use with SSL 3.0, but then it's likely that anything still using that protocol, which is nearly two decades old, will be vulnerable to any number of other attacks anyway, so there seems little point in bending over backwards to accommodate SSL 3.0

Applying Encrypt-then-MAC

Once the use of encrypt-then-MAC has been negotiated, processing of TLS/DTLS packets switches from the standard:

- `encrypt(data || MAC || pad)`

to the new:

- `encrypt(data || pad) || MAC`

with the MAC covering the entire packet up to the start of the MAC value. In TLS notation, the MAC calculation for TLS 1.0 without the explicit Initialization Vector (IV) is:

```
* * * MAC (MAC_write_key, seq_num +
*           TLSCipherText.type +
*           TLSCipherText.version +
*           TLSCipherText.length +
*           ENC (content + padding + padding_length));
*
```

and for TLS 1.1 and greater with an explicit IV is:

```
* * * MAC (MAC_write_key, seq_num +
*           TLSCipherText.type +
*           TLSCipherText.version +
*           TLSCipherText.length +
*           IV +
*           ENC (content + padding + padding_length));
*
```

(For DTLS, the sequence number is replaced by the combined epoch and sequence number as per DTLS) The final MAC value is then appended to the encrypted data and padding. This calculation is identical to the existing one,

with the exception that the MAC calculation is run over the payload ciphertext (the TLSCipherText PDU) rather than the plaintext (the TLSCompressed PDU)

The overall TLS packet [2] is then:

```
* * struct {
*     ContentType type;
*     ProtocolVersion version;
*     uint16 length;
*     GenericBlockCipher fragment;
*     opaque MAC;
* } TLSCiphertext;
```

The equivalent DTLS packet is then:

```
* * struct {
*     ContentType type;
*     ProtocolVersion version;
*     uint16 epoch;
*     uint48 sequence_number;
*     uint16 length;
*     GenericBlockCipher fragment;
*     opaque MAC;
* } TLSCiphertext;
```

This is identical to the existing TLS/DTLS layout, with the only difference being that the MAC value is moved outside the encrypted data

Note from the GenericBlockCipher annotation that this only applies to standard block ciphers that have distinct encrypt and MAC operations. It does not apply to GenericStreamCiphers or to GenericAEADCiphers that already include integrity protection with the cipher. If a server receives an encrypt-then-MAC request extension from a client and then selects a stream or Authenticated Encryption with Associated Data (AEAD) ciphersuite, it MUST NOT send an encrypt-then-MAC response extension back to the client

Decryption reverses this processing. The MAC SHALL be evaluated before any further processing such as decryption is performed, and if the MAC verification fails, then processing SHALL terminate immediately. For TLS, a fatal bad_record_mac MUST be generated. For DTLS, the record MUST be discarded, and a fatal bad_record_mac MAY be generated. This immediate response to a bad MAC eliminates any timing channels that may be available through the use of manipulated packet data

Some implementations may prefer to use a truncated MAC rather than a full-length one. In this case, they MAY negotiate the use of a truncated MAC through the TLS truncated_hmac extension as defined in TLS-Ext

Rehandshake Issues

The status of encrypt-then-MAC vs. MAC-then-encrypt can potentially change during one or more rehandshakes. Implementations SHOULD retain the current session state across all rehandshakes for that session. (In other words, if the mechanism for the current session is X, then the renegotiated session should also use X.) Although implementations SHOULD NOT change the state during a rehandshake, if they wish to be more flexible, then the following rules apply:

As the above table points out, implementations MUST NOT renegotiate a downgrade from encrypt-then-MAC to MAC-then-encrypt. Note that a client or server that doesn't wish to implement the mechanism-change during-rehandshake ability can (as a client) not request a mechanism change and (as a server) deny the mechanism change

Note that these rules apply across potentially many rehandshakes. For example, if a session were in the encrypt-then-MAC state and a rehandshake selected a GenericAEADCiphers ciphersuite and a subsequent rehandshake then selected a MAC-then-encrypt ciphersuite, this would be an error since the renegotiation process has resulted in a downgrade from encrypt-then-MAC to MAC-then-encrypt (via the AEAD ciphersuite)

(As the text above has already pointed out, implementations SHOULD avoid having to deal with these ciphersuite calisthenics by retaining the initially negotiated mechanism across all rehandshakes)

If an upgrade from MAC-then-encrypt to encrypt-then-MAC is negotiated as per the second line in the table above, then the change will take place in the first message that follows the Change Cipher Spec (CCS) message. In other words, all messages up to and including the CCS will use MAC-then-encrypt, and then the message that follows will continue with encrypt-then-MAC

Security Considerations

This document defines encrypt-then-MAC, an improved security mechanism to replace the current MAC-then-encrypt one. Encrypt-then MAC is regarded as more secure than the current mechanism and should mitigate or eliminate a number of attacks on the current mechanism, provided that the instructions on MAC processing given in Section 3 are applied.

An active attacker who can emulate a client or server with extension intolerance may cause some implementations to fall back to older protocol versions that don't support extensions, which will in turn force a fallback to non-encrypt-then-MAC behaviour. A straightforward solution to this problem is to avoid fallback to older, less secure protocol versions. If fallback behaviour is unavoidable, then mechanisms to address this issue, which affects all capabilities that are negotiated via TLS extensions, are being developed by the TLS working group. Anyone concerned about this type of attack should consult the TLS working group documents for guidance on appropriate defence mechanisms.

IANA Considerations

IANA has added the extension code point 22 (0x16) for the encrypt_then_mac extension to the TLS "ExtensionType Values" registry as specified in TLS.

[1] <http://orm-chimera-prod.s3.amazonaws.com/1230000000545/ch04.html>

[2] <https://tools.ietf.org/html/rfc7366>

For API Documentation:

See Also

[ProtocolPP::jprotocol](#)
[ProtocolPP::jtlsa](#)

For Additional Documentation:

See Also

[jprotocol](#)
[jtlsa](#)

Protocol++ written by : John Peter Greninger © John Peter Greninger 2015-2019 All Rights Reserved

All copyrights and trademarks are the property of their respective owners

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
- Any and all modifications must be returned to John Peter Greninger at GitHub.com <https://github.com/jpgreninger/protocolpp> for evaluation. Inclusion of modifications in the source code shall be determined solely by John Peter Greninger. Failure to provide modifications shall render this license NULL and VOID and revoke any rights to use of Protocol++.
- Commercial use requires a fee-based license obtainable at www.protocolpp.com
- Academic use requires written and notarized permission from John Peter and Sheila Rocha Greninger
- US Copyrights at <https://www.copyright.gov/>
 - TXu002059872 (Version 1.0.0)
 - TXu002066632 (Version 1.2.7)

- TXu002082674 (Version 1.4.0)
- TXu002097880 (Version 2.0.0)

The names of its contributors may not be used to endorse or promote products derived from this software without specific prior written permission

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE

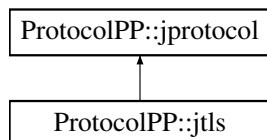
The documentation for this class was generated from the following file:

- [include/jtls.h](#)

7.128 ProtocolPP::jtls Class Reference

```
#include <jtls.h>
```

Inheritance diagram for ProtocolPP::jtls:



Public Member Functions

- [jtls \(std::shared_ptr< jrand > &rand, std::shared_ptr< jtlsa > &security\)](#)
 - [jtls \(std::shared_ptr< jrand > &rand, std::shared_ptr< jtlsa > &security, std::string &file\)](#)
 - virtual [~jtls \(\)](#)
- Standard deconstructor.*
- void [encap_packet \(std::shared_ptr< jarray< uint8_t >> &input, std::shared_ptr< jarray< uint8_t >> &output\)](#)
 - void [decap_packet \(std::shared_ptr< jarray< uint8_t >> &input, std::shared_ptr< jarray< uint8_t >> &output\)](#)
 - void [set_hdr \(jarray< uint8_t > &hdr\)](#)
 - void [set_field \(field_t field, uint64_t value\)](#)
 - [jarray< uint8_t > get_hdr \(\)](#)
 - uint64_t [get_field \(field_t field, jarray< uint8_t > &header\)](#)
 - void [to_xml \(tinyxml2::XMLPrinter &myxml, direction_t direction\)](#)

Static Public Member Functions

- static void [get_prf \(tls_ciphersuite_t ciphersuite, jarray< uint8_t > secret, jarray< uint8_t > label, jarray< uint8_t > seed, std::shared_ptr< jarray< uint8_t >> &output, unsigned int length\)](#)

Additional Inherited Members

7.128.1 Constructor & Destructor Documentation

7.128.1.1 `ProtocolPP::jtls::jtls (std::shared_ptr<jrand> & rand, std::shared_ptr<jtsa> & security)`

Constructor for TLS

Parameters

<i>rand</i>	- Random data generation for IVs and padding
<i>security</i>	- Security association (SA) for this TLS flow

7.128.1.2 `ProtocolPP::jtls::jtls (std::shared_ptr<jrand> & rand, std::shared_ptr<jtsa> & security, std::string & file)`

Constructor for TLS

Parameters

<i>rand</i>	- Random data generation for IVs and padding
<i>security</i>	- Security association (SA) for this TLS flow
<i>file</i>	- file for input/output data

7.128.1.3 `virtual ProtocolPP::jtls::~jtls () [inline], [virtual]`

Standard deconstructor.

7.128.2 Member Function Documentation

7.128.2.1 `void ProtocolPP::jtls::decap_packet (std::shared_ptr<jarray<uint8_t>> & input, std::shared_ptr<jarray<uint8_t>>> & output) [virtual]`

Decap will produce a payload from the packet passed

Parameters

<i>input</i>	- packet to decapsulate with TLS
<i>output</i>	- packet encapsulated with TLS

Implements [ProtocolPP::jprotocol](#).

7.128.2.2 `void ProtocolPP::jtls::encap_packet (std::shared_ptr<jarray<uint8_t>> & input, std::shared_ptr<jarray<uint8_t>>> & output) [virtual]`

Encap will produce a packet from the payload passed

Parameters

<i>input</i>	- payload to protect with TLS
<i>output</i>	- packet encapsulated with TLS

Implements [ProtocolPP::jprotocol](#).

7.128.2.3 `uint64_t ProtocolPP::jtls::get_field (field_t field, jarray<uint8_t> & header) [virtual]`

Retrieve the field from the TLS header

Parameters

<i>field</i>	- field to retrieve
<i>header</i>	- TLS header to retrieve field from

Returns

value of the field

Implements [ProtocolPP::jprotocol](#).

7.128.2.4 `jarray<uint8_t> ProtocolPP::jtls::get_hdr() [virtual]`

Retrieve the TLS header

Returns

- the header

Implements [ProtocolPP::jprotocol](#).

7.128.2.5 `static void ProtocolPP::jtls::get_prf(tls_ciphersuite_t ciphersuite, jarray< uint8_t > secret, jarray< uint8_t > label, jarray< uint8_t > seed, std::shared_ptr< jarray< uint8_t >> & output, unsigned int length) [static]`

Generate PRF material

Parameters

<i>ciphersuite</i>	- Ciphersuite to generate PRF data for
<i>secret</i>	- Key for hash function
<i>label</i>	- Label for the connection
<i>seed</i>	- Seed for the PRF computation
<i>output</i>	- Pointer to hold output data
<i>length</i>	- amount of data to produce

7.128.2.6 `void ProtocolPP::jtls::set_field(field_t field, uint64_t value) [virtual]`

Update the type field in the TLS header

Parameters

<i>field</i>	- TLS field to update
<i>value</i>	- new value for the field

Implements [ProtocolPP::jprotocol](#).

7.128.2.7 `void ProtocolPP::jtls::set_hdr(jarray< uint8_t > & hdr) [virtual]`

Update the type field in the TLS header

Parameters

<i>hdr</i>	- new header
------------	--------------

Implements [ProtocolPP::jprotocol](#).

7.128.2.8 void ProtocolPP::jtls::to_xml (*tinyxml2::XMLPrinter* & *myxml*, *direction_t direction*) [virtual]

Print the protocol and security objects to XML

Parameters

<i>myxml</i>	- XMLPrinter object
<i>direction</i>	- facilitator for random descriptor generation

Implements [ProtocolPP::jprotocol](#).

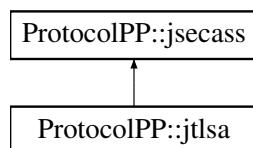
The documentation for this class was generated from the following file:

- [include/jtls.h](#)

7.129 ProtocolPP::jtlsa Class Reference

```
#include <jtlsa.h>
```

Inheritance diagram for ProtocolPP::jtlsa:



Public Member Functions

- [jtlsa \(\)](#)
 - [jtlsa \(direction_t dir, tls_ciphersuite_t ciphersuite, tlsver_t ver, tlstype_t type, uint16_t epoch, uint64_t seqnum, unsigned int icvlen, unsigned int ivlen, std::shared_ptr< jarray< uint8_t >> iv, unsigned int ckeylen, std::shared_ptr< jarray< uint8_t >> cipherkey, unsigned int akeylen, std::shared_ptr< jarray< uint8_t >> authkey, unsigned int saltlen, std::shared_ptr< jarray< uint8_t >> salt, unsigned int arlen, jarray< uint8_t > arwin, unsigned int mtu, bool randiv, bool ivex, bool encthenmac\)](#)
 - [jtlsa \(jtlsa &rhs\)](#)
 - [jtlsa \(std::shared_ptr< jtlsa > &rhs\)](#)
 - virtual [~jtlsa \(\)](#)
- Standard deconstructor.*
- template<typename T>
[void set_field \(field_t field, T fieldval\)](#)
 - template<typename T>
[T get_field \(field_t field\)](#)
 - [void to_xml \(tinyxml2::XMLPrinter &myxml, direction_t direction\)](#)

7.129.1 Constructor & Destructor Documentation

7.129.1.1 ProtocolPP::jtlsa::jtlsa ()

Standard Constructor with defaults

7.129.1.2 ProtocolPP::jtlsa::jtlsa (direction_t dir, tls_ciphersuite_t ciphersuite, tlsver_t ver, tlstype_t type, uint16_t epoch, uint64_t seqnum, unsigned int icvlen, unsigned int ivlen, std::shared_ptr< jarray< uint8_t >> iv, unsigned int ckeylen, std::shared_ptr< jarray< uint8_t >> cipherkey, unsigned int akeylen, std::shared_ptr< jarray< uint8_t >> authkey, unsigned int saltlen, std::shared_ptr< jarray< uint8_t >> salt, unsigned int arlen, jarray< uint8_t > arwin, unsigned int mtu, bool randiv, bool ivex, bool encthenmac)

Security Association for TLS

	field name	Example
e_t	DIRECTION <i>dir</i>	- Direction of processing (ENCAP or DECAP) ProtocolPP::direction_t>(ProtocolPP::field_t::DIRECTION)
	ciphersuite	- Ciphersuite to use with TLS/SSL to use ProtocolPP::field_t::CIPHERSUITE)
	ver	- Version of TLS/SSL to use ProtocolPP::field_t::VERSION)
	CIPHERSUITE <i>icvlen</i>	- Length of the ICV tag ProtocolPP::tls_ciphersuite_t myciphersuite = get_field<ProtocolPP::tls_ciphersuite_t>(ProtocolPP::field_t::CIPHERSUITE)
	mtu	- Maximum transmission unit ProtocolPP::field_t::MTU)
	type	- Default type of packet to send (usually Application) ProtocolPP::field_t::TYPE)
	VERSION <i>epoch</i>	- Initial Epoch for DTLS ProtocolPP::tlsvver_t myver = get_field<ProtocolPP::tlsvver_t>(ProtocolPP::field_t::VERSION)
	seqnum	- Initial sequence number ProtocolPP::field_t::SEQNUM)
	ivlen	- Length of the initialization vector (IV) ProtocolPP::field_t::IV)
	TYPE <i>iv</i>	- Initialization Vector (IV) ProtocolPP::tlstype_t mytype = get_field<ProtocolPP::tlstype_t>(ProtocolPP::field_t::TYPE)
	RANDIV <i>keylen</i>	- Length of the authentication key ProtocolPP::field_t::RANDIV)
	authkey	- Key for the authentication algorithm ProtocolPP::field_t::AUTHKEY)
	IVEX <i>saltlen</i>	- Length of the salt ProtocolPP::field_t::IVEX)
	salt	- salt for CTR, CCM, GCM and ChaCha20 ciphers ProtocolPP::field_t::SALT)
	ENCTHENMAC	Number og packets to track in the replay window ProtocolPP::field_t::ENCTHENMAC)
	arwin	- Anti-replay window for tracking packets ProtocolPP::field_t::ARWIN)
	randiv	- use random IV instead of ENCTHENMAC);
	EPOCH <i>ivex</i>	- IV is sent in the clear instead of encrypted if asserted ProtocolPP::field_t::EPOCH)
	encthenmac	- Encrypt then MAC flag ProtocolPP::field_t::ENCTHENMAC)
	ICVLEN	ProtocolPP::field_t::ICVLEN);
7.129.1.3	ProtocolPP::jtlsa::jtlsa (jtlsa & rhs)	uint32_t myicvlen = get_field<uint32_t>(ProtocolPP- ::field_t::ICVLEN);
	IVLEN	ProtocolPP::field_t::IVLEN);
	Constructor for TLS Security Association Parameters	uint32_t myivlen = get_field<uint32_t>(ProtocolPP- ::field_t::IVLEN);
	CKEYLEN <i>rhs</i>	uint32_t myckeylen = get_field<uint32_t>(ProtocolPP- ::field_t::CKEYLEN);
	AKEYLEN	ProtocolPP::field_t::AKEYLEN);
7.129.1.4	ProtocolPP::jtlsa::jtlsa (std::shared_ptr< jtlsa & rhs)	uint32_t myakeylen = get_field<uint32_t>(ProtocolPP- ::field_t::AKEYLEN);
	SALTLEN	ProtocolPP::field_t::SALTLEN);
	Constructor for TLS Security Association Parameters	uint32_t mysaltlen = get_field<uint32_t>(ProtocolPP- ::field_t::SALTLEN);
	ARLEN <i>rhs</i>	ProtocolPP::field_t::ARLEN);
	MTU	ProtocolPP::field_t::MTU);
7.129.1.5	virtual ProtocolPP::jtlsa::~jtlsa ()	ProtocolPP::field_t::MTU);
	Standard Constructor.	ProtocolPP::field_t::SEQNUM);
t>	7.129.2 ARWin Number Function Documentation	ProtocolPP::jarray<uint8_t> myarwin = get_field<ProtocolPP- ::field_t::ARWIN)
	7.129.2.1 template<typename T > T ProtocolPP::jtarray< uint8_t > (std::shared_ptr< ProtocolPP::field_t::ARWIN >)	ProtocolPP::jarray<uint8_t> myarwin = get_field<ProtocolPP- ::field_t::ARWIN)
tr<ProtocolPP- _t>>	Retrieves the field from the TLS security association Parameters	ProtocolPP::jarray<ProtocolPP- ::shared_ptr<ProtocolPP- ::array<uint8_t>> myarwin = get_field<std::shared_- ptr<ProtocolPP::jarray<uint8_- t>>>(ProtocolPP::field_t::IV)
	CIPHERKEY	ProtocolPP::jarray<ProtocolPP- ::shared_ptr<ProtocolPP- ::array<uint8_t>> myarwin = get_field<std::shared_-
	Copyright 2017-2019 John Peter Greninger Generated on Sat Aug 31 2019 02:28:11 for Protocol++ (ProtocolPP) 3.0.1 by Doxygen	ProtocolPP::jarray<uint8_t> myarwin = get_field<ProtocolPP- ::field_t::CIPHE- RKEY)
	AUTHKEY	ProtocolPP::jarray<uint8_t> myarwin = get_field<ProtocolPP- ::field_t::AUTHKEY)

<i>field</i>	- field to retrieve
--------------	---------------------

Returns

value of the field

7.129.2.2 template<typename T > void ProtocolPP::jtlsa::set_field (*field_t field, T fieldval*)

Update the field in the TLS security association

Parameters

<i>field</i>	- TLS field to update
<i>fieldval</i>	- new value for the field

7.129.2.3 void ProtocolPP::jtlsa::to_xml (*tinyxml2::XMLPrinter & myxml, direction_t direction*) [virtual]

Print the protocol and security objects to XML

Parameters

<i>myxml</i>	- XMLPrinter object
<i>direction</i>	- facilitator for random descriptor generation

Implements [ProtocolPP::jsecass](#).

The documentation for this class was generated from the following file:

- [include/jtlsa.h](#)

7.130 jtlsa Class Reference

```
#include "include/jtlsa.h"
```

7.130.1 Detailed Description

7.130.2 Transport Layer Security Association (TLSA)

For additional information see http://en.wikipedia.org/wiki/Transport_Layer_Security

The TLS protocol exchanges records—which encapsulate the data to be exchanged in a specific format (see below). Each record can be compressed, padded, appended with a message authentication code (MAC), or encrypted, all depending on the state of the connection. Each record has a content type field that designates the type of data encapsulated, a length field and a TLS version field. The data encapsulated may be control or procedural messages of the TLS itself, or simply the application data needed to be transferred by TLS. The specifications (cipher suite, keys etc.) required to exchange application data by TLS, are agreed upon in the "TLS handshake" between the client requesting the data and the server responding to requests. The protocol therefore defines both the structure of payloads transferred in TLS and the procedure to establish and monitor the transfer

TLS Records

This is the general format for all TLS records (packets) [1]

Byte0	Byte1	Byte2	Byte3		+3
0	Content Type				
0x16	1..4	Version		Length	
	5..n		Payload		
	n..n	Version	Length	MAC	
Message Type	m..p			Padding (block ciphers only)	
				Handshake Message Data Length Figure 7.113: General Format for TLS Records (Packets)	
				Handshake Message Data	

Content Type

This field identifies the Record Layer Protocol Type contained in this Record

Version

This field identifies the major and minor version of TLS for the contained message. For a ClientHello message, this need [Table 7.6: TLS Handshake supported by the client](#)

Byte0	Byte1	Byte2	Byte3		
Length	XXXXXXXXXXXXXXXXXXXXXXXXXXXXXX-				
0x15	The length of Protocol Message(s) , MAC and Padding, not to exceed 2^{14} bytes (16KB)				
Protocol Message(s)					
One or more messages identified by the Protocol field. Note that this field may be encrypted depending on the state of the connection	Version	Length			
MAC and Padding	XXXXXXXXXXXXXXXXXXXXXXXXXXXXXX-				
A message authentication code computed over the Protocol message, with additional key material included Note Level	Description				
that this field may be encrypted, or not included entirely, depending on the state of the connection No MAC or Padding can be present at end of TLS records before all cipher algorithms and parameters have been negotiated and handshaked and then confirmed by sending a CipherStateChange record (see below) for signaling that these parameters will take effect in all further records sent by the same peer	MAC (optional)				
Handshake protocol					
Most messages exchanged during the setup of the TLS session are based on this record, unless an error or warning occurs and needs to be signalled by an Alert protocol record (see below), or the encryption mode of the session is modified by another record (see ChangeCipherSpec protocol below)	Padding (block ciphers only)				

Message Type 7.82: TLS Alert Protocol

This field identifies the handshake message type

Handshake message data length

This is a 3-byte field indicating the length of the handshake data, not including the header Note that multiple handshake messages may be combined within one record

Alert protocol

This record should normally not be sent during normal handshaking or application exchanges. However, this message can be sent at any time during the handshake and up to the closure of the session. If this is used to signal a fatal error, the session will be closed immediately after sending this record, so this record is used to give a reason for this closure. If the alert level is flagged as a warning, the remote can decide to close the session if it decides that the session is not reliable enough for its needs (before doing so, the remote may also send its own signal)

Level

This field identifies the level of alert. If the level is fatal, the sender should close the session immediately. Otherwise, the recipient may decide to terminate the session itself, by sending its own fatal alert and closing the session itself immediately after sending it. The use of Alert records is optional, however if it is missing before the session closure,

the session may be resumed automatically (with its handshakes)

Normal closure of a session after termination of the transported application should preferably be alerted with at least the Close notify Alert type (with a simple warning level) to prevent such automatic resume of a new session. Signaling explicitly the normal closure of a secure session before effectively closing its transport layer is useful to prevent or detect attacks (like attempts to truncate the securely transported data, if it intrinsically does not have a predetermined length or duration that the recipient of the secured data may expect)

ChangeCipherSpec protocol

CCS protocol type

Currently only 1

Application protocol

Length

Length of the application data (excluding the protocol header and including the MAC and padding trailers)

MAC

20 bytes for the SHA1 based HMAC, 16 bytes for the MD5 based HMAC

Padding

Variable length; last byte contains the padding length

[1] <http://orm-chimera-prod.s3.amazonaws.com/123000000545/ch04.html>

[2] <https://tools.ietf.org/html/rfc7366>

For API Documentation:

See Also

[ProtocolPP::jprotocol](#)
[ProtocolPP::jtls](#)

For Additional Documentation:

See Also

[jprotocol](#)
[jtls](#)

Protocol++ written by : John Peter Greninger © John Peter Greninger 2015-2019 All Rights Reserved

All copyrights and trademarks are the property of their respective owners

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer
- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution
- Any and all modifications must be returned to John Peter Greninger at GitHub.com <https://github.com/jpgreninger/protocolpp> for evaluation. Inclusion of modifications in the source code shall be determined solely by John Peter Greninger. Failure to provide modifications shall render this license NULL and VOID and revoke any rights to use of Protocol++
- Commercial use requires a fee-based license obtainable at www.protocolpp.com
- Academic use requires written and notarized permission from John Peter and Sheila Rocha Greninger
- US Copyrights at <https://www.copyright.gov/>
 - TXu002059872 (Version 1.0.0)
 - TXu002066632 (Version 1.2.7)
 - TXu002097880 (Version 2.0.0)

+	Byte 0
	Byte 1-4
	Byte 5-(m-1)
	Byte m-(p-1)
	Byte p-(q-1)

Parameters

<i>security</i>	- Parameters necessary to setup UDP flow
-----------------	--

7.132.1.2 ProtocolPP::judp::judp (std::shared_ptr<judpsa> & *security*, std::string & *file*)

constructor for normal UDP with file input

Parameters

<i>security</i>	- Parameters necessary to setup UDP flow
<i>file</i>	- path to file to use for payload data

7.132.1.3 virtual ProtocolPP::judp::~judp () [inline], [virtual]

Standard deconstructor.

7.132.2 Member Function Documentation**7.132.2.1 void ProtocolPP::judp::decap_packet (std::shared_ptr<jarray<uint8_t>> & *input*) [virtual]**

This function is for use with the constructor that accepts a file Decap will extract the data and write it to the file specified in the constructor

Parameters

<i>input</i>	- UDP packet to decapsulate and write to file
--------------	---

Reimplemented from [ProtocolPP::jprotocol](#).

7.132.2.2 void ProtocolPP::judp::decap_packet (std::shared_ptr<jarray<uint8_t>> & *input*, std::shared_ptr<jarray<uint8_t>> & *output*) [virtual]

This function is for use with the constructor without a file handle. Decap will produce a payload from the packet passed in

Parameters

<i>input</i>	- UDP packet to decapsulate
<i>output</i>	- decapsulated payload

Implements [ProtocolPP::jprotocol](#).

7.132.2.3 void ProtocolPP::judp::encap_packet (std::shared_ptr<jarray<uint8_t>> & *output*) [virtual]

This function is for use with the constructor that accepts a file. Encap will produce a packet each time it's called from the data found in the file until all data is exhausted

Parameters

<i>output</i>	- packet generated from payload data extracted from the file
---------------	--

Reimplemented from [ProtocolPP::jprotocol](#).

```
7.132.2.4 void ProtocolPP::judp::encap_packet( std::shared_ptr<jarray< uint8_t >> & input, std::shared_ptr<jarray< uint8_t >> & output ) [virtual]
```

This function is for use with the constructor without a file handle. Encap will produce a packet from the payload passed in

Parameters

<i>input</i>	- payload to encapsulate with UDP
<i>output</i>	- encapsulated UDP packet

Implements [ProtocolIPP::jprotocol](#).

7.132.2.5 uint64_t ProtocolIPP::judp::get_field (field_t field) [virtual]

Retrieves the field from the security association

Returns

source address from the UDP header

Reimplemented from [ProtocolIPP::jprotocol](#).

7.132.2.6 uint64_t ProtocolIPP::judp::get_field (field_t field, jarray< uint8_t > & hdr) [virtual]

Retrieves the field from the UDP header

Returns

source address from the UDP header

Implements [ProtocolIPP::jprotocol](#).

7.132.2.7 jarray<uint8_t> ProtocolIPP::judp::get_hdr () [virtual]

Retrieves the complete UDP header

Returns

the UDP header

Implements [ProtocolIPP::jprotocol](#).

7.132.2.8 void ProtocolIPP::judp::set_field (field_t field, uint64_t value) [virtual]

Allows the user to update the source address in the UDP header

Parameters

<i>field</i>	- field to update in UDP
<i>value</i>	- new value

Implements [ProtocolIPP::jprotocol](#).

7.132.2.9 void ProtocolIPP::judp::set_hdr (jarray< uint8_t > & hdr) [virtual]

Allows the user to update the complete UDP header

Parameters

<i>hdr</i>	- new UDP header
------------	------------------

Implements [ProtocolPP::jprotocol](#).

field
dir
src
dst
mtu

7.132.2.10 void [ProtocolPP::judp::to_xml](#) ([tinyxml2::XMLPrinter](#) & *myxml*, [direction_t](#) *direction*) [virtual]

Print the protocol object to XML

Parameters

<i>myxml</i>	- XMLPrinter object
<i>direction</i>	- facilitator for random generation

Implements [ProtocolPP::jprotocol](#).

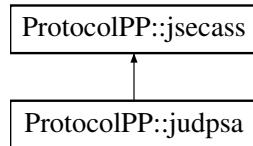
The documentation for this class was generated from the following file:

- [include/judp.h](#)

7.133 ProtocolPP::judpsa Class Reference

```
#include <judpsa.h>
```

Inheritance diagram for ProtocolPP::judpsa:



Public Member Functions

- [judpsa \(\)](#)
Standard copy constructor.
- [judpsa \(\[direction_t\]\(#\) dir, \[uint16_t\]\(#\) src, \[uint16_t\]\(#\) dst, \[unsigned int\]\(#\) mtu\)](#)
- [judpsa \(\[judpsa\]\(#\) &rhs\)](#)
Copy constructor from shared pointer.
- [virtual ~judpsa \(\)](#)
Standard deconstructor.
- template<typename T >
 [void set_field \(\[field_t\]\(#\) field, T fieldval\)](#)
- template<typename T >
 [T get_field \(\[field_t\]\(#\) field\)](#)
- [void to_xml \(\[tinyxml2::XMLPrinter\]\(#\) &myxml, \[direction_t\]\(#\) direction\)](#)

7.133.1 Constructor & Destructor Documentation

7.133.1.1 [ProtocolPP::judpsa::judpsa \(\)](#)

Standard constructor using default values

7.133.1.2 ProtocolPP::judpsa::judpsa (*direction_t dir*, *uint16_t src*, *uint16_t dst*, *unsigned int mtu*)

Parameters necessary to setup a UDP flow

Parameters

		field type
<i>dir</i>	- Direction of UDP data flow (ENCAP or DECAP)	direction_t
<i>src</i>	- Source port for data	
<i>dst</i>	- Destination port for data	
<i>mtu</i>	- Maximum transmission size	uint16_t

7.133.1.3 ProtocolPP::judpsa::judpsa (*judpsa & rhs*)

Standard copy constructor.

7.133.1.4 ProtocolPP::judpsa::judpsa (std::shared_ptr<*judpsa* > & *rhs*)

Copy constructor from shared pointer.

7.133.1.5 virtual ProtocolPP::judpsa::~judpsa () [inline], [virtual]

Standard deconstructor.

7.133.2 Member Function Documentation

7.133.2.1 template<typename T> T ProtocolPP::judpsa::get_field (*field_t field*)

Retrieves a boolean value from the field requested. If the function is called with a field that is not a boolean, this function prints an error to the screen

Due to their dynamic nature, some fields are only available in judp which include the following fields

- LENGTH
- CHECKSUM

Parameters

<i>field</i>	- field to set
--------------	----------------

Returns

- boolean value from the field

7.133.2.2 template<typename T> void ProtocolPP::judpsa::set_field (*field_t field, T fieldval*)

Sets a boolean field in the security association. If the function is called with a field that is not boolean, this function prints an error to the screen

Due to their dynamic nature, some fields are only available in judp which include the following fields

- LENGTH
- CHECKSUM

Parameters

<i>field</i>	- field to set
<i>fieldval</i>	- boolean value for the field

7.133.2.3 void ProtocolPP::judpsa::to_xml (*tinyxml2::XMLPrinter & myxml*, *direction_t direction*) [virtual]

print the protocol and security objects as XML

Parameters

<i>myxml</i>	- object to print to
<i>direction</i>	- randomization

Implements [ProtocolPP::jsecass](#).

The documentation for this class was generated from the following file:

- [include/judpsa.h](#)

7.134 judpsa Class Reference

```
#include "include/judpsa.h"
```

7.134.1 Detailed Description

7.134.2 User Datagram Protocol Security Association (UDPSA)

See https://en.wikipedia.org/wiki/User_Datagram_Protocol

This User Datagram Protocol (UDP) is defined to make available a datagram mode of packet-switched computer communication in the environment of an interconnected set of computer networks. This protocol assumes that the Internet Protocol (IP) [1] is used as the underlying protocol. This protocol provides a procedure for application programs to send messages to other programs with a minimum of protocol mechanism. The protocol is transaction oriented, and delivery and duplicate protection are not guaranteed. Applications requiring ordered reliable delivery of streams of data should use the Transmission Control Protocol (TCP) [2]

User Datagram Header Format [1]

UDP Datagram Header Format								
Bit #	0	7	8	15	16	23	24	31
0								Destination Port
32								Header and Data Checksum

Figure 7.117: User Datagram Protocol (UDP) Header Format

Source Port

An optional field, when meaningful, it indicates the port of the sending process and may be assumed to be the port to which a reply should be addressed in the absence of any other information. If not used, a value of zero is inserted.

Destination Port

Has a meaning within the context of a particular internet destination address

Length

The length in octets of this user datagram including this header and the data (This means the minimum value of the length is eight)

For API Documentation:

See Also

[ProtocolPP::jprotocol](#)
[ProtocolPP::judp](#)
[ProtocolPP::jsecasss](#)

For Additional Documentation:

See Also

[jprotocol](#)
[judp](#)
[jsecasss](#)

Protocol++ written by : John Peter Greninger © John Peter Greninger 2015-2019 All Rights Reserved

All copyrights and trademarks are the property of their respective owners

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer
- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution
- Any and all modifications must be returned to John Peter Greninger at GitHub.com <https://github.com/jpgreninger/protocolpp> for evaluation. Inclusion of modifications in the source code shall be determined solely by John Peter Greninger. Failure to provide modifications shall render this license NULL and VOID and revoke any rights to use of Protocol++
- Commercial use requires a fee-based license obtainable at www.protocolpp.com
- Academic use requires written and notarized permission from John Peter and Sheila Rocha Greninger
- US Copyrights at <https://www.copyright.gov/>
 - TXU002059872 (Version 1.0.0)
 - TXU002066632 (Version 1.2.7)
 - TXU002082674 (Version 1.4.0)
 - TXU002097880 (Version 2.0.0)

The names of its contributors may not be used to endorse or promote products derived from this software without specific prior written permission

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE

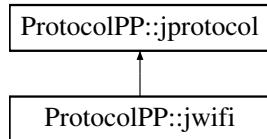
The documentation for this class was generated from the following file:

- include/judpsa.h

7.135 ProtocolPP::jwifi Class Reference

```
#include <jwifi.h>
```

Inheritance diagram for ProtocolPP::jwifi:



Public Member Functions

- `jwifi (std::shared_ptr< jwifisa > &security)`
- virtual `~jwifi ()`
Standard deconstructor.
- void `encap_packet (std::shared_ptr< jarray< uint8_t >> &input, std::shared_ptr< jarray< uint8_t >> &output)`
- void `decap_packet (std::shared_ptr< jarray< uint8_t >> &input, std::shared_ptr< jarray< uint8_t >> &output)`
- void `set_hdr (jarray< uint8_t > &hdr)`
- void `set_field (field_t field, uint64_t value)`
- `jarray< uint8_t > get_hdr ()`
- `uint64_t get_field (field_t field)`
- `uint64_t get_field (field_t field, jarray< uint8_t > &header)`
- void `to_xml (tinyxml2::XMLPrinter &myxml, direction_t direction)`

Static Public Member Functions

- static void `pbkdf2 (auth_t auth, std::shared_ptr< jarray< uint8_t >> &passphrase, std::shared_ptr< jarray< uint8_t >> &ssid, std::shared_ptr< jarray< uint8_t >> &output, unsigned int outlen, unsigned int iterations)`
- static void `prf (auth_t auth, std::shared_ptr< jarray< uint8_t >> &key, std::shared_ptr< jarray< uint8_t >> &label, std::shared_ptr< jarray< uint8_t >> &context, std::shared_ptr< jarray< uint8_t >> &prfout, unsigned int outlen)`
- static void `kdf (auth_t auth, std::shared_ptr< jarray< uint8_t >> &key, std::shared_ptr< jarray< uint8_t >> &label, std::shared_ptr< jarray< uint8_t >> &context, std::shared_ptr< jarray< uint8_t >> &devkey, unsigned int outlen)`

Additional Inherited Members

7.135.1 Constructor & Destructor Documentation

7.135.1.1 `ProtocolPP::jwifi::jwifi (std::shared_ptr< jwifisa > & security)`

Constructor for Wifi

Parameters

<i>security</i>	- Security association (SA) for this Wifi flow
-----------------	--

7.135.1.2 virtual ProtocolPP::jwifi::~jwifi() [inline], [virtual]

Standard deconstructor.

7.135.2 Member Function Documentation

7.135.2.1 void ProtocolPP::jwifi::decap_packet(std::shared_ptr<jarray<uint8_t>> &*input*, std::shared_ptr<jarray<uint8_t>> &*output*) [virtual]

Decap will produce a payload from the packet passed

Parameters

<i>input</i>	- packet to decapsulate with WEP/WPA
<i>output</i>	- packet encapsulated with WEP/WPA

Implements [ProtocolPP::jprotocol](#).

7.135.2.2 void ProtocolPP::jwifi::encap_packet(std::shared_ptr<jarray<uint8_t>> &*input*, std::shared_ptr<jarray<uint8_t>> &*output*) [virtual]

Encap will produce a packet from the payload passed

Parameters

<i>input</i>	- payload to protect with WEP/WPA
<i>output</i>	- packet encapsulated with WEP/WPA

Implements [ProtocolPP::jprotocol](#).

7.135.2.3 uint64_t ProtocolPP::jwifi::get_field(field_t *field*) [virtual]

Retrieve the field from the Wifi security association

Parameters

<i>field</i>	- field to retrieve
--------------	---------------------

Returns

field

Reimplemented from [ProtocolPP::jprotocol](#).

7.135.2.4 uint64_t ProtocolPP::jwifi::get_field(field_t *field*, jarray<uint8_t> &*header*) [virtual]

Retrieve the field from the Wifi header

Parameters

<i>field</i>	- field to retrieve
<i>header</i>	- header to extract the field from

Returns

field

Implements [ProtocolPP::jprotocol](#).**7.135.2.5 jarray<uint8_t> ProtocolPP::jwifi::get_hdr() [virtual]**

Retrieve the Wifi header

Returns

current Wifi header

Implements [ProtocolPP::jprotocol](#).**7.135.2.6 static void ProtocolPP::jwifi::kdf(auth_t auth, std::shared_ptr<jarray< uint8_t >> &key, std::shared_ptr<jarray< uint8_t >> &label, std::shared_ptr<jarray< uint8_t >> &context, std::shared_ptr<jarray< uint8_t >> &devkey, unsigned int outlen) [static]**

Key derivation function (KDF) for Wifi

Parameters

<i>auth</i>	- authentication algorithm (auth_t::SHA1, auth_t::SHA256, auth_t::SHA384, auth_t::SHA512)
<i>key</i>	- key for hash function
<i>label</i>	- string identifying purpose of keys derived using KDF
<i>context</i>	- bit string to provide context to identify the derived key
<i>devkey</i>	- the calculated derived key
<i>outlen</i>	- length of the output derived key (in bits)

7.135.2.7 static void ProtocolPP::jwifi::pbkdf2(auth_t auth, std::shared_ptr<jarray< uint8_t >> &passphrase, std::shared_ptr<jarray< uint8_t >> &ssid, std::shared_ptr<jarray< uint8_t >> &output, unsigned int outlen, unsigned int iterations) [static]

PBKDF2 function used to derive the Pairwise Master Key (PMK)

Parameters

<i>auth</i>	- Authentication function to use (SHA1, SHA256, SHA384, SHA512)
<i>passphrase</i>	- Passphrase from the user (PSK or EAP generated)
<i>ssid</i>	- SSID of this Wifi connection
<i>output</i>	- Pointer to hold output PMK
<i>outlen</i>	- Length of the requested output. Length must be equal to or shorter than output of the HASH function
<i>iterations</i>	- Number of times to process data to derive PMK

7.135.2.8 static void ProtocolPP::jwifi::prf(auth_t auth, std::shared_ptr<jarray< uint8_t >> &key, std::shared_ptr<jarray< uint8_t >> &label, std::shared_ptr<jarray< uint8_t >> &context, std::shared_ptr<jarray< uint8_t >> &prfout, unsigned int outlen) [static]

Pseudo random function (PRF) for Wifi

Parameters

<i>auth</i>	- authentication algorithm (auth_t::SHA1, auth_t::SHA256, auth_t::SHA384, auth_t::SHA512)
<i>key</i>	- key for hash function
<i>label</i>	- string identifying purpose of keys derived using KDF
<i>context</i>	- bit string to provide context to identify the derived key
<i>prfout</i>	- the PRF output
<i>outlen</i>	- length of the output derived key (in bits)

7.135.2.9 void ProtocolPP::jwifi::set_field (*field_t field*, *uint64_t value*) [virtual]

Update Wifi field with the new value

Parameters

<i>field</i>	- field to update
<i>value</i>	- new value for the field

Implements [ProtocolPP::jprotocol](#).

7.135.2.10 void ProtocolPP::jwifi::set_hdr (*jarray< uint8_t > & hdr*) [virtual]

Update the current Wifi header with a new header

Parameters

<i>hdr</i>	- new Wifi header for this flow
------------	---------------------------------

Implements [ProtocolPP::jprotocol](#).

7.135.2.11 void ProtocolPP::jwifi::to_xml (*tinyxml2::XMLPrinter & myxml*, *direction_t direction*) [virtual]

Print the protocol and security objects to XML

Parameters

<i>myxml</i>	- XMLPrinter object to print with
<i>direction</i>	- facilitator for random generation

Implements [ProtocolPP::jprotocol](#).

The documentation for this class was generated from the following file:

- [include/jwifi.h](#)

7.136 jwifi Class Reference

```
#include "include/jwifi.h"
```

7.136.1 Detailed Description

7.136.2 Wifi/WiGig Protocol

Pairwise Master Key (PMK) derivation

The pairwise master key (PMK) is derived from the group master key (GMK), and then used to create the pairwise transient key (PTK), the EAPOL key confirmation key (KCK), and the EAPOL key encryption key (KEK). There are

two ways to obtain a PMK, either by use of a preshared key (PSK) or through the use of Extensible Authentication Protocol (EAP). When using the PSK, the PMK is derived by using the PBKDF2 algorithm to obtain the PMK from the passphrase and SSID as follows:

```
HMACSHAX(PSK,SSID,Iterations)
for i  $\leftarrow$  0 do
    output  $\leftarrow$  HMACSHAX(PSK,SSID||0x00000001)
    input  $\leftarrow$  output
for i  $\leftarrow$  1 to Iterations do
    output  $\leftarrow$  HMACSHAX(PSK,input)
    input  $\leftarrow$  output
return L(output,0,Len)
```

Pseudo random function (PRF)

A PRF is used in a number of places in this standard. Depending on its use, it may need to output 128 bits, 192 bits, 256 bits, 384 bits, or 512 bits. This subclause defines five functions:

- PRF-128, which outputs 128 bits
- PRF-192, which outputs 192 bits
- PRF-256, which outputs 256 bits
- PRF-384, which outputs 384 bits
- PRF-512, which outputs 512 bits

In the following, K is a key; A is a unique label for each different purpose of the PRF; B is a variable-length string; Y is a single octet containing 0; X is a single octet containing the loop parameter i; and || denotes concatenation:

```
HMACSHA1(K,A,B,X)  $\leftarrow$  HMACSHA1(K,A||Y||B||X)
PRF(K,A,B,Len)
for i  $\leftarrow$  0 to (Len + 159)/160 do
    R  $\leftarrow$  R||HSHA1(K,A,B,i)
return L(R, 0, Len)
```

- PRF-128(K, A, B) = PRF(K, A, B, 128)
- PRF-192(K, A, B) = PRF(K, A, B, 192)
- PRF-256(K, A, B) = PRF(K, A, B, 256)
- PRF-384(K, A, B) = PRF(K, A, B, 384)
- PRF-512(K, A, B) = PRF(K, A, B, 512)

When the negotiated AKM is 00-0F-AC:5 or 00-0F-AC:6, the KDF specified in 11.6.1.7.2 shall be used instead of the PRF construction defined here. In this case, A is used as the KDF label and B as the KDF Context and the PRF functions are defined as follows:

- PRF-128(K, A, B) = KDF-128(K, A, B)
- PRF-192(K, A, B) = KDF-192(K, A, B)
- PRF-256(K, A, B) = KDF-256(K, A, B)
- PRF-384(K, A, B) = KDF-384(K, A, B)

- PRF-512(K, A, B) = KDF-512(K, A, B)

Key derivation function (KDF)

The KDF for the FT key hierarchy is a variant of the pseudorandom function (PRF) defined in 11.6.1.2 and is defined as follows:

Output \leftarrow *KDFLength*(*K*, *label*, *Context*) where

- *K*, a 256-bit key derivation key
- *label*, a string identifying the purpose of the keys derived using this KDF
- *Context*, a bit string that provides context to identify the derived key
- Length, the length of the derived key in bits

Output: a Length-bit derived key

```

result ← ""
iterations ←  $\frac{\text{Length}}{\text{Hashlen}}$ 
do i = 1 to iterations
    result ← result || HMACSHA256(K, i || label || Context || Length)
od
return first Length bits of result, and securely delete all unused bits

```

In this algorithm, *i* and *Length* are encoded as 16-bit unsigned integers, represented using the bit ordering conventions of 8.2.2. *K*, *label*, and *Context* are bit strings and are represented using the ordering conventions of 8.2.2. Hashlen is the output length of the hash in bits

CTR with CBC-MAC Protocol (CCMP)

Subclause 11.4.3 specifies the CCMP, which provides data confidentiality, authentication, integrity, and replay protection. CCMP is mandatory for RSN compliance. CCMP is based on the CCM of the AES encryption algorithm. CCM combines CTR for data confidentiality and CBC-MAC for authentication and integrity. CCM protects the integrity of both the MPDU Data field and selected portions of the IEEE 802.11 MPDU header.

The AES algorithm is defined in FIPS PUB 197-2001. All AES processing used within CCMP uses AES with a 128-bit key and a 128-bit block size. CCM is defined in IETF RFC 3610. CCM is a generic mode that can be used with any block-oriented encryption algorithm. CCM has two parameters (*M* and *L*), and CCMP uses the following values for the CCM parameters:

- *M* = 8; indicating that the MIC is 8 octets
- *L* = 2; indicating that the Length field is 2 octets, which is sufficient to hold the length of the largest possible IEEE 802.11 MPDU, expressed in octets

CCM requires a fresh temporal key for every session. CCM also requires a unique nonce value for each frame protected by a given temporal key, and CCMP uses a 48-bit packet number (PN) for this purpose. Reuse of a PN with the same temporal key voids all security guarantees.

When CCMP is selected as the RSN pairwise cipher and management frame protection is negotiated, individually addressed robust management frames and the group addressed management frames that receive "Group Addressed Privacy" as indicated in Table8-38 shall be protected with CCMP.

CCMP MPDU format

Figure11-16 depicts the MPDU when using CCMP

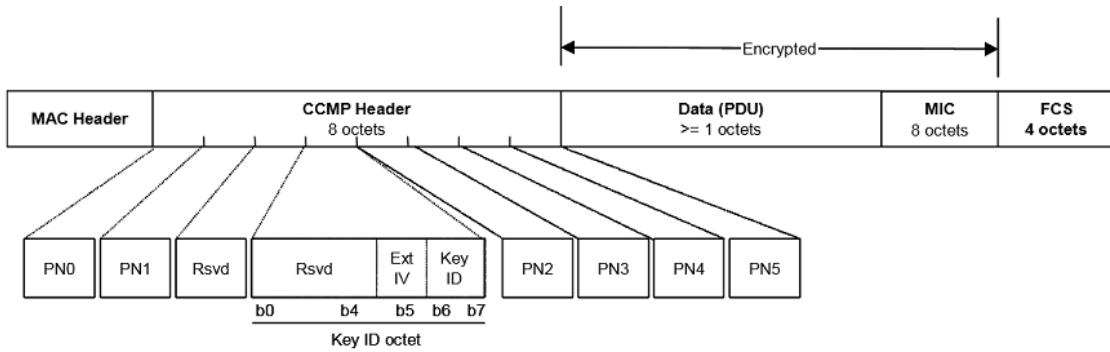


Figure 7.118: Wifi Encapsulation with CCMP Encoding

CCMP processing expands the original MPDU size by 16 octets, 8 octets for the CCMP Header field and 8 octets for the MIC field. The CCMP Header field is constructed from the PN, ExtIV, and Key ID subfields. PN is a 48-bit PN represented as an array of 6 octets. PN5 is the most significant octet of the PN, and PN0 is the least significant. Note that CCMP does not use the WEP ICV.

The ExtIV subfield (bit 5) of the Key ID octet signals that the CCMP Header field extends the MPDU header by a total of 8 octets, compared to the 4 octets added to the MPDU header when WEP is used. The ExtIV bit (bit5) is always set to 1 for CCMP.

Bits 6–7 of the Key ID octet are for the Key ID subfield.

The reserved bits shall be set to 0 and shall be ignored on reception.

CCMP cryptographic encapsulation

The CCMP cryptographic encapsulation process is depicted in Figure 11-17.

CCMP encrypts the payload of a plaintext MPDU and encapsulates the resulting cipher text using the following steps:

- Increment the PN, to obtain a fresh PN for each MPDU, so that the PN never repeats for the same temporal key. Note that retransmitted MPDUs are not modified on retransmission
- Use the fields in the MPDU header to construct the additional authentication data (AAD) for CCM. The CCM algorithm provides integrity protection for the fields included in the AAD. MPDU header fields that may change when retransmitted are muted by being masked to 0 when calculating the AAD
- Construct the CCM Nonce block from the PN, A2, and the Priority field of the MPDU where A2 is MPDU Address 2
- Place the new PN and the key identifier into the 8-octet CCMP header
- Use the temporal key, AAD, nonce, and MPDU data to form the cipher text and MIC. This step is known as CCM originator processing
- Form the encrypted MPDU by combining the original MPDU header, the CCMP header, the encrypted data and MIC, as described in 11.4.3.2

The CCM reference describes the processing of the key, nonce, AAD, and data to produce the encrypted output. See 11.4.3.3.2 to 11.4.3.3.6 for details of the creation of the AAD and nonce from the MPDU and the associated MPDU-specific processing.

PN processing

The PN is incremented by a positive number for each MPDU. The PN shall never repeat for a series of encrypted MPDUs using the same temporal key.

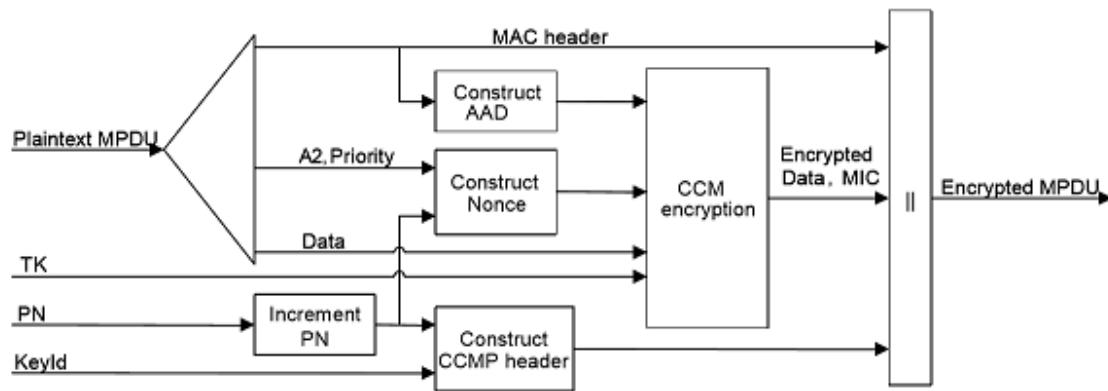
**Figure 11-17—CCMP encapsulation block diagram**

Figure 7.119: CCMP Encapsulation for Wifi Packets

Construct AAD

The format of the AAD is shown in Figure 11-18

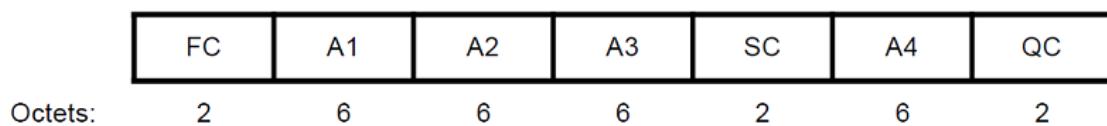
**Figure 11-18—AAD construction**

Figure 7.120: CCMP Authentication Algorithm Data (AAD) Formatting

The length of the AAD varies depending on the presence or absence of the QC and A4 fields and is shown in Table 11-1

Table 11-1—AAD length

QC field	A4 field	AAD length (octets)
Absent	Absent	22
Present	Absent	24
Absent	Present	28
Present	Present	30

Figure 7.121: CCMP Length Calculation for Authentication Algorithm Data (AAD)

The AAD is constructed from the MPDU header. The AAD does not include the header Duration field, because the Duration field value might change due to normal IEEE 802.11 operation (e.g., a rate change during retransmission). The AAD includes neither the Duration/ID field nor the HT Control field because the contents of these fields might change during normal operation (e.g., due to a rate change preceding retransmission). The HT Control field might also be inserted or removed during normal operation (e.g., retransmission of an A-MPDU where the original A-MPDU included an MRQ that has already generated a response). For similar reasons, several subfields in the Frame Control field are masked to 0. AAD construction is performed as follows:

- a. FC – MPDU Frame Control field, with
 - 1. Subtype bits (bits 4 5 6) in a Data MPDU masked to 0
 - 2. Retry bit (bit 11) masked to 0
 - 3. Power Management bit (bit 12) masked to 0
 - 4. More Data bit (bit 13) masked to 0
 - 5. Protected Frame bit (bit 14) always set to 1
 - 6. Order bit (bit 15) as follows: i. Masked to 0 in all data MPDUs containing a QoS Control field ii. Unmasked otherwise
- b. A1 – MPDU Address 1 field
- c. A2 – MPDU Address 2 field
- d. A3 – MPDU Address 3 field
- e. SC – MPDU Sequence Control field, with the Sequence Number subfield (bits 4–15 of the Sequence Control field) masked to 0. The Fragment Number subfield is not modified
- f. A4 – MPDU Address field, if present.

g. QC – QoS Control field, if present, a 2-octet field that includes the MSDU priority. The QC TID is used in the construction of the AAD. When both the STA and its peer have their SPP A-MSDU Capable fields equal to 1, bit 7 (the A-MSDU Present field) is used in the construction of the AAD. The remaining QC fields are masked to 0 for the AAD calculation (bits 4 to 6, bits 8 to 15, and bit 7 when either the STA or its peer has the SPP A-MSDU Capable field equal to 0).

Construct CCM nonce

The Nonce field occupies 13 octets, and its structure is shown in Figure 11-19. The structure of the Nonce Flags subfield of the Nonce field is shown in Figure 11-20



Figure 11-19—Nonce construction

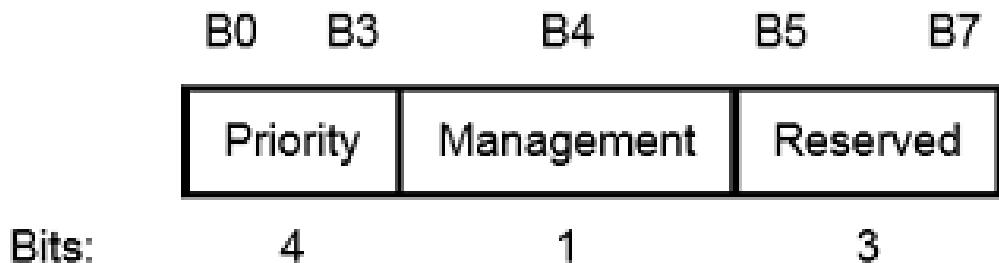


Figure 11-20—Nonce Flags subfield

Figure 7.122: NONCE Formatting for Wifi Security Protocol

The Nonce field has an internal structure of Nonce Flags || A2 || PN (“||” is concatenation), where

- The Priority subfield of the Nonce Flags field shall be set to the fixed value 0 when there is no QC field present in the MPDU header. When the QC field is present, bits 0 to 3 of the Priority subfield shall be set to the value of the QC TID (bits 0 to 3 of the QC field)
- When management frame protection is negotiated, the Management field of the Nonce Flags field shall be set to 1 if the Type field of the Frame Control field is 00 (Management frame); otherwise it is set to 0
- Bits 5 to 7 of the Nonce Flags field are reserved and shall be set to 0 on transmission
- MPDU address A2 field occupies octets 1–6. This shall be encoded with the octets ordered with A2 octet 0 at octet index 1 and A2 octet 5 at octet index 6

- The PN field occupies octets 7–12. The octets of PN shall be ordered so that PN0 is at octet index 12 and PN5 is at octet index 7.

Construct CCMP header

The format of the 8-octet CCMP header is given in 11.4.3.2. The header encodes the PN, Key ID, and ExtIV field values used to encrypt the MPDU.

CCM originator processing

CCM is a generic authenticate-and-encrypt block cipher mode, and in this standard, CCM is used with the AES block cipher. There are four inputs to CCM originator processing:

- a. Key: the temporal key (16 octets)
- b. Nonce: the nonce (13 octets) constructed as described in 11.4.3.3.4
- c. Frame body: the frame body of the MPDU
- d. AAD: the AAD (22–30 octets) constructed from the MPDU header as described in 11.4.3.3.3.

The CCM originator processing provides authentication and integrity of the frame body and the AAD as well as data confidentiality of the frame body. The output from the CCM originator processing consists of the encrypted data and 8 additional octets of encrypted MIC (see Figure 11-16). A CCMP protected individually addressed robust management frame shall be protected with the TK.

CCMP decapsulation

CCMP decrypts the payload of a cipher text MPDU and decapsulates a plaintext MPDU using the following steps:

- a. The encrypted MPDU is parsed to construct the AAD and nonce values
- b. The AAD is formed from the MPDU header of the encrypted MPDU
- c. The Nonce value is constructed from the A2, PN, and Nonce Flags fields
- d. The MIC is extracted for use in the CCM integrity checking
- e. The CCM recipient processing uses the temporal key, AAD, nonce, MIC, and MPDU cipher text data to recover the MPDU plaintext data as well as to check the integrity of the AAD and MPDU plaintext data
- f. The received MPDU header and the MPDU plaintext data from the CCM recipient processing are concatenated to form a plaintext MPDU
- g. The decryption processing prevents replay of MPDUs by validating that the PN in the MPDU is greater than the replay counter maintained for the session

When the received frame is a CCMP protected individually addressed robust management frame, contents of the MMPDU body after protection is removed shall be delivered to the SME via the MLME primitive designated for that management frame rather than through the MA-UNITDATA.indication primitive.

CCM recipient processing

CCM recipient processing uses the same parameters as CCM originator processing. A CCMP protected individually addressed robust management frame shall use the same TK as a Data MPDU.

Figure 11-21—CCMP decapsulation block diagram

There are four inputs to CCM recipient processing:

- Key: the temporal key (16 octets)
- Nonce: the nonce (13 octets) constructed as described in 11.4.3.3.4
- Encrypted frame body: the encrypted frame body from the received MPDU. The encrypted frame body includes an 8-octet MIC
- AAD: the AAD (22–30 octets) that is the canonical MPDU header as described in 11.4.3.3.3.

The CCM recipient processing checks the authentication and integrity of the frame body and the AAD as well as decrypting the frame body. The plaintext is returned only if the MIC check is successful. There is one output from error-free CCM recipient processing:

- Frame body: the plaintext frame body, which is 8 octets smaller than the encrypted frame body.

Decrypted CCMP MPDU

The decapsulation process succeeds when the calculated MIC matches the MIC value obtained from decrypting the received encrypted MPDU. The original MPDU header is concatenated with the plaintext data resulting from the successful CCM recipient processing to create the plaintext MPDU.

PN and replay detection

To effect replay detection, the receiver extracts the PN from the CCMP header. See 11.4.3.2 for a description of how the PN is encoded in the CCMP header. The following processing rules are used to detect replay:

- a. The PN values sequentially number each MPDU
- b. Each transmitter shall maintain a single PN (48-bit counter) for each PTKSA, GTKSA, and STKSA
- c. The PN shall be implemented as a 48-bit monotonically incrementing non-negative integer, initialized to 1 when the corresponding temporal key is initialized or refreshed
- d. A receiver shall maintain a separate set of PN replay counters for each PTKSA, GTKSA, and STKSA. The receiver initializes these replay counters to 0 when it resets the temporal key for a peer. The replay counter is set to the PN value of accepted CCMP MPDUs
- e. For each PTKSA, GTKSA, and STKSA, the recipient shall maintain a separate replay counter for each IEEE 802.11 MSDU or A-MSDU priority and shall use the PN recovered from a received frame to detect replayed frames, subject to the limitation of the number of supported replay counters indicated in the RSN Capabilities field (see 8.4.2.27). A replayed frame occurs when the PN extracted from a received frame is less than or equal to the current replay counter value for the frame's MSDU or A-MSDU priority and frame type. A transmitter shall not use IEEE 802.11 MSDU or A-MSDU priorities without ensuring that the receiver supports the required number of replay counters. The transmitter shall not reorder frames within a replay counter, but may reorder frames across replay counters. One possible reason for reordering frames is the IEEE 802.11 MSDU or A-MSDU priority
- f. If `dot11RSNAProtectedManagementFramesActivated` is true, the recipient shall maintain a single replay counter for received individually addressed robust management frames and shall use the PN from the received frame to detect replays. A replayed frame occurs when the PN from the frame is less than or equal to the current management frame replay counter value. The transmitter shall preserve the order of protected robust management frames sent to the same DA
- g. The receiver shall discard MSDUs, A-MSDUs, and MMPDUs whose constituent MPDU PN values are not sequential. A receiver shall discard any MPDU that is received with its PN less than or equal to the replay counter. When discarding a frame, the receiver shall increment by 1 the value of `dot11RSNASTatsCCMPReplays` for data frames or `dot11RSNASTatsRobustMgmtCCMPReplays` for robust management frames.
- h. For MSDUs or A-MSDUs sent using the Block Ack feature, reordering of received MSDUs or AMSDUs according to the Block Ack receiver operation (described in 9.21.4) is performed prior to replay detection

7.136.3 Broadcast/multicast integrity protocol (BIP)

BIP provides data integrity and replay protection for group addressed robust Management frames after successful establishment of an IGTKSA (see 12.6.1.1.9). BIP-CMAC-128 provides data integrity and replay protection, using AES-128 in CMAC Mode with a 128-bit integrity key and a CMAC TLen value of 128 (16 octets). BIP-CMAC-256 provides data integrity and replay protection, using AES-256 in CMAC Mode with a 256-bit integrity key and a CMAC TLen value of 128 (16 octets). NIST Special Publication 800-38B defines the CMAC algorithm, and NIST SP 800-38D defines the GMAC algorithm. BIP processing uses AES with a 128-bit or 256-bit integrity key and a CMAC TLen value of 128 (16 octets). The CMAC output for BIP-CMAC-256 is not truncated and shall be 128 bits (16 octets). The CMAC output for BIP-CMAC-128 is truncated to 64 bits:

- MIC = Truncate-64(CMAC Output)

BIP-GCMP-128 uses AES with a 128-bit integrity key, and BIP-GCMP-256 uses AES with a 256-bit integrity key. The authentication tag for both BIP-GCMP-128 and BIP-GCMP-256 is not truncated and shall be 128 bits (16 octets)

BIP uses the IGTK to compute the MMPDU MIC. The authenticator shall distribute one new IGTK and IGTK PN (IPN) whenever it distributes a new GTK. The IGTK is identified by the MAC address of the transmitting STA plus an IGTK identifier that is encoded in the MME Key ID field

BIP MMPDU format

The Management MIC element shall follow all of the other elements in the management frame body but precede the FCS. See 9.4.2.55 for the format of the Management MIC element. Figure 12-22 shows the BIP MMPDU



Figure 12-22—BIP Encapsulation

Figure 7.123: BIP Encapsulation

BIP AAD construction

The BIP Additional Authentication Data (AAD) shall be constructed from the MPDU header. The Duration field in the AAD shall be masked to 0. The AAD construction shall use a copy of the IEEE 802.11 header without the SC field for the MPDU, with the following exceptions:

a. FC—MPDU Frame Control field, with:

1. Retry subfield (bit 11) masked to 0
2. Power Management subfield (bit 12) masked to 0
3. More Data subfield (bit 13) masked to 0

b. A1—MPDU Address 1 field

c. A2—MPDU Address 2 field

d. A3—MPDU Address 3 field

Figure 12-23 depicts the format of the AAD. The length of the AAD is 20 octets

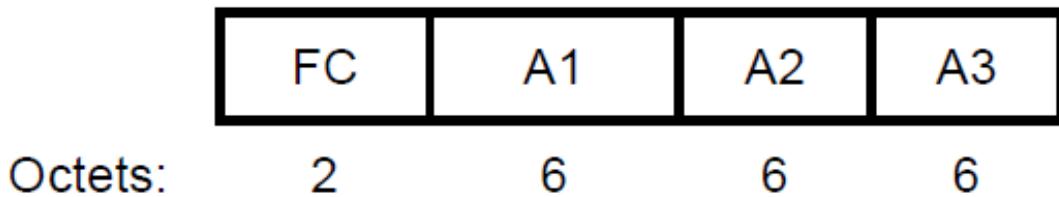


Figure 7.124: BIP AAD formatting

BIP replay protection

The MME Sequence Number field represents a sequence number whose length is 6 octets. When management frame protection is negotiated, the receiver shall maintain a 48-bit replay counter for each IGTK. The receiver shall set the receive replay counter to the value of the IPN in the IGTK key data encapsulation (KDE) (see 12.7.2) provided by the Authenticator in either the 4-way handshake, FT 4-way handshake, FT handshake, or group key handshake. The transmitter shall maintain a single IPN for each IGTK. The IPN shall be implemented as a 48-bit strictly increasing integer, initialized to 1 when the corresponding IGTK is initialized. The transmitter may reinitialize the sequence counter when the IGTK is refreshed. See 12.5.4.5 and 12.5.4.6 for per packet BIP processing.

NOTE—When the IPN space is exhausted, the choices available to an implementation are to replace the IGTK or to end communications

When `dot11QMFActivated` is true, the receiver shall maintain an additional replay counter for each ACI for received group addressed robust Management frames that use QMF. The receiver shall use the ACI encoded in the Sequence Number field of received GQMFs protected by BIP to select the replay counter to use for the received frame, and shall use the IPN from the received frame to detect replays

If `dot11RSNAProtectedManagementFramesActivated` is true and `dot11MeshSecurityActivated` is true, the recipient shall maintain a single replay counter for received group addressed robust Management frames that do not use the QMF service and shall use the PN from the received frame to detect replays. If `dot11QMFActivated` is also true, the recipient shall maintain an additional replay counter for each ACI for received group addressed robust Management frames that use the QMF service. The QMF receiver shall use the ACI encoded in the Sequence Number field of the received frame to select the replay counter to use for the received frame, and shall use the PN from the received frame to detect replays. A replayed frame occurs when the PN from the frame is less than or equal to the value of the management frame replay counter that corresponds to the ACI of the frame. The transmitter shall preserve the order of protected robust Management frames transmitted to the same DA without the QMF service. When the QMF service is used, the transmitter shall not reorder robust GQMFs within an AC when the frames are transmitted to the same RA

BIP transmission

When a STA transmits a protected group addressed robust Management frame, it shall

- a. Select the IGTK currently active for transmission of frames to the intended group of recipients and construct the MME (see 9.4.2.55) with the MIC field masked to 0 and the Key ID field set to the corresponding IGTK Key ID value. If the frame is not a GQMF, the transmitting STA shall insert a strictly increasing integer into the MME IPN field. If the frame is a GQMF, then the transmitting STA shall maintain a 48-bit counter for use as the IPN, the counter shall be incremented for each GQMF until the two least significant bits of the counter match the ACI of the AC that is used to transmit the frame, and the counter value shall be inserted into the MME IPN field of the frame. For BIP-GMAC-128 and BIP-GMAC-256, the initialization vector passed to GMAC shall be a concatenation of Address 2 from the MAC header of the MPDU and the non-negative integer inserted into the MMP IPN field
- b. Compute AAD as specified in 12.5.4.3
- c. Compute an integrity value over the concatenation of AAD and the management frame body including MME, and insert the output into the MME MIC field. For BIP-CMAC-128, the integrity value is 64 bits and is computed using AES-128-CMAC; for BIP-CMAC-256, the integrity value is 128 bits and is computed using AES-256-CMAC; for BIP-GMAC-128, the integrity value is 128 bits and is computed using AES-128-GMAC; and, for BIP-GMAC-256, the integrity value is 128 bits and is computed using AES-256-GMAC.
- d. Compose the frame as the IEEE 802.11 header, management frame body, including MME, and FCS. The MME shall appear last in the frame body
- e. Transmit the frame

BIP reception

When a STA with management frame protection negotiated receives a group addressed robust Management frame protected by BIP-CMAC-128, BIP-CMAC-256, BIP-GMAC-128, or BIP-GMAC-256, it shall

- a. Identify the appropriate IGTK and associated state based on the MME Key ID field. If no such IGTK exists, silently drop the frame and terminate BIP processing for this reception
- b. Perform replay protection on the received frame. The receiver shall interpret the MME IPN field as a 48-bit unsigned integer
 1. If the frame is not a GQMF, the receiver shall compare this MME IPN integer value to the value of the receive replay counter for the IGTK identified by the MME Key ID field. If the integer value from the received MME IPN field is less than or equal to the replay counter value for this IGTK, the receiver shall discard the frame and increment the `dot11RSNAStatsCMACReplays` counter by 1
 2. If the frame is a GQMF, the receiver shall compare this MME IPN integer value to the value of the receive replay counter for the IGTK identified by the MME Key ID field and the AC represented by the value of the ACI subfield of the received frame. If the integer value from the received MME IPN field is less than or equal

to the replay counter value for this IGTK and AC, the receiver shall discard the frame and increment the dot11RSNAStatsCMACReplays counter by 1

c. Compute AAD for this Management frame, as specified in 12.5.4.3. For BIP-GMAC-128 and BIPGMAC-256, an initialization vector for GMAC is constructed as the concatenation of Address 2 from the MAC header of the MPDU and the 48-bit unsigned integer from the MME IPN field

d. Extract and save the received MIC value, and compute a verifier over the concatenation of AAD, the management frame body, and MME, with the MIC field masked to 0 in the MME. For BIP-CMAC-128, the integrity value is 64 bits and is computed using AES-128-CMAC; for BIP-CMAC-256, the integrity value is 128 bits and is computed using AES-256-CMAC; for BIP-GMAC-128, the integrity value is 128 bits and is computed using AES-128-GMAC; and, for BIP-GMAC-256, the integrity value is 128 bits and is computed using AES-256-GMAC. If the result does not match the received MIC value, then the receiver shall discard the frame, increment the dot11RSNAStatsBIPMICErrors counter by 1, and terminate BIP processing for this reception

e. If the frame is not a GQMF, update the replay counter for the IGTK identified by the MME Key ID field with the integer value of the MME IPN field

f. If the frame is a GQMF, update the replay counter for the IGTK identified by the MME Key ID field and the AC represented by the value of the ACI subfield of the received frame with the integer value of the MME IPN field

If management frame protection is negotiated, group addressed robust Management frames that are received without BIP protection shall be discarded

7.136.4 AES-GCM Use in Wifi

IEEE-802.11-2016 specifies the GCMP, which provides data confidentiality, authentication, integrity, and replay protection. A DMG RSNA STA shall support GCMP-128. GCMP is based on the GCM of the AES encryption algorithm. GCM protects the integrity of both the MPDU Data field and selected portions of the MPDU header. The AES algorithm is defined in FIPS PUB 197. All AES processing used within GCMP uses AES with a 128-bit key (GCMP-128) or a 256-bit key (GCMP-256). GCM is defined in NIST Special Publication 800-38D. GCM is a generic mode that can be used with any block-oriented encryption algorithm

GCM requires a fresh temporal key for every session. GCM also requires a unique nonce value for each frame protected by a given temporal key, and GCMP uses a 96-bit nonce that includes a 48-bit packet number (PN) for this purpose. Reuse of a PN with the same temporal key voids all security guarantees. GCMP uses a 128-bit MIC

When GCMP is selected as the RSN pairwise cipher and management frame protection is negotiated, individually addressed robust Management frames and the group addressed Management frames that receive "Group Addressed Privacy" as indicated in Table 9-47 shall be protected with GCMP

GCMP MPDU format

Figure 12-24 depicts the MPDU when using GCMP

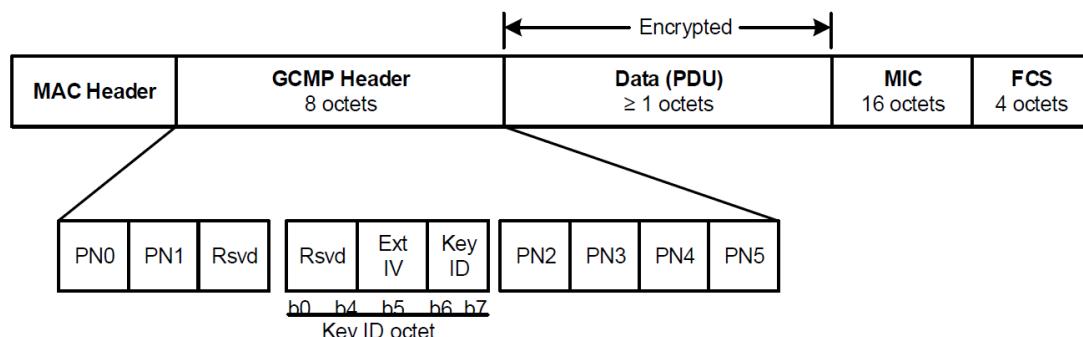


Figure 7.125: Wifi Packet Formatting For GCMP Encapsulation

GCMP cryptographic encapsulation

GCMP encrypts the Frame Body field of a plaintext MPDU and encapsulates the resulting cipher text using the following steps:

- a. Increment the PN, to obtain a fresh PN for each MPDU, so that the PN never repeats for the same temporal key
- b. Use the fields in the MPDU header to construct the additional authentication data (AAD) for GCM. The GCM algorithm provides integrity protection for the fields included in the AAD. MPDU header fields that may change when retransmitted are masked to 0 when calculating the AAD
- c. Construct the GCM Nonce block from the PN and A2, where A2 is MPDU Address 2
- d. Place the new PN and the key identifier into the 8-octet GCMP Header
- e. Use the temporal key, AAD, nonce, and MPDU data to form the cipher text and MIC. This step is known as GCM originator processing
- f. Form the encrypted MPDU by combining the original MPDU header, the GCMP header, the encrypted data and MIC

GCMP PN processing

The PN is incremented by a positive number for each MPDU. The PN shall be incremented in steps of 1 for constituent MPDUs of fragmented MSDUs and MMPDUs. The PN shall never repeat for a series of encrypted MPDUs using the same temporal key. If the PN is larger than dot11PNExhaustionThreshold, an MLME-PN-EXHAUSTION-indication primitive shall be generated

Construct AAD

The AAD for GCMP is constructed in the same manner as CCMP

Construct GCM nonce

The Nonce field occupies 12 octets, and its structure is shown in Figure 12-26

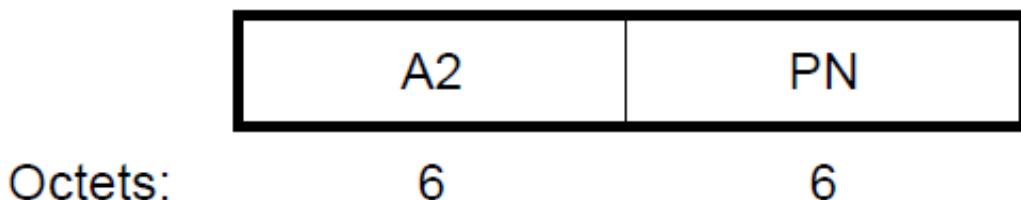


Figure 7.126: NONCE Formatting of Wifi GCM Security Packets

The Nonce field has an internal structure of A2 || PN, where

- MPDU address A2 field occupies octets 0 to 5. This shall be encoded with the octets ordered with A2 octet 0 at octet index 0 and A2 octet 5 at octet index 5
- The PN field occupies octets 6 to 11. The octets of PN shall be ordered so that PN0 is at octet index 11 and PN5 is at octet index 6

GCM originator processing

GCM is a generic authenticate-and-encrypt block cipher mode, and in this standard, GCM is used with the AES block cipher

There are four inputs to GCM originator processing:

- a. Key: the temporal key (16 octets)
- b.Nonce: the nonce (12 octets) constructed as described in 12.5.5.3.4
- c. Frame body: the plaintext frame body of the MPDU

d. AAD: the AAD (22-30 octets) constructed from the MPDU header as described in 12.5.5.3.3

The GCM originator processing provides authentication and integrity of the frame body and the AAD as well as data confidentiality of the frame body. The output from the GCM originator processing consists of the encrypted data and 16 additional octets of encrypted MIC (see Figure 12-24). The PN values sequentially number each MPDU. Each transmitter shall maintain a single PN (48-bit counter) for each PTKSA, GTKSA, and STKSA. The PN shall be implemented as a 48-bit strictly increasing integer, initialized to 1 when the corresponding temporal key is initialized or refreshed

A transmitter shall not use IEEE 802.11 MSDU or A-MSDU priorities without ensuring that the receiver supports the required number of replay counters. The transmitter shall not reorder GCMP protected frames that are transmitted to the same RA within a replay counter, but may reorder frames across replay counters

One possible reason for reordering frames is the IEEE 802.11 MSDU or A-MSDU priority. The transmitter shall preserve the order of protected robust Management frames that are transmitted to the same DA without the QMF service. When the QMF service is used, the transmitter shall not reorder robust IQMFs within an AC when the frames are transmitted to the same RA

A GCMP protected individually addressed robust Management frame shall be protected using the same TK as a Data frame

GCMP decapsulation

GCMP decrypts the Frame Body field of a cipher text MPDU and decapsulates a plaintext MPDU using the following steps:

- a. The encrypted MPDU is parsed to construct the AAD and nonce values
- b. The AAD is formed from the MPDU header of the encrypted MPDU
- c. The Nonce value is constructed from the A2 and PN fields
- d. The MIC is extracted for use in the GCM integrity checking
- e. The GCM recipient processing uses the temporal key, AAD, nonce, MIC, and MPDU cipher text data to recover the MPDU plaintext data as well as to check the integrity of the AAD and MPDU plaintext data
- f. The received MPDU header and the MPDU plaintext data from the GCM recipient processing are concatenated to form a plaintext MPDU
- g. The decryption processing prevents replay of MPDUs by validating that the PN in the MPDU is greater than the replay counter maintained for the session

When the received frame is a GCMP protected individually addressed robust Management frame, the contents of the MMPDU body after protection is removed and shall be delivered to the SME via the MLME primitive designated for that MMPDU rather than through the MA-UNITDATA.indication primitive

GCM recipient processing

GCM recipient processing shall use the same parameters as GCM originator processing. A GCMP protected individually addressed robust Management frame shall use the same TK as a Data frame

There are four inputs to GCM recipient processing:

- Key: the temporal key (16 octets)
- Nonce: the nonce (12 octets) constructed as described in 12.5.5.4
- Encrypted frame body: the encrypted frame body from the received MPDU. The encrypted frame body includes a 16-octet MIC
- AAD: the AAD (22-30 octets) that is the canonical MPDU header as described in 12.5.5.3.3

The GCM recipient processing checks the authentication and integrity of the frame body and the AAD as well as decrypting the frame body. The plaintext is returned only if the MIC check is successful

There is one output from error-free GCM recipient processing:

- Frame body: the plaintext frame body, which is 16 octets smaller than the encrypted frame body

Decrypted GCMP MPDU

The decapsulation process succeeds when the calculated MIC matches the MIC value obtained from decrypting the received encrypted MPDU. The original MPDU header is concatenated with the plaintext data resulting from the successful GCM recipient processing to create the plaintext MPDU

PN and replay detection

To effect replay detection, the receiver extracts the PN from the GCMP header. See 12.5.5.2 for a description of how the PN is encoded in the GCMP header. The following processing rules are used to detect replay:

- a. The receiver shall maintain a separate set of replay counters for each PTKSA, GTKSA, and STKSA. The receiver initializes these replay counters to 0 when it resets the temporal key for a peer. The replay counter is set to the PN value of accepted GCMP MPDUs
- b. For each PTKSA, GTKSA, and STKSA, the recipient shall maintain a separate replay counter for each TID, subject to the limitation of the number of supported replay counters indicated in the RSN Capabilities field (see 9.4.2.25), and shall use the PN from a received frame to detect replayed frames. A replayed frame occurs when the PN from a received frame is less than or equal to the current replay counter value for the frame's MSDU or A-MSDU priority and frame type
- c. If dot11RSNAProtectedManagementFramesActivated is true, the recipient shall maintain a single replay counter for received individually addressed robust Management frames that are received with the To DS subfield equal to 0 and shall use the PN from the received frame to detect replays. If dot11QMFActivated is also true, the recipient shall maintain an additional replay counter for each ACI for received individually addressed robust Management frames that are received with the To DS subfield equal to 1. The QMF receiver shall use the ACI encoded in the Sequence Number field of the received frame to select the replay counter to use for the received frame, and shall use the PN from the received frame to detect replays. A replayed frame occurs when the PN from the frame is less than or equal to the current value of the management frame replay counter that corresponds to the ACI of the frame
- d. The receiver shall discard any Data frame that is received with its PN less than or equal to the value of the replay counter that is associated with the TA and priority value of the received MPDU. The receiver shall discard MSDUs and MMPDUs whose constituent MPDU PN values are not incrementing in steps of 1. If dot11RSNAProtectedManagementFramesActivated is true, the receiver shall discard any individually addressed robust Management frame that is received with its PN less than or equal to the value of the replay counter associated with the TA of that individually addressed Management frame
- e. When discarding a frame, the receiver shall increment by 1 dot11RSNAStatsGCMPReplays for Data frames or dot11RSNAStatsRobustMgmtGCMPReplays for robust Management frames
- f. For MSDUs or A-MSDUs sent using the block ack feature, reordering of received MSDUs or A-MSDUs according to the block ack receiver operation (described in 10.24.4) is performed prior to replay detection

MAC frame formats

Each frame consists of the following basic components:

- a. A MAC header, which comprises frame control, duration, address, optional sequence control information, optional QoS Control information (QoS data frames only), and optional HT Control fields (+HTC frames only)
- b. A variable-length frame body, which contains information specific to the frame type and subtype
- c. A FCS, which contains an IEEE 32-bit CRC

Conventions

Structures defined in the MAC sublayer are described as a sequence of fields in specific order. Each figure in Clause 8 depicts the fields/subfields as they appear in the MAC frame and in the order in which they are passed to the physical layer convergence procedure (PLCP), from left to right.

In figures, all bits within fields are numbered, from 0 to k, where the length of the field is k + 1 bits. Bits within numeric fields that are longer than a single bit are depicted in increasing order of significance, i.e., with the lowest numbered bit having the least significance. The octet boundaries within a field can be obtained by taking the bit numbers of the field modulo 8. Octets within numeric fields that are longer than a single octet are depicted in increasing order of significance, from lowest numbered bit to highest numbered bit. The octets in fields longer than a single octet are sent to the PLCP in order from the octet containing the lowest numbered bits to the octet containing the highest numbered bits.

Any field containing a CRC is an exception to this convention and is transmitted commencing with the coefficient of the highest-order term.

MAC addresses are assigned as ordered sequences of bits. The Individual/Group bit is always transferred first and is bit 0 of the first octet.

Organizationally unique identifiers (OUIs) and Organization Identifiers are specified in two forms: an ordered sequence of octets, and a numeric form. Treating the OUI or Organization Identifier as an ordered sequence of octets, the leftmost octet is always transferred first. This is equivalent to transmitting the most significant octet of the numeric form first.

Values specified in decimal are coded in natural binary unless otherwise stated. The values in Table8-1 are in binary, with the bit assignments shown in the table. Values in other tables are shown in decimal notation.

Reception, in references to frames or fields within frames (e.g., received Beacon frames or a received Duration-/ID field), applies to MPDUs or MAC management protocol data units (MMPDUs) indicated from the PHY layer without error and validated by FCS within the MAC sublayer. Without further qualification, reception by the MAC sublayer implies that the frame contents are valid, and that the protocol version is supported (see 8.2.4.1.2), with no implication regarding frame addressing or regarding whether the frame type or other fields in the MAC header are meaningful to the MAC entity that has received the frame.

A frame that contains the HT Control field, including the Control Wrapper frame, is referred to as a +HTC frame. A QoS Data frame that is transmitted by a mesh STA is referred to as a Mesh Data frame.

Parentheses enclosing portions of names or acronyms are used to designate a set of related names that vary based on the inclusion of the parenthesized portion. For example,

- QoS +CF-Poll frame refers to the three QoS data subtypes that include “+CF-Poll”: the QoS Data+CF-Poll frame, subtype 1010; QoS Data+CF-Ack+CF-Poll frame, subtype 1011; and QoS CF-Ack+CF-Poll frame, subtype 1111
- QoS CF-Poll frame refers specifically to the QoS CF-Poll frame, subtype 1110
- QoS (+)CF-Poll frame refers to all four QoS data subtypes with CF-Poll: the QoS CF-Poll frame, subtype 1110; the QoS CF-Ack+CF-Poll frame, subtype 1111; the QoS Data+CF-Poll frame, subtype 1010; and the QoS Data+CF-Ack+CF-Poll frame, subtype 1011
- QoS (+)Null frame refers to all three QoS data subtypes with “no data”: the QoS Null (no data) frame, subtype 1100; the QoS CF-Poll (no data) frame, subtype 1110; and the QoS CF-Ack+CF-Poll frame, subtype 1111
- QoS +CF-Ack frame refers to the three QoS data subtypes that include “+CF-Ack”: the QoS Data+CF-Ack frame, subtype 1001; QoS Data+CF-Ack+CF-Poll frame, subtype 1011; and QoS CF-Ack+CF-Poll frame, subtype 1111
- Whereas (QoS) CF-Poll frame refers to the QoS CF-Poll frame, subtype 1110, and the CF-Poll frame, subtype 0110.

Reserved fields and subfields are set to 0 upon transmission and are ignored upon reception.

General frame format

The MAC frame format comprises a set of fields that occur in a fixed order in all frames. Figure8-1 depicts the general MAC frame format. The first three fields (Frame Control, Duration/ID, and Address 1) and the last field (FCS) in Figure8-1 constitute the minimal frame format and are present in all frames, including reserved types and subtypes. The fields Address 2, Address 3, Sequence Control, Address 4, QoS Control, HT Control, and Frame Body are present only in certain frame types and subtypes. Each field is defined in 8.2.4. The format of each of the individual subtypes of each frame type is defined in 8.3. The components of management frame bodies are defined in 8.4. The formats of management frames of subtype Action are defined in 8.5.

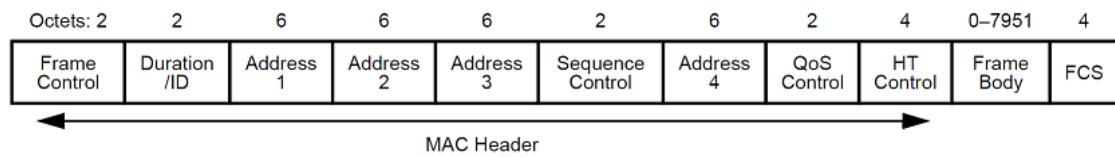
**Figure 8-1—MAC frame format**

Figure 7.127: General Frame Format for Wifi Packets

The Frame Body field is of variable size. The maximum frame body size is determined by the maximum MSDU size (2304 octets), plus the length of the Mesh Control field (6, 12, or 18 octets) if present, the maximum unencrypted MMPDU size excluding the MAC header and FCS (2304 octets) or the maximum AMSDU size (3839 or 7935 octets, depending upon the STA's capability), plus any overhead from security encapsulation.

Frame fields

The Frame Control field consists of the following subfields: Protocol Version, Type, Subtype, To DS, FromDS, More Fragments, Retry, Power Management, More Data, Protected Frame, and Order. The format of the Frame Control field is illustrated in Figure 8-2.

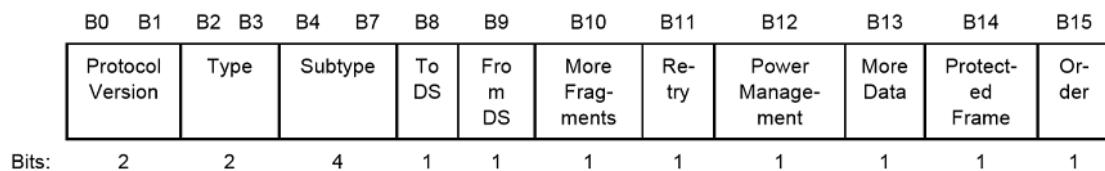
**Figure 8-2—Frame Control field**

Figure 7.128: Frame Control Field Format for Wifi Packets

Protocol Version field

The Protocol Version field is 2 bits in length and is invariant in size and placement across all revisions of this standard. For this standard, the value of the protocol version is 0. All other values are reserved. The revision level will be incremented only when a fundamental incompatibility exists between a new revision and the prior edition of the standard. See 9.24.2.

Type and Subtype fields

The Type field is 2 bits in length, and the Subtype field is 4 bits in length. The Type and Subtype fields together identify the function of the frame. There are three frame types: control, data, and management. Each of the frame types has several defined subtypes. In data frames, the most significant bit (MSB) of the Subtype field, b7, is defined as the QoS subfield. Table 8-1 defines the valid combinations of type and subtype. (The numeric values in Table 8-1 are shown in binary, see IEEE802.11-2012)

Each Subtype field bit position is used to indicate a specific modification of the basic data frame (subtype 0). Frame Control bit 4 is set to 1 in data subtypes that include +CF-Ack, bit 5 is set to 1 in data subtypes that include +CF-Poll, bit 6 is set to 1 in data subtypes that contain no Frame Body field, and bit 7 is set to 1 in the QoS data subtypes, which have QoS Control fields in their MAC headers.

Table 8-1—Valid type and subtype combinations

Type value b3 b2	Type description	Subtype value b7 b6 b5 b4	Subtype description
00	Management	0000	Association request
00	Management	0001	Association response
00	Management	0010	Reassociation request
00	Management	0011	Reassociation response
00	Management	0100	Probe request
00	Management	0101	Probe response
00	Management	0110	Timing Advertisement

Table 8-1—Valid type and subtype combinations (continued)

Type value b3 b2	Type description	Subtype value b7 b6 b5 b4	Subtype description
00	Management	0111	Reserved
00	Management	1000	Beacon
00	Management	1001	ATIM
00	Management	1010	Disassociation
00	Management	1011	Authentication
00	Management	1100	Deauthentication
00	Management	1101	Action
00	Management	1110	Action No Ack
00	Management	1111	Reserved
01	Control	0000–0110	Reserved
01	Control	0111	Control Wrapper
01	Control	1000	Block Ack Request (BlockAckReq)
01	Control	1001	Block Ack (BlockAck)
01	Control	1010	PS-Poll
01	Control	1011	RTS
01	Control	1100	CTS
01	Control	1101	ACK
01	Control	1110	CF-End
01	Control	1111	CF-End + CF-Ack
10	Data	0000	Data
10	Data	0001	Data + CF-Ack
10	Data	0010	Data + CF-Poll
10	Data	0011	Data + CF-Ack + CF-Poll
10	Data	0100	Null (no data)
10	Data	0101	CF-Ack (no data)
10	Data	0110	CF-Poll (no data)
10	Data	0111	CF-Ack + CF-Poll (no data)
10	Data	1000	QoS Data
10	Data	1001	QoS Data + CF-Ack
10	Data	1010	QoS Data + CF-Poll
10	Data	1011	QoS Data + CF-Ack + CF-Poll
10	Data	1100	QoS Null (no data)
10	Data	1101	Reserved
10	Data	1110	QoS CF-Poll (no data)
10	Data	1111	QoS CF-Ack + CF-Poll (no data)
11	Reserved	0000–1111	Reserved

Figure 7.129: Wifi Type and SubType Values

Copyright 2017-2019 John Peter Greninger : Generated on Sat Aug 31 2019 22:28:11 for Protocol++ (ProtocolPP) 3.0.1 by Doxygen

To DS and From DS fields

The meaning of the combinations of values for the To DS and From DS fields are shown in Table8-2

Table 8-2—To/From DS combinations in data frames

To DS and From DS values	Meaning
To DS = 0 From DS = 0	A data frame direct from one STA to another STA within the same IBSS, a data frame direct from one non-AP STA to another non-AP STA within the same BSS, or a data frame outside the context of a BSS, as well as all management and control frames.
To DS = 1 From DS = 0	A data frame destined for the DS or being sent by a STA associated with an AP to the Port Access Entity in that AP.
To DS = 0 From DS = 1	A data frame exiting the DS or being sent by the Port Access Entity in an AP, or a group addressed Mesh Data frame with Mesh Control field present using the three-address MAC header format.
To DS = 1 From DS = 1	A data frame using the four-address MAC header format. This standard defines procedures for using this combination of field values only in a mesh BSS.

Figure 7.130: Wifi To and From Data Combinations

More Fragments field

The More Fragments field is 1 bit in length and is set to 1 in all data or management type frames that have another fragment of the current MSDU or current MMPDU to follow. It is set to 0 in all other frames.

Retry field

The Retry field is 1 bit in length and is set to 1 in any data or management type frame that is a retransmission of an earlier frame. It is set to 0 in all other frames. A receiving STA uses this indication to aid in the process of eliminating duplicate frames.

Power Management field

The Power Management field is 1 bit in length and is used to indicate the power management mode of a STA. The value of this field is either reserved (as defined below) or remains constant in each frame from a particular STA within a frame exchange sequence (see AnnexG). The value indicates the mode of the STA after the successful completion of the frame exchange sequence.

In an infrastructure BSS, the following applies:

- The Power Management field is reserved in all management frames that are not bufferable management frames
- The Power Management field is reserved in all management frames transmitted by a STA to an AP with which it is not associated
- The Power Management field is reserved in all frames transmitted by the AP
- Otherwise, a value of 1 indicates that the STA will be in PS mode. A value of 0 indicates that the STA will be in active mode.

In an IBSS, the following applies:

- The Power Management field is reserved in all management frames that are not bufferable management frames and that are not individually addressed Probe Request frames
- Otherwise, a value of 1 indicates that the STA will be in PS mode. A value of 0 indicates that the STA will be in active mode.

In an MBSS, the following applies:

- A value of 0 in group addressed frames, in management frames transmitted to nonpeer STAs, and in Probe Response frames indicates that the mesh STA will be in active mode towards all neighbor mesh STAs. A value of 1 in group addressed frames, in management frames transmitted to nonpeer STAs, and in Probe Response frames indicates that the mesh STA will be in deep sleep mode towards all nonpeer mesh power STAs
- A value of 0 in individually addressed frames transmitted to a peer mesh STA indicates that the mesh STA will be in active mode towards this peer mesh STA. A value of 1 in individually addressed frames transmitted to a peer mesh STA, except Probe Response frames, indicates that the mesh STA will be in either light sleep mode or deep sleep mode towards this peer mesh STA. When the QoS Control field is present in the frame, the Mesh Power Save Level subfield in the QoS Control field indicates whether the mesh STA will be in light sleep mode or in deep sleep mode for the recipient mesh STA as specified in 8.2.4.5.11.

The mesh power mode transition rules are described in 13.14.3.

More Data field

The More Data field is 1 bit in length and is used to indicate to a STA in PS mode that more BUs are buffered for that STA at the AP. The More Data field is valid in individually addressed data or management type frames transmitted by an AP to a STA in PS mode. A value of 1 indicates that at least one additional buffered BU is present for the same STA.

The More Data field is optionally set to 1 in individually addressed data type frames transmitted by a CF Pollable STA to the PC in response to a CF-Poll to indicate that the STA has at least one additional buffered MSDU available for transmission in response to a subsequent CF-Poll.

For a STA in which the More Data Ack subfield of its QoS Capability element is 1 and that has APSD enabled, an AP optionally sets the More Data field to 1 in ACK frames to this STA to indicate that the AP has a pending transmission for the STA.

For a STA with TDLS peer PSM enabled and the More Data Ack subfield equal to 1 in the QoS Capability element of its transmitted TDLS Setup Request frame or TDLS Setup Response frame, a TDLS peer STA optionally sets the More Data field to 1 in ACK frames to this STA to indicate that it has a pending transmission for the STA.

The More Data field is 1 in individually addressed frames transmitted by a mesh STA to a peer mesh STA that is either in light sleep mode or in deep sleep mode for the corresponding mesh peering, when additional BUs remain to be transmitted to this peer mesh STA.

The More Data field is set to 0 in all other individually addressed frames.

The More Data field is set to 1 in group addressed frames transmitted by the AP when additional group addressed bufferable units (BUs) remain to be transmitted by the AP during this beacon interval. The More Data field is set to 0 in group addressed frames transmitted by the AP when no more group addressed BUs remain to be transmitted by the AP during this beacon interval and in all group addressed frames transmitted by non-AP STAs.

The More Data field is 1 in group addressed frames transmitted by a mesh STA when additional group addressed BUs remain to be transmitted. The More Data field is 0 in group addressed frames transmitted by a mesh STA when no more group addressed BUs remain to be transmitted.

Protected Frame field

The Protected Frame field is 1 bit in length. The Protected Frame field is set to 1 if the Frame Body field contains information that has been processed by a cryptographic encapsulation algorithm. The Protected Frame field is set to 1 only within data frames and within management frames of subtype Authentication, and individually addressed robust management frames. The Protected Frame field is set to 0 in all other frames. When the Protected Frame field is equal to 1, the Frame Body field is protected utilizing the cryptographic encapsulation algorithm and expanded as defined in Clause 11. The Protected Frame field is set to 0 in Data frames of subtype Null Function, CF-ACK (no data), CF-Poll (no data), CF-ACK+CF-Poll (no data), QoS Null (no data), QoS CF-Poll (no data), and QoS CF-ACK+CF-Poll (no data) (see, for example, 11.4.2.2 and 11.4.3.1 that show that the frame body needs to be 1 octet or longer to apply the encapsulation).

Order field

The Order field is 1 bit in length. It is used for two purposes:

- It is set to 1 in a non-QoS data frame transmitted by a non-QoS STA to indicate that the frame contains an MSDU, or fragment thereof, that is being transferred using the StrictlyOrdered service class
- It is set to 1 in a QoS data or management frame transmitted with a value of HT_GF or HT_MF for the FO-RMAT parameter of the TXVECTOR to indicate that the frame contains an HT Control field. Otherwise, the Order field is set to 0.

Duration/ID field

The Duration/ID field is 16 bits in length. The contents of this field vary with frame type and subtype, with whether the frame is transmitted during the CFP, and with the QoS capabilities of the sending STA. The contents of the field are defined as follows:

- In control frames of subtype PS-Poll, the Duration/ID field carries the association identifier (AID) of the STA that transmitted the frame in the 14 least significant bits (LSB), and the 2 most significant bits (MSB) both set to 1. The value of the AID is in the range 1–2007
- In frames transmitted by the PC and non-QoS STAs, during the CFP, the Duration/ID field is set to a fixed value of 32768
- In all other frames sent by non-QoS STAs and control frames sent by QoS STAs, the Duration/ID field contains a duration value as defined for each frame type in 8.3
- In data and management frames sent by QoS STAs, the Duration/ID field contains a duration value as defined for each frame type in 8.2.5.

See 9.24.3 on the processing of this field in received frames.

The encoding of the Duration/ID field is given in Table8-3.

Table 8-3—Duration/ID field encoding

Bits 0–13	Bit 14	Bit 15	Usage
0–32 767		0	Duration value (in microseconds) within all frames other than PS-Poll frames transmitted during the CP, and under HCF for frames transmitted during the CFP
0	0	1	Fixed value under point coordination function (PCF) within frames transmitted during the CFP
1–16 383	0	1	Reserved
0	1	1	Reserved
1–2007	1	1	AID in PS-Poll frames
2008–16 383	1	1	Reserved

Figure 7.131: Wifi Duration or Identification Field Formatting

The Duration/ID fields in the MAC headers of MPDUs in an A-MPDU all carry the same value NOTE—The reference point for the Duration/ID field is the end of the PPDU carrying the MPDU Setting the Duration/ ID field to the same value in the case of A-MPDU aggregation means that each MPDU consistently specifies the same NAV setting.

Address fields

There are four address fields in the MAC frame format. These fields are used to indicate the basic service set identifier (BSSID), source address (SA), destination address (DA), transmitting STA address (TA), and receiving STA address (RA). Certain frames may not contain some of the address fields.

Certain address field usage is specified by the relative position of the address field (1–4) within the MAC header, independent of the type of address present in that field. For example, receiver address matching is always performed on the contents of the Address 1 field in received frames, and the receiver address of CTS and ACK frames is always obtained from the Address 2 field in the corresponding RTS frame, or from the frame being acknowledged.

Address representation

Each Address field contains a 48-bit address as defined in 9.2 of IEEE Std 802-2001

Address designation

A MAC sublayer address is one of the following two types:

- a. Individual address. The address assigned to a particular STA on the network
- b. Group address. A multidestination address, which may be in use by one or more STAs on a given network. The two kinds of group addresses are as follows:
 - 1. Multicast-group address. An address associated by higher level convention with a group of logically related STAs
 - 2. Broadcast address. A distinguished, predefined group address that always denotes the set of all STAs on a given LAN. All ones are interpreted to be the broadcast address. This group is predefined for each communication medium to consist of all STAs actively connected to that medium; it is used to broadcast to all the active STAs on that medium.

The address space is also partitioned into locally administered and universal (globally administered) addresses. The nature of a body and the procedures by which it administers these universal (globally administered) addresses is beyond the scope of this standard. See IEEE Std 802-2001 for more information

Sequence Control field

The Sequence Control field is 16 bits in length and consists of two subfields, the Sequence Number and the Fragment Number. The format of the Sequence Control field is illustrated in Figure 8-3. The sequence Control field is not present in control frames.

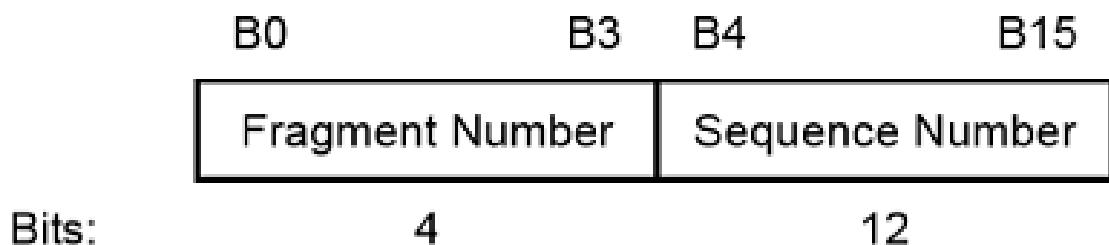


Figure 8-3—Sequence Control field

Figure 7.132: Wifi Sequence Control Field Formatting

Sequence Number field

The Sequence Number field is a 12-bit field indicating the sequence number of an MSDU, A-MSDU, or MMPDU. Each MSDU, A-MSDU, or MMPDU transmitted by a STA is assigned a sequence number. Sequence numbers are not assigned to control frames, as the Sequence Control field is not present.

Each fragment of an MSDU or MMPDU contains a copy of the sequence number assigned to that MSDU or MMPDU. The sequence number remains constant in all retransmissions of an MSDU, MMPDU, or fragment thereof.

Fragment Number field

The Fragment Number field is a 4-bit field indicating the number of each fragment of an MSDU or MMPDU. The fragment number is set to 0 in the first or only fragment of an MSDU or MMPDU and is incremented by one for each successive fragment of that MSDU or MMPDU. The fragment number is set to 0 in the only fragment of an A-MSDU. The fragment number remains constant in all retransmissions of the fragment.

QoS Control field

The QoS Control field is a 16-bit field that identifies the TC or TS to which the frame belongs as well as various other QoS-related, A-MSDU related, and mesh-related information about the frame that varies by frame type, subtype, and type of transmitting STA. The QoS Control field is present in all data frames in which the QoS subfield of the Subtype field is equal to 1 (see 8.2.4.1.3). Each QoS Control field comprises five or eight subfields, as defined for the particular sender (HC or non-AP STA) and frame type and subtype. The usage of these subfields and the various possible layouts of the QoS Control field are described 8.2.4.5.2 to 8.2.4.5.12 and illustrated in Table 8-4 of the IEEE802.11-2012 specification

Table 8-4—QoS Control field

Applicable frame (sub) types	Bits 0–3	Bit 4	Bits 5–6	Bit 7	Bits 8	Bit 9	Bit 10	Bits 11– 15
QoS CF-Poll and QoS CF-Ack+CF-Poll frames sent by HC	TID	EOSP	Ack Policy	Reserved	TXOP Limit			
QoS Data+CF-Poll and QoS Data+CF-Ack+CF-Poll frames sent by HC	TID	EOSP	Ack Policy	A-MSDU Present	TXOP Limit			
QoS Data and QoS Data+CF-Ack frames sent by HC	TID	EOSP	Ack Policy	A-MSDU Present	AP PS Buffer State			
QoS Null frames sent by HC	TID	EOSP	Ack Policy	Reserved	AP PS Buffer State			
QoS Data and QoS Data+CF-Ack frames sent by non-AP STAs that are not a TPU buffer STA or a TPU sleep STA in a nonmesh BSS	TID	0	Ack Policy	A-MSDU Present	TXOP Duration Requested			
	TID	1	Ack Policy	A-MSDU Present	Queue Size			

Table 8-4—QoS Control field (continued)

Applicable frame (sub) types	Bits 0–3	Bit 4	Bits 5–6	Bit 7	Bits 8	Bit 9	Bit 10	Bits 11– 15
QoS Null frames sent by non-AP STAs that are not a TPU buffer STA or a TPU sleep STA in a nonmesh BSS	TID	0	Ack Policy	Reserve d	TXOP Duration Requested			
	TID	1	Ack Policy	Reserve d	Queue Size			
QoS Data and QoS Data+CF-Ack frames sent by TPU buffer STAs in a nonmesh BSS	TID	EOSP	Ack Policy	A-MSDU Present	Reserved			
QoS Null frames sent by TPU buffer STAs in a nonmesh BSS	TID	EOSP	Ack Policy	Reserve d	Reserved			
QoS Data and QoS Data+CF-Ack frames sent by TPU sleep STAs in a nonmesh BSS	TID	Reserv ed	Ack Policy	A-MSDU Present	Reserved			
QoS Null frames sent by TPU sleep STAs in a nonmesh BSS	TID	Reserv ed	Ack Policy	Reserve d	Reserved			
All frames sent by mesh STAs in a mesh BSS	TID	EOSP	Ack Policy	A-MSDU Present	Mesh Control Present	Mesh Power Save Level	RSPI	Reserved

Figure 7.133: Wifi Quality of Service (QoS) Field Formatting

HT Control field

The HT Control field is always present in a Control Wrapper frame and is present in QoS Data and management frames as determined by the Order bit of the Frame Control field as defined in 8.2.4.1.10. NOTE—The only Control frame subtype for which HT Control field is present is the Control Wrapper frame. A control frame that is described as +HTC (e.g., RTS+HTC, CTS+HTC, BlockAck+HTC or BlockAckReq+HTC) implies the use of the Control Wrapper frame to carry that control frame. The format of the 4-octet HT Control field is shown in Figure 8-5. Subfields of the HT control field are defined in the 8.2.4.6 section of the IEEE802.11-2012 specification.

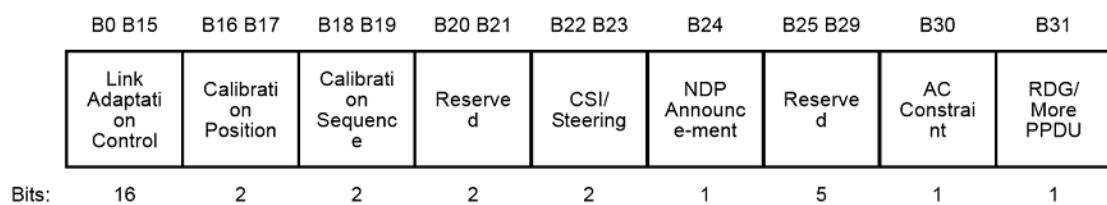
**Figure 8-5—HT Control field**

Figure 7.134: Wifi HT Control Field Formatting

Frame Body field

The Frame Body is a variable-length field that contains information specific to individual frame types and subtypes. The minimum length of the frame body is 0 octets. The maximum length of the frame body is defined by the maximum length MSDU plus the length of Mesh Control field as defined in 8.2.4.7.3, if present, plus any overhead for encryption as defined in Clause 11, or by the maximum length A-MSDU plus any overhead for encryption as

defined in Clause11.

Overhead for encryption

The overhead for encryption is described in Clause11. When the Mesh Control field is present in the frame body, the Mesh Control field is encrypted as a part of data.

Mesh Control field

The Mesh Control field is present in the unfragmented Mesh Data frame, in the first fragment of the Mesh Data frame, and in the management frame of subtype Action, Category Multihop Action (Multihop Action frame) transmitted by a mesh STA.

In Mesh Data frames, when the Mesh Control Present subfield in the QoS Control field is 1, the Mesh Control field is prepended to the MSDU and located as follows:

- When the frame body contains an MSDU (or a fragment thereof) and the frame is not encrypted, the Mesh Control field is located in the first octets of the frame body
- When the frame body contains an MSDU (or a fragment thereof) and the frame is encrypted, the Mesh Control field is located in the first octets of the encrypted data portion
- When the frame body contains an A-MSDU, the Mesh Control field is located in the Aggregate MSDU sub-frame header as shown in Figure8-33.

In the Multihop Action frame, the Mesh Control field is present as specified in 8.5.18.

The Mesh Control field is of variable length (6, 12, or 18 octets). The structure of the Mesh Control field is defined in Figure8-9.

FCS field

The FCS field is a 32-bit field containing a 32-bit CRC. The FCS is calculated over all the fields of the MAC header and the Frame Body field. These are referred to as the calculation fields.

The FCS is calculated using the following standard generator polynomial of degree 32:

$$G(x) = x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1$$

The FCS is the ones complement of the sum (modulo 2) of the following:

- a. The remainder of $x^k(x^{31} + x^{30} + x^{29} + \dots + x^2 + x + 1)$ divided (modulo 2) by G(x), where k is the number of bits in the calculation fields, and
- b. The remainder after multiplication of the contents (treated as a polynomial) of the calculation fields by x^{32} and then division by G(x).

The FCS field is transmitted commencing with the coefficient of the highest-order term.

As a typical implementation, at the transmitter, the initial remainder of the division is preset to all ones and is then modified by division of the calculation fields by the generator polynomial G(x). The ones complement of this remainder is transmitted, with the highest-order bit first, as the FCS field.

At the receiver, the initial remainder is preset to all ones and the serial incoming bits of the calculation fields and FCS, when divided by G(x), results (in the absence of transmission errors) in a unique nonzero remainder value. The unique remainder value is the polynomial:

$$G(x) = x^{31} + x^{30} + x^{26} + x^{25} + x^{24} + x^{18} + x^{15} + x^{14} + x^{12} + x^{11} + x^{10} + x^8 + x^6 + x^5 + x^4 + x^3 + x + 1$$

Data frame format

The format of a data frame is defined in Figure8-30. The Frame Control, Duration/ID, Address 1, Address 2, Address 3, and Sequence Control fields are present in all data frame subtypes. The presence of the Address 4 field is determined by the setting of the To DS and From DS subfields of the Frame Control field (see below). The QoS Control field is present when the QoS subfield of the Subtype field is set to 1.

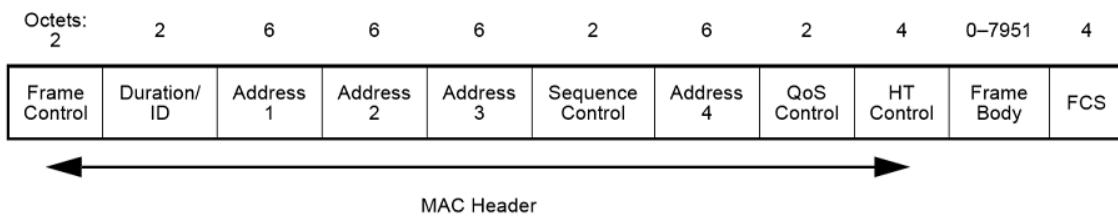
**Figure 8-30—Data frame**

Figure 7.135: WiFi Frame Formatting With Optional Fields

WiGig Frame Control Field format when type=1 and subtype=6

When the value of the Type subfield is equal to 1 and the value of the Subtype subfield is equal to 6, the remaining subfields within the Frame Control field are the following: Control Frame Extension, Power Management, More Data, Protected Frame, and Order. In this case, the format of the Frame Control field is illustrated in Figure 8-2a.

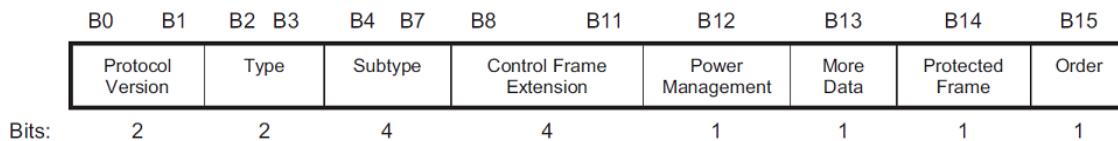
**Figure 8-2a—Frame Control field when Type is equal to 1 and Subtype is equal to 6**

Figure 7.136: Additional Formatting for WiGig Packets

For API Documentation:**See Also**

[ProtocolPP::jprotocol](#)
[ProtocolPP::jmodes](#)
[ProtocolPP::jwifisa](#)
[ProtocolPP::sm4](#)

For Additional Documentation:**See Also**

[jprotocol](#)
[jmodes](#)
[jwifisa](#)
[sm4](#)

Protocol++ written by : John Peter Greninger © John Peter Greninger 2015-2019 All Rights Reserved

All copyrights and trademarks are the property of their respective owners

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer

- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution
- Any and all modifications must be returned to John Peter Greninger at GitHub.com <https://github.com/jpgreninger/protocolpp> for evaluation. Inclusion of modifications in the source code shall be determined solely by John Peter Greninger. Failure to provide modifications shall render this license NULL and VOID and revoke any rights to use of Protocol++
- Commercial use requires a fee-based license obtainable at www.protocolpp.com
- Academic use requires written and notarized permission from John Peter and Sheila Rocha Greninger
- US Copyrights at [https://www.copyright.gov/](http://www.copyright.gov/)
 - TXU002059872 (Version 1.0.0)
 - TXU002066632 (Version 1.2.7)
 - TXU002082674 (Version 1.4.0)
 - TXU002097880 (Version 2.0.0)

The names of its contributors may not be used to endorse or promote products derived from this software without specific prior written permission

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE

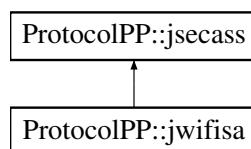
The documentation for this class was generated from the following file:

- [include/jwifi.h](#)

7.137 ProtocolPP::jwifisa Class Reference

```
#include <jwifisa.h>
```

Inheritance diagram for ProtocolPP::jwifisa:



Public Member Functions

- [jwifisa \(\)](#)
- [jwifisa \(direction_t dir, protocol_t mode, cipher_t cipher, auth_t auth, auth_t kdfalg, uint64_t addr1, uint64_t addr2, uint64_t addr3, uint64_t addr4, uint64_t pn, uint16_t framectl, wifictext_t cttext, uint16_t id, uint16_t seqctl, uint16_t qosctl, uint32_t htctl, uint8_t keyid, unsigned int icvlen, unsigned int ckeylen, std::shared_ptr<jarray<uint8_t>> cipherkey, unsigned int arlen, jarray<uint8_t> arwin, bool extiv, bool fcs, bool sppcap\)](#)
- [jwifisa \(jwifisa &rhs\)](#)
- [jwifisa \(std::shared_ptr<jwifisa > &rhs\)](#)

- virtual `~jwifisa ()`
Standard deconstructor.
- template<typename T >
`void set_field (field_t field, T fieldval)`
- template<typename T >
`T get_field (field_t field)`
- void `to_xml (tinyxml2::XMLPrinter &myxml, direction_t direction)`

7.137.1 Constructor & Destructor Documentation

7.137.1.1 ProtocolPP::jwifisa::jwifisa ()

Standard constructor with default settings

7.137.1.2 ProtocolPP::jwifisa::jwifisa (direction_t dir, protocol_t mode, cipher_t cipher, auth_t auth, auth_t kdfalg, uint64_t addr1, uint64_t addr2, uint64_t addr3, uint64_t addr4, uint64_t pn, uint16_t framectl, wifictext_t cttext, uint16_t id, uint16_t seqctl, uint16_t qosctl, uint32_t htctl, uint8_t keyid, unsigned int icvlen, unsigned int ckeylen, std::shared_ptr< jarray< uint8_t >> cipherkey, unsigned int arlen, jarray< uint8_t > arwin, bool extiv, bool fcs, bool sppcap)

Security Association for Wifi/WiGig

Parameters

<code>dir</code>	- Direction of processing (ENCAP or DECAP)
<code>mode</code>	- Protocol (either WIFI or WIGIG)
<code>cipher</code>	- Mode of encryption (AES_GCM, AES_CCM, SM4_GCM, SM4_CCM, NULL_CIPHER)
<code>auth</code>	- Mode of authentication (AES_CMAC, AES_GMAC, NULL_AUTH)
<code>kdfalg</code>	- Key derivation function (KDF) algorithm (SHA1, SHA256, SHA384, SHA512)
<code>addr1</code>	- First address
<code>addr2</code>	- Second address
<code>addr3</code>	- Third address
<code>addr4</code>	- Fourth address
<code>pn</code>	- Six byte packet number
<code>framectl</code>	- Frame control sequence
<code>cttext</code>	- Control frame extension
<code>id</code>	- Duration Identification
<code>seqctl</code>	- Sequence control
<code>qosctl</code>	- Quality of Service (QoS) control
<code>htctl</code>	- High Traffic (HT) control
<code>keyid</code>	- Key identification field
<code>icvlen</code>	- length of the ICV
<code>ckeylen</code>	- length of the key
<code>cipherkey</code>	- Key for the encryption algorithm
<code>arlen</code>	- Number of packets to track with replay window
<code>arwin</code>	- Anti-replay window for tracking packets
<code>extiv</code>	- Extended IV (always present for CCMP)
<code>fcs</code>	- Enables CRC-32 for the packet

<i>sppcap</i>	- SPP A-MSDU Capable field
---------------	----------------------------

7.137.1.3 ProtocolPP::jwifisa::jwifisa (*jwifisa* & *rhs*)

Constructor for Wifi

Parameters

<i>rhs</i>	- Security association (SA) for this Wifi flow
------------	--

7.137.1.4 ProtocolPP::jwifisa::jwifisa (std::shared_ptr< *jwifisa* > & *rhs*)

Constructor for Wifi

Parameters

<i>rhs</i>	- Security association (SA) for this Wifi flow
------------	--

7.137.1.5 virtual ProtocolPP::jwifisa::~jwifisa () [inline], [virtual]

Standard deconstructor.

7.137.2 Member Function Documentation

7.137.2.1 template<typename T> T ProtocolPP::jwifisa::get_field (*field_t* *field*)

Retrieve the field from the Wifi security association

Parameters

<i>field</i>	- field to retrieve
--------------	---------------------

Returns

field value

7.137.2.2 template<typename T> void ProtocolPP::jwifisa::set_field (*field_t* *field*, T *fieldval*)

Update Wifi field with the new value

Parameters

<i>field</i>	- field to update
<i>fieldval</i>	- new value for the field

7.137.2.3 void ProtocolPP::jwifisa::to_xml (tinyxml2::XMLPrinter & *myxml*, direction_t *direction*) [virtual]

Print the protocol and security objects to XML

Parameters

<i>myxml</i>	- XMLPrinter object to print with
<i>direction</i>	- facilitator for random generation

Implements [ProtocolIPP::jsecass](#).

The documentation for this class was generated from the following file:

- [include/jwifisa.h](#)

7.138 jwifisa Class Reference

```
#include "jwifisa.h"
```

7.138.1 Detailed Description

7.138.2 Wifi/WiGig Protocol Security Association

Pairwise Master Key (PMK) derivation

The pairwise master key (PMK) is derived from the group master key (GMK), and then used to create the pairwise transient key (PTK), the EAPOL key confirmation key (KCK), and the EAPOL key encryption key (KEK). There are two ways to obtain a PMK, either by use of a preshared key (PSK) or through the use of Extensible Authentication Protocol (EAP). When using the PSK, the PMK is derived by using the PBKDF2 algorithm to obtain the PMK from the passphrase and SSID as follows:

```
HMACSHAX(PSK,SSID,Iterations)
for i ← 0 do
    output ← HMACSHAX(PSK,SSID||0x00000001)
    input ← output
for i ← 1 to Iterations do
    output ← HMACSHAX(PSK,input)
    input ← output
return L(output,0,Len)
```

Pseudo random function (PRF)

A PRF is used in a number of places in this standard. Depending on its use, it may need to output 128 bits, 192 bits, 256 bits, 384 bits, or 512 bits. This subclause defines five functions:

- PRF-128, which outputs 128 bits
- PRF-192, which outputs 192 bits
- PRF-256, which outputs 256 bits
- PRF-384, which outputs 384 bits
- PRF-512, which outputs 512 bits

In the following, K is a key; A is a unique label for each different purpose of the PRF; B is a variable-length string; Y is a single octet containing 0; X is a single octet containing the loop parameter i; and || denotes concatenation:

$HMACSHA1(K,A,B,X) \leftarrow HMACSHA1(K,A||Y||B||X)$

$PRF(K,A,B,Len)$

```

for  $i \leftarrow 0$  to  $(Len + 159)/160$  do
     $R \leftarrow R || H_S HA_1(K, A, B, i)$ 
return L(R, 0, Len)

```

- PRF-128(K, A, B) = PRF(K, A, B, 128)
- PRF-192(K, A, B) = PRF(K, A, B, 192)
- PRF-256(K, A, B) = PRF(K, A, B, 256)
- PRF-384(K, A, B) = PRF(K, A, B, 384)
- PRF-512(K, A, B) = PRF(K, A, B, 512)

When the negotiated AKM is 00-0F-AC:5 or 00-0F-AC:6, the KDF specified in 11.6.1.7.2 shall be used instead of the PRF construction defined here. In this case, A is used as the KDF label and B as the KDF Context and the PRF functions are defined as follows:

- PRF-128(K, A, B) = KDF-128(K, A, B)
- PRF-192(K, A, B) = KDF-192(K, A, B)
- PRF-256(K, A, B) = KDF-256(K, A, B)
- PRF-384(K, A, B) = KDF-384(K, A, B)
- PRF-512(K, A, B) = KDF-512(K, A, B)

Key derivation function (KDF)

The KDF for the FT key hierarchy is a variant of the pseudorandom function (PRF) defined in 11.6.1.2 and is defined as follows:

Output $\leftarrow KDFLength(K, label, Context)$ where

- K, a 256-bit key derivation key
- label, a string identifying the purpose of the keys derived using this KDF
- Context, a bit string that provides context to identify the derived key
- Length, the length of the derived key in bits

Output: a Length-bit derived key

```

result  $\leftarrow ""$ 
iterations  $\leftarrow \frac{Length}{Hashlen}$ 
do  $i = 1$  to iterations
    result  $\leftarrow result || HMACSHA256(K, i || label || Context || Length)$ 
od

```

return first Length bits of *result*, and securely delete all unused bits

In this algorithm, i and $Length$ are encoded as 16-bit unsigned integers, represented using the bit ordering conventions of 8.2.2. K , $label$, and $Context$ are bit strings and are represented using the ordering conventions of 8.2.2. $Hashlen$ is the output length of the hash in bits

CTR with CBC-MAC Protocol (CCMP)

Subclause 11.4.3 specifies the CCMP, which provides data confidentiality, authentication, integrity, and replay protection. CCMP is mandatory for RSN compliance. CCMP is based on the CCM of the AES encryption algorithm.

CCM combines CTR for data confidentiality and CBC-MAC for authentication and integrity. CCM protects the integrity of both the MPDU Data field and selected portions of the IEEE 802.11 MPDU header.

The AES algorithm is defined in FIPS PUB 197-2001. All AES processing used within CCMP uses AES with a 128-bit key and a 128-bit block size. CCM is defined in IETF RFC 3610. CCM is a generic mode that can be used with any block-oriented encryption algorithm. CCM has two parameters (M and L), and CCMP uses the following values for the CCM parameters:

- M = 8; indicating that the MIC is 8 octets
- L = 2; indicating that the Length field is 2 octets, which is sufficient to hold the length of the largest possible IEEE 802.11 MPDU, expressed in octets

CCM requires a fresh temporal key for every session. CCM also requires a unique nonce value for each frame protected by a given temporal key, and CCMP uses a 48-bit packet number (PN) for this purpose. Reuse of a PN with the same temporal key voids all security guarantees.

When CCMP is selected as the RSN pairwise cipher and management frame protection is negotiated, individually addressed robust management frames and the group addressed management frames that receive "Group Addressed Privacy" as indicated in Table8-38 shall be protected with CCMP.

CCMP MPDU format

Figure11-16 depicts the MPDU when using CCMP

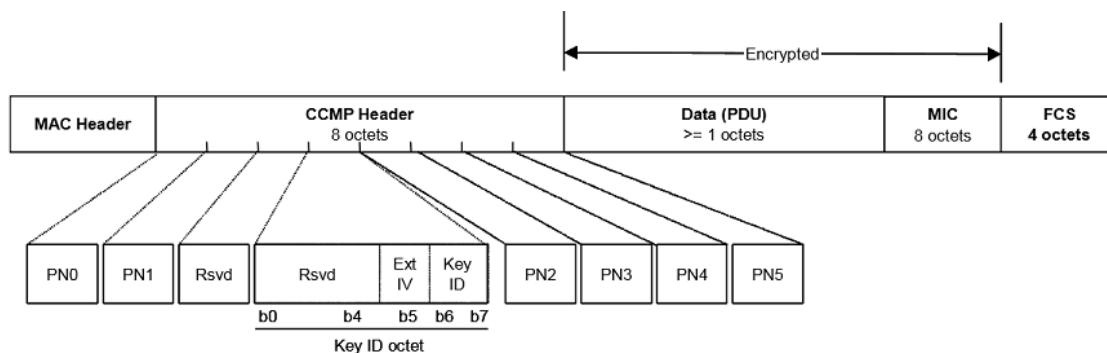


Figure 7.137: Wifi Encapsulation with CCMP Encoding

CCMP processing expands the original MPDU size by 16 octets, 8 octets for the CCMP Header field and 8 octets for the MIC field. The CCMP Header field is constructed from the PN, ExtIV, and Key ID subfields. PN is a 48-bit PN represented as an array of 6 octets. PN5 is the most significant octet of the PN, and PN0 is the least significant. Note that CCMP does not use the WEP ICV.

The ExtIV subfield (bit 5) of the Key ID octet signals that the CCMP Header field extends the MPDU header by a total of 8 octets, compared to the 4 octets added to the MPDU header when WEP is used. The ExtIV bit (bit5) is always set to 1 for CCMP.

Bits 6–7 of the Key ID octet are for the Key ID subfield.

The reserved bits shall be set to 0 and shall be ignored on reception.

CCMP cryptographic encapsulation

The CCMP cryptographic encapsulation process is depicted in Figure11-17.

CCMP encrypts the payload of a plaintext MPDU and encapsulates the resulting cipher text using the following steps:

- a. Increment the PN, to obtain a fresh PN for each MPDU, so that the PN never repeats for the same temporal key. Note that retransmitted MPDUs are not modified on retransmission
- b. Use the fields in the MPDU header to construct the additional authentication data (AAD) for CCM. The CCM

algorithm provides integrity protection for the fields included in the AAD. MPDU header fields that may change when retransmitted are muted by being masked to 0 when calculating the AAD

- c. Construct the CCM Nonce block from the PN, A2, and the Priority field of the MPDU where A2 is MPDU Address 2
- d. Place the new PN and the key identifier into the 8-octet CCMP header
- e. Use the temporal key, AAD, nonce, and MPDU data to form the cipher text and MIC. This step is known as CCM originator processing
- f. Form the encrypted MPDU by combining the original MPDU header, the CCMP header, the encrypted data and MIC, as described in 11.4.3.2

The CCM reference describes the processing of the key, nonce, AAD, and data to produce the encrypted output. See 11.4.3.3.2 to 11.4.3.3.6 for details of the creation of the AAD and nonce from the MPDU and the associated MPDU-specific processing.

PN processing

The PN is incremented by a positive number for each MPDU. The PN shall never repeat for a series of encrypted MPDUs using the same temporal key.

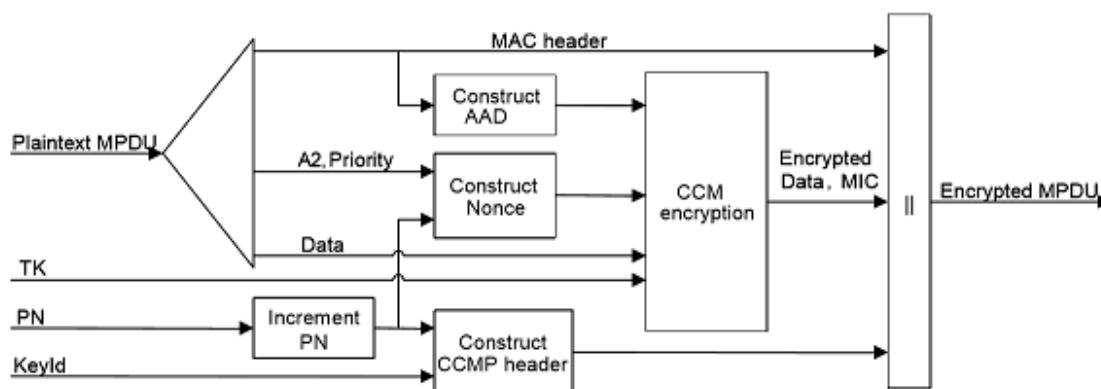


Figure 11-17—CCMP encapsulation block diagram

Figure 7.138: CCMP Encapsulation for Wifi Packets

Construct AAD

The format of the AAD is shown in Figure 11-18

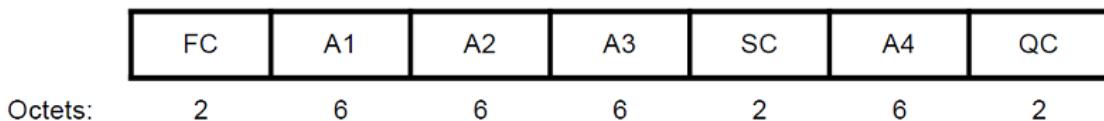


Figure 11-18—AAD construction

Figure 7.139: CCMP Authentication Algorithm Data (AAD) Formatting

The length of the AAD varies depending on the presence or absence of the QC and A4 fields and is shown in Table 11-1

Table 11-1—AAD length

QC field	A4 field	AAD length (octets)
Absent	Absent	22
Present	Absent	24
Absent	Present	28
Present	Present	30

Figure 7.140: CCMP Length Calculation for Authentication Algorithm Data (AAD)

The AAD is constructed from the MPDU header. The AAD does not include the header Duration field, because the Duration field value might change due to normal IEEE 802.11 operation (e.g., a rate change during retransmission). The AAD includes neither the Duration/ID field nor the HT Control field because the contents of these fields might change during normal operation (e.g., due to a rate change preceding retransmission). The HT Control field might also be inserted or removed during normal operation (e.g., retransmission of an A-MPDU where the original A-MPDU included an MRQ that has already generated a response). For similar reasons, several subfields in the Frame Control field are masked to 0. AAD construction is performed as follows:

- a. FC – MPDU Frame Control field, with
 - 1. Subtype bits (bits 4 5 6) in a Data MPDU masked to 0
 - 2. Retry bit (bit 11) masked to 0
 - 3. Power Management bit (bit 12) masked to 0
 - 4. More Data bit (bit 13) masked to 0
 - 5. Protected Frame bit (bit 14) always set to 1
 - 6. Order bit (bit 15) as follows: i. Masked to 0 in all data MPDUs containing a QoS Control field ii. Unmasked otherwise
- b. A1 – MPDU Address 1 field
- c. A2 – MPDU Address 2 field
- d. A3 – MPDU Address 3 field
- e. SC – MPDU Sequence Control field, with the Sequence Number subfield (bits 4–15 of the Sequence Control field) masked to 0. The Fragment Number subfield is not modified
- f. A4 – MPDU Address field, if present.

g. QC – QoS Control field, if present, a 2-octet field that includes the MSDU priority. The QC TID is used in the construction of the AAD. When both the STA and its peer have their SPP A-MSDU Capable fields equal to 1, bit 7 (the A-MSDU Present field) is used in the construction of the AAD. The remaining QC fields are masked to 0 for the AAD calculation (bits 4 to 6, bits 8 to 15, and bit 7 when either the STA or its peer has the SPP A-MSDU Capable field equal to 0).

Construct CCM nonce

The Nonce field occupies 13 octets, and its structure is shown in Figure 11-19. The structure of the Nonce Flags subfield of the Nonce field is shown in Figure 11-20



Figure 11-19—Nonce construction

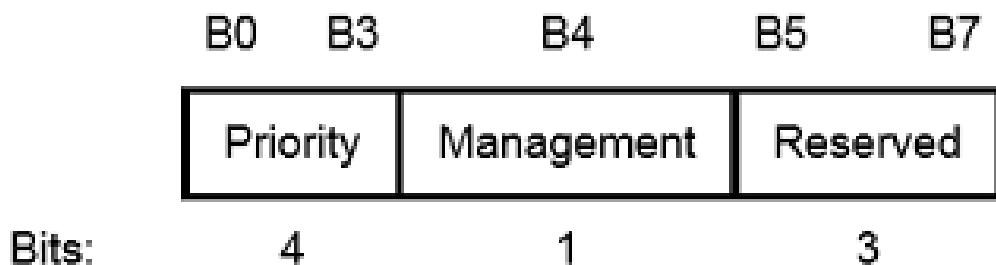


Figure 11-20—Nonce Flags subfield

Figure 7.141: NONCE Formatting for Wifi Security Protocol

The Nonce field has an internal structure of Nonce Flags || A2 || PN ("||" is concatenation), where

- The Priority subfield of the Nonce Flags field shall be set to the fixed value 0 when there is no QC field present in the MPDU header. When the QC field is present, bits 0 to 3 of the Priority subfield shall be set to the value of the QC TID (bits 0 to 3 of the QC field)
- When management frame protection is negotiated, the Management field of the Nonce Flags field shall be set to 1 if the Type field of the Frame Control field is 00 (Management frame); otherwise it is set to 0
- Bits 5 to 7 of the Nonce Flags field are reserved and shall be set to 0 on transmission
- MPDU address A2 field occupies octets 1–6. This shall be encoded with the octets ordered with A2 octet 0 at octet index 1 and A2 octet 5 at octet index 6

- The PN field occupies octets 7–12. The octets of PN shall be ordered so that PN0 is at octet index 12 and PN5 is at octet index 7.

Construct CCMP header

The format of the 8-octet CCMP header is given in 11.4.3.2. The header encodes the PN, Key ID, and ExtIV field values used to encrypt the MPDU.

CCM originator processing

CCM is a generic authenticate-and-encrypt block cipher mode, and in this standard, CCM is used with the AES block cipher. There are four inputs to CCM originator processing:

- a. Key: the temporal key (16 octets)
- b. Nonce: the nonce (13 octets) constructed as described in 11.4.3.3.4
- c. Frame body: the frame body of the MPDU
- d. AAD: the AAD (22–30 octets) constructed from the MPDU header as described in 11.4.3.3.3.

The CCM originator processing provides authentication and integrity of the frame body and the AAD as well as data confidentiality of the frame body. The output from the CCM originator processing consists of the encrypted data and 8 additional octets of encrypted MIC (see Figure 11-16). A CCMP protected individually addressed robust management frame shall be protected with the TK.

CCMP decapsulation

CCMP decrypts the payload of a cipher text MPDU and decapsulates a plaintext MPDU using the following steps:

- a. The encrypted MPDU is parsed to construct the AAD and nonce values
- b. The AAD is formed from the MPDU header of the encrypted MPDU
- c. The Nonce value is constructed from the A2, PN, and Nonce Flags fields
- d. The MIC is extracted for use in the CCM integrity checking
- e. The CCM recipient processing uses the temporal key, AAD, nonce, MIC, and MPDU cipher text data to recover the MPDU plaintext data as well as to check the integrity of the AAD and MPDU plaintext data
- f. The received MPDU header and the MPDU plaintext data from the CCM recipient processing are concatenated to form a plaintext MPDU
- g. The decryption processing prevents replay of MPDUs by validating that the PN in the MPDU is greater than the replay counter maintained for the session

When the received frame is a CCMP protected individually addressed robust management frame, contents of the MMPDU body after protection is removed shall be delivered to the SME via the MLME primitive designated for that management frame rather than through the MA-UNITDATA.indication primitive.

CCM recipient processing

CCM recipient processing uses the same parameters as CCM originator processing. A CCMP protected individually addressed robust management frame shall use the same TK as a Data MPDU.

Figure 11-21—CCMP decapsulation block diagram

There are four inputs to CCM recipient processing:

- Key: the temporal key (16 octets)
- Nonce: the nonce (13 octets) constructed as described in 11.4.3.3.4
- Encrypted frame body: the encrypted frame body from the received MPDU. The encrypted frame body includes an 8-octet MIC
- AAD: the AAD (22–30 octets) that is the canonical MPDU header as described in 11.4.3.3.3.

The CCM recipient processing checks the authentication and integrity of the frame body and the AAD as well as decrypting the frame body. The plaintext is returned only if the MIC check is successful. There is one output from error-free CCM recipient processing:

- Frame body: the plaintext frame body, which is 8 octets smaller than the encrypted frame body.

Decrypted CCMP MPDU

The decapsulation process succeeds when the calculated MIC matches the MIC value obtained from decrypting the received encrypted MPDU. The original MPDU header is concatenated with the plaintext data resulting from the successful CCM recipient processing to create the plaintext MPDU.

PN and replay detection

To effect replay detection, the receiver extracts the PN from the CCMP header. See 11.4.3.2 for a description of how the PN is encoded in the CCMP header. The following processing rules are used to detect replay:

- a. The PN values sequentially number each MPDU
- b. Each transmitter shall maintain a single PN (48-bit counter) for each PTKSA, GTKSA, and STKSA
- c. The PN shall be implemented as a 48-bit monotonically incrementing non-negative integer, initialized to 1 when the corresponding temporal key is initialized or refreshed
- d. A receiver shall maintain a separate set of PN replay counters for each PTKSA, GTKSA, and STKSA. The receiver initializes these replay counters to 0 when it resets the temporal key for a peer. The replay counter is set to the PN value of accepted CCMP MPDUs
- e. For each PTKSA, GTKSA, and STKSA, the recipient shall maintain a separate replay counter for each IEEE 802.11 MSDU or A-MSDU priority and shall use the PN recovered from a received frame to detect replayed frames, subject to the limitation of the number of supported replay counters indicated in the RSN Capabilities field (see 8.4.2.27). A replayed frame occurs when the PN extracted from a received frame is less than or equal to the current replay counter value for the frame's MSDU or A-MSDU priority and frame type. A transmitter shall not use IEEE 802.11 MSDU or A-MSDU priorities without ensuring that the receiver supports the required number of replay counters. The transmitter shall not reorder frames within a replay counter, but may reorder frames across replay counters. One possible reason for reordering frames is the IEEE 802.11 MSDU or A-MSDU priority
- f. If `dot11RSNAProtectedManagementFramesActivated` is true, the recipient shall maintain a single replay counter for received individually addressed robust management frames and shall use the PN from the received frame to detect replays. A replayed frame occurs when the PN from the frame is less than or equal to the current management frame replay counter value. The transmitter shall preserve the order of protected robust management frames sent to the same DA
- g. The receiver shall discard MSDUs, A-MSDUs, and MMPDUs whose constituent MPDU PN values are not sequential. A receiver shall discard any MPDU that is received with its PN less than or equal to the replay counter. When discarding a frame, the receiver shall increment by 1 the value of `dot11RSNASTatsCCMPReplays` for data frames or `dot11RSNASTatsRobustMgmtCCMPReplays` for robust management frames.
- h. For MSDUs or A-MSDUs sent using the Block Ack feature, reordering of received MSDUs or AMSDUs according to the Block Ack receiver operation (described in 9.21.4) is performed prior to replay detection

7.138.3 Broadcast/multicast integrity protocol (BIP)

BIP provides data integrity and replay protection for group addressed robust Management frames after successful establishment of an IGTKSA (see 12.6.1.1.9). BIP-CMAC-128 provides data integrity and replay protection, using AES-128 in CMAC Mode with a 128-bit integrity key and a CMAC TLen value of 128 (16 octets). BIP-CMAC-256 provides data integrity and replay protection, using AES-256 in CMAC Mode with a 256-bit integrity key and a CMAC TLen value of 128 (16 octets). NIST Special Publication 800-38B defines the CMAC algorithm, and NIST SP 800-38D defines the GMAC algorithm. BIP processing uses AES with a 128-bit or 256-bit integrity key and a CMAC TLen value of 128 (16 octets). The CMAC output for BIP-CMAC-256 is not truncated and shall be 128 bits (16 octets). The CMAC output for BIP-CMAC-128 is truncated to 64 bits:

- MIC = Truncate-64(CMAC Output)

BIP-GCMP-128 uses AES with a 128-bit integrity key, and BIP-GCMP-256 uses AES with a 256-bit integrity key. The authentication tag for both BIP-GCMP-128 and BIP-GCMP-256 is not truncated and shall be 128 bits (16 octets)

BIP uses the IGTK to compute the MMPDU MIC. The authenticator shall distribute one new IGTK and IGTK PN (IPN) whenever it distributes a new GTK. The IGTK is identified by the MAC address of the transmitting STA plus an IGTK identifier that is encoded in the MME Key ID field

BIP MMPDU format

The Management MIC element shall follow all of the other elements in the management frame body but precede the FCS. See 9.4.2.55 for the format of the Management MIC element. Figure 12-22 shows the BIP MMPDU



Figure 12-22—BIP Encapsulation

Figure 7.142: BIP Encapsulation

BIP AAD construction

The BIP Additional Authentication Data (AAD) shall be constructed from the MPDU header. The Duration field in the AAD shall be masked to 0. The AAD construction shall use a copy of the IEEE 802.11 header without the SC field for the MPDU, with the following exceptions:

a. FC—MPDU Frame Control field, with:

1. Retry subfield (bit 11) masked to 0
2. Power Management subfield (bit 12) masked to 0
3. More Data subfield (bit 13) masked to 0

b. A1—MPDU Address 1 field

c. A2—MPDU Address 2 field

d. A3—MPDU Address 3 field

Figure 12-23 depicts the format of the AAD. The length of the AAD is 20 octets

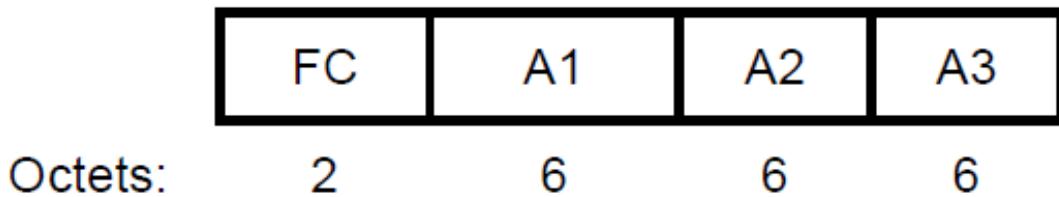


Figure 7.143: BIP AAD formatting

BIP replay protection

The MME Sequence Number field represents a sequence number whose length is 6 octets. When management frame protection is negotiated, the receiver shall maintain a 48-bit replay counter for each IGTK. The receiver shall set the receive replay counter to the value of the IPN in the IGTK key data encapsulation (KDE) (see 12.7.2) provided by the Authenticator in either the 4-way handshake, FT 4-way handshake, FT handshake, or group key handshake. The transmitter shall maintain a single IPN for each IGTK. The IPN shall be implemented as a 48-bit strictly increasing integer, initialized to 1 when the corresponding IGTK is initialized. The transmitter may reinitialize the sequence counter when the IGTK is refreshed. See 12.5.4.5 and 12.5.4.6 for per packet BIP processing.

NOTE—When the IPN space is exhausted, the choices available to an implementation are to replace the IGTK or to end communications

When dot11QMFActivated is true, the receiver shall maintain an additional replay counter for each ACI for received group addressed robust Management frames that use QMF. The receiver shall use the ACI encoded in the Sequence Number field of received GQMFs protected by BIP to select the replay counter to use for the received frame, and shall use the IPN from the received frame to detect replays

If dot11RSNAProtectedManagementFramesActivated is true and dot11MeshSecurityActivated is true, the recipient shall maintain a single replay counter for received group addressed robust Management frames that do not use the QMF service and shall use the PN from the received frame to detect replays. If dot11QMFActivated is also true, the recipient shall maintain an additional replay counter for each ACI for received group addressed robust Management frames that use the QMF service. The QMF receiver shall use the ACI encoded in the Sequence Number field of the received frame to select the replay counter to use for the received frame, and shall use the PN from the received frame to detect replays. A replayed frame occurs when the PN from the frame is less than or equal to the value of the management frame replay counter that corresponds to the ACI of the frame. The transmitter shall preserve the order of protected robust Management frames transmitted to the same DA without the QMF service. When the QMF service is used, the transmitter shall not reorder robust GQMFs within an AC when the frames are transmitted to the same RA

BIP transmission

When a STA transmits a protected group addressed robust Management frame, it shall

- a. Select the IGTK currently active for transmission of frames to the intended group of recipients and construct the MME (see 9.4.2.55) with the MIC field masked to 0 and the Key ID field set to the corresponding IGTK Key ID value. If the frame is not a GQMF, the transmitting STA shall insert a strictly increasing integer into the MME IPN field. If the frame is a GQMF, then the transmitting STA shall maintain a 48-bit counter for use as the IPN, the counter shall be incremented for each GQMF until the two least significant bits of the counter match the ACI of the AC that is used to transmit the frame, and the counter value shall be inserted into the MME IPN field of the frame. For BIP-GMAC-128 and BIP-GMAC-256, the initialization vector passed to GMAC shall be a concatenation of Address 2 from the MAC header of the MPDU and the non-negative integer inserted into the MMP IPN field
- b. Compute AAD as specified in 12.5.4.3
- c. Compute an integrity value over the concatenation of AAD and the management frame body including MME, and insert the output into the MME MIC field. For BIP-CMAC-128, the integrity value is 64 bits and is computed using AES-128-CMAC; for BIP-CMAC-256, the integrity value is 128 bits and is computed using AES-256-CMAC; for BIP-GMAC-128, the integrity value is 128 bits and is computed using AES-128-GMAC; and, for BIP-GMAC-256, the integrity value is 128 bits and is computed using AES-256-GMAC.
- d. Compose the frame as the IEEE 802.11 header, management frame body, including MME, and FCS. The MME shall appear last in the frame body
- e. Transmit the frame

BIP reception

When a STA with management frame protection negotiated receives a group addressed robust Management frame protected by BIP-CMAC-128, BIP-CMAC-256, BIP-GMAC-128, or BIP-GMAC-256, it shall

- a. Identify the appropriate IGTK and associated state based on the MME Key ID field. If no such IGTK exists, silently drop the frame and terminate BIP processing for this reception
- b. Perform replay protection on the received frame. The receiver shall interpret the MME IPN field as a 48-bit unsigned integer
 1. If the frame is not a GQMF, the receiver shall compare this MME IPN integer value to the value of the receive replay counter for the IGTK identified by the MME Key ID field. If the integer value from the received MME IPN field is less than or equal to the replay counter value for this IGTK, the receiver shall discard the frame and increment the dot11RSNAStatsCMACReplays counter by 1
 2. If the frame is a GQMF, the receiver shall compare this MME IPN integer value to the value of the receive replay counter for the IGTK identified by the MME Key ID field and the AC represented by the value of the ACI subfield of the received frame. If the integer value from the received MME IPN field is less than or equal

to the replay counter value for this IGTK and AC, the receiver shall discard the frame and increment the dot11RSNAStatsCMACReplays counter by 1

c. Compute AAD for this Management frame, as specified in 12.5.4.3. For BIP-GMAC-128 and BIPGMAC-256, an initialization vector for GMAC is constructed as the concatenation of Address 2 from the MAC header of the MPDU and the 48-bit unsigned integer from the MME IPN field

d. Extract and save the received MIC value, and compute a verifier over the concatenation of AAD, the management frame body, and MME, with the MIC field masked to 0 in the MME. For BIP-CMAC-128, the integrity value is 64 bits and is computed using AES-128-CMAC; for BIP-CMAC-256, the integrity value is 128 bits and is computed using AES-256-CMAC; for BIP-GMAC-128, the integrity value is 128 bits and is computed using AES-128-GMAC; and, for BIP-GMAC-256, the integrity value is 128 bits and is computed using AES-256-GMAC. If the result does not match the received MIC value, then the receiver shall discard the frame, increment the dot11RSNAStatsBIPMICErrors counter by 1, and terminate BIP processing for this reception

e. If the frame is not a GQMF, update the replay counter for the IGTK identified by the MME Key ID field with the integer value of the MME IPN field

f. If the frame is a GQMF, update the replay counter for the IGTK identified by the MME Key ID field and the AC represented by the value of the ACI subfield of the received frame with the integer value of the MME IPN field

If management frame protection is negotiated, group addressed robust Management frames that are received without BIP protection shall be discarded

7.138.4 AES-GCM Use in Wifi

IEEE-802.11-2016 specifies the GCMP, which provides data confidentiality, authentication, integrity, and replay protection. A DMG RSNA STA shall support GCMP-128. GCMP is based on the GCM of the AES encryption algorithm. GCM protects the integrity of both the MPDU Data field and selected portions of the MPDU header. The AES algorithm is defined in FIPS PUB 197. All AES processing used within GCMP uses AES with a 128-bit key (GCMP-128) or a 256-bit key (GCMP-256). GCM is defined in NIST Special Publication 800-38D. GCM is a generic mode that can be used with any block-oriented encryption algorithm

GCM requires a fresh temporal key for every session. GCM also requires a unique nonce value for each frame protected by a given temporal key, and GCMP uses a 96-bit nonce that includes a 48-bit packet number (PN) for this purpose. Reuse of a PN with the same temporal key voids all security guarantees. GCMP uses a 128-bit MIC

When GCMP is selected as the RSN pairwise cipher and management frame protection is negotiated, individually addressed robust Management frames and the group addressed Management frames that receive "Group Addressed Privacy" as indicated in Table 9-47 shall be protected with GCMP

GCMP MPDU format

Figure 12-24 depicts the MPDU when using GCMP

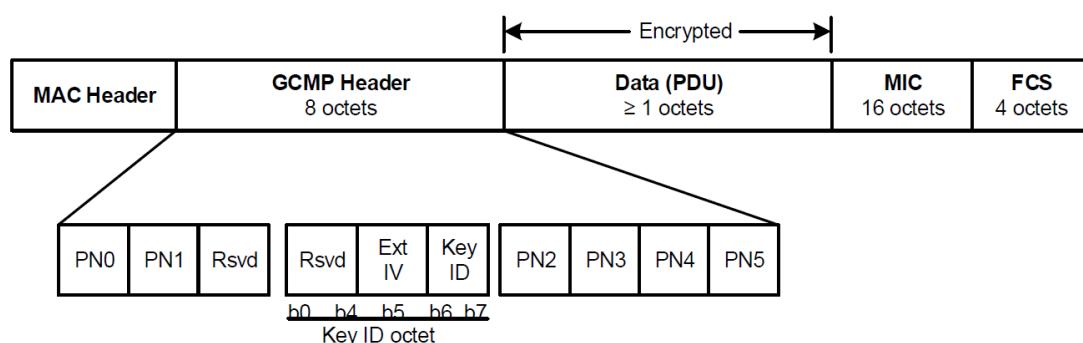


Figure 7.144: Wifi Packet Formatting For GCMP Encapsulation

GCMP cryptographic encapsulation

GCMP encrypts the Frame Body field of a plaintext MPDU and encapsulates the resulting cipher text using the following steps:

- a. Increment the PN, to obtain a fresh PN for each MPDU, so that the PN never repeats for the same temporal key
- b. Use the fields in the MPDU header to construct the additional authentication data (AAD) for GCM. The GCM algorithm provides integrity protection for the fields included in the AAD. MPDU header fields that may change when retransmitted are masked to 0 when calculating the AAD
- c. Construct the GCM Nonce block from the PN and A2, where A2 is MPDU Address 2
- d. Place the new PN and the key identifier into the 8-octet GCMP Header
- e. Use the temporal key, AAD, nonce, and MPDU data to form the cipher text and MIC. This step is known as GCM originator processing
- f. Form the encrypted MPDU by combining the original MPDU header, the GCMP header, the encrypted data and MIC

GCMP PN processing

The PN is incremented by a positive number for each MPDU. The PN shall be incremented in steps of 1 for constituent MPDUs of fragmented MSDUs and MMPDUs. The PN shall never repeat for a series of encrypted MPDUs using the same temporal key. If the PN is larger than dot11PNEhaustionThreshold, an MLME-PN-EXHAUSTION-indication primitive shall be generated

Construct AAD

The AAD for GCMP is constructed in the same manner as CCMP

Construct GCM nonce

The Nonce field occupies 12 octets, and its structure is shown in Figure 12-26

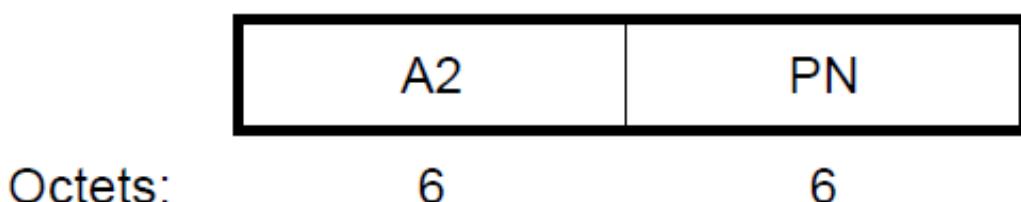


Figure 7.145: NONCE Formatting of Wifi GCM Security Packets

The Nonce field has an internal structure of A2 || PN, where

- MPDU address A2 field occupies octets 0 to 5. This shall be encoded with the octets ordered with A2 octet 0 at octet index 0 and A2 octet 5 at octet index 5
- The PN field occupies octets 6 to 11. The octets of PN shall be ordered so that PN0 is at octet index 11 and PN5 is at octet index 6

GCM originator processing

GCM is a generic authenticate-and-encrypt block cipher mode, and in this standard, GCM is used with the AES block cipher

There are four inputs to GCM originator processing:

- a. Key: the temporal key (16 octets)
- b.Nonce: the nonce (12 octets) constructed as described in 12.5.5.3.4
- c. Frame body: the plaintext frame body of the MPDU

d. AAD: the AAD (22-30 octets) constructed from the MPDU header as described in 12.5.5.3.3

The GCM originator processing provides authentication and integrity of the frame body and the AAD as well as data confidentiality of the frame body. The output from the GCM originator processing consists of the encrypted data and 16 additional octets of encrypted MIC (see Figure 12-24). The PN values sequentially number each MPDU. Each transmitter shall maintain a single PN (48-bit counter) for each PTKSA, GTKSA, and STKSA. The PN shall be implemented as a 48-bit strictly increasing integer, initialized to 1 when the corresponding temporal key is initialized or refreshed

A transmitter shall not use IEEE 802.11 MSDU or A-MSDU priorities without ensuring that the receiver supports the required number of replay counters. The transmitter shall not reorder GCMP protected frames that are transmitted to the same RA within a replay counter, but may reorder frames across replay counters

One possible reason for reordering frames is the IEEE 802.11 MSDU or A-MSDU priority. The transmitter shall preserve the order of protected robust Management frames that are transmitted to the same DA without the QMF service. When the QMF service is used, the transmitter shall not reorder robust IQMFs within an AC when the frames are transmitted to the same RA

A GCMP protected individually addressed robust Management frame shall be protected using the same TK as a Data frame

GCMP decapsulation

GCMP decrypts the Frame Body field of a cipher text MPDU and decapsulates a plaintext MPDU using the following steps:

- a. The encrypted MPDU is parsed to construct the AAD and nonce values
- b. The AAD is formed from the MPDU header of the encrypted MPDU
- c. The Nonce value is constructed from the A2 and PN fields
- d. The MIC is extracted for use in the GCM integrity checking
- e. The GCM recipient processing uses the temporal key, AAD, nonce, MIC, and MPDU cipher text data to recover the MPDU plaintext data as well as to check the integrity of the AAD and MPDU plaintext data
- f. The received MPDU header and the MPDU plaintext data from the GCM recipient processing are concatenated to form a plaintext MPDU
- g. The decryption processing prevents replay of MPDUs by validating that the PN in the MPDU is greater than the replay counter maintained for the session

When the received frame is a GCMP protected individually addressed robust Management frame, the contents of the MMPDU body after protection is removed and shall be delivered to the SME via the MLME primitive designated for that MMPDU rather than through the MA-UNITDATA.indication primitive

GCM recipient processing

GCM recipient processing shall use the same parameters as GCM originator processing. A GCMP protected individually addressed robust Management frame shall use the same TK as a Data frame

There are four inputs to GCM recipient processing:

- Key: the temporal key (16 octets)
- Nonce: the nonce (12 octets) constructed as described in 12.5.5.4
- Encrypted frame body: the encrypted frame body from the received MPDU. The encrypted frame body includes a 16-octet MIC
- AAD: the AAD (22-30 octets) that is the canonical MPDU header as described in 12.5.5.3.3

The GCM recipient processing checks the authentication and integrity of the frame body and the AAD as well as decrypting the frame body. The plaintext is returned only if the MIC check is successful

There is one output from error-free GCM recipient processing:

- Frame body: the plaintext frame body, which is 16 octets smaller than the encrypted frame body

Decrypted GCMP MPDU

The decapsulation process succeeds when the calculated MIC matches the MIC value obtained from decrypting the received encrypted MPDU. The original MPDU header is concatenated with the plaintext data resulting from the successful GCM recipient processing to create the plaintext MPDU

PN and replay detection

To effect replay detection, the receiver extracts the PN from the GCMP header. See 12.5.5.2 for a description of how the PN is encoded in the GCMP header. The following processing rules are used to detect replay:

- a. The receiver shall maintain a separate set of replay counters for each PTKSA, GTKSA, and STKSA. The receiver initializes these replay counters to 0 when it resets the temporal key for a peer. The replay counter is set to the PN value of accepted GCMP MPDUs
- b. For each PTKSA, GTKSA, and STKSA, the recipient shall maintain a separate replay counter for each TID, subject to the limitation of the number of supported replay counters indicated in the RSN Capabilities field (see 9.4.2.25), and shall use the PN from a received frame to detect replayed frames. A replayed frame occurs when the PN from a received frame is less than or equal to the current replay counter value for the frame's MSDU or A-MSDU priority and frame type
- c. If dot11RSNAProtectedManagementFramesActivated is true, the recipient shall maintain a single replay counter for received individually addressed robust Management frames that are received with the To DS subfield equal to 0 and shall use the PN from the received frame to detect replays. If dot11QMFActivated is also true, the recipient shall maintain an additional replay counter for each ACI for received individually addressed robust Management frames that are received with the To DS subfield equal to 1. The QMF receiver shall use the ACI encoded in the Sequence Number field of the received frame to select the replay counter to use for the received frame, and shall use the PN from the received frame to detect replays. A replayed frame occurs when the PN from the frame is less than or equal to the current value of the management frame replay counter that corresponds to the ACI of the frame
- d. The receiver shall discard any Data frame that is received with its PN less than or equal to the value of the replay counter that is associated with the TA and priority value of the received MPDU. The receiver shall discard MSDUs and MMPDUs whose constituent MPDU PN values are not incrementing in steps of 1. If dot11RSNAProtectedManagementFramesActivated is true, the receiver shall discard any individually addressed robust Management frame that is received with its PN less than or equal to the value of the replay counter associated with the TA of that individually addressed Management frame
- e. When discarding a frame, the receiver shall increment by 1 dot11RSNAStatsGCMPReplays for Data frames or dot11RSNAStatsRobustMgmtGCMPReplays for robust Management frames
- f. For MSDUs or A-MSDUs sent using the block ack feature, reordering of received MSDUs or A-MSDUs according to the block ack receiver operation (described in 10.24.4) is performed prior to replay detection

MAC frame formats

Each frame consists of the following basic components:

- a. A MAC header, which comprises frame control, duration, address, optional sequence control information, optional QoS Control information (QoS data frames only), and optional HT Control fields (+HTC frames only)
- b. A variable-length frame body, which contains information specific to the frame type and subtype
- c. A FCS, which contains an IEEE 32-bit CRC

Conventions

Structures defined in the MAC sublayer are described as a sequence of fields in specific order. Each figure in Clause 8 depicts the fields/subfields as they appear in the MAC frame and in the order in which they are passed to the physical layer convergence procedure (PLCP), from left to right.

In figures, all bits within fields are numbered, from 0 to k, where the length of the field is k + 1 bits. Bits within numeric fields that are longer than a single bit are depicted in increasing order of significance, i.e., with the lowest numbered bit having the least significance. The octet boundaries within a field can be obtained by taking the bit numbers of the field modulo 8. Octets within numeric fields that are longer than a single octet are depicted in increasing order of significance, from lowest numbered bit to highest numbered bit. The octets in fields longer than a single octet are sent to the PLCP in order from the octet containing the lowest numbered bits to the octet containing the highest numbered bits.

Any field containing a CRC is an exception to this convention and is transmitted commencing with the coefficient of the highest-order term.

MAC addresses are assigned as ordered sequences of bits. The Individual/Group bit is always transferred first and is bit 0 of the first octet.

Organizationally unique identifiers (OUIs) and Organization Identifiers are specified in two forms: an ordered sequence of octets, and a numeric form. Treating the OUI or Organization Identifier as an ordered sequence of octets, the leftmost octet is always transferred first. This is equivalent to transmitting the most significant octet of the numeric form first.

Values specified in decimal are coded in natural binary unless otherwise stated. The values in Table8-1 are in binary, with the bit assignments shown in the table. Values in other tables are shown in decimal notation.

Reception, in references to frames or fields within frames (e.g., received Beacon frames or a received Duration-/ID field), applies to MPDUs or MAC management protocol data units (MMPDUs) indicated from the PHY layer without error and validated by FCS within the MAC sublayer. Without further qualification, reception by the MAC sublayer implies that the frame contents are valid, and that the protocol version is supported (see 8.2.4.1.2), with no implication regarding frame addressing or regarding whether the frame type or other fields in the MAC header are meaningful to the MAC entity that has received the frame.

A frame that contains the HT Control field, including the Control Wrapper frame, is referred to as a +HTC frame. A QoS Data frame that is transmitted by a mesh STA is referred to as a Mesh Data frame.

Parentheses enclosing portions of names or acronyms are used to designate a set of related names that vary based on the inclusion of the parenthesized portion. For example,

- QoS +CF-Poll frame refers to the three QoS data subtypes that include “+CF-Poll”: the QoS Data+CF-Poll frame, subtype 1010; QoS Data+CF-Ack+CF-Poll frame, subtype 1011; and QoS CF-Ack+CF-Poll frame, subtype 1111
- QoS CF-Poll frame refers specifically to the QoS CF-Poll frame, subtype 1110
- QoS (+)CF-Poll frame refers to all four QoS data subtypes with CF-Poll: the QoS CF-Poll frame, subtype 1110; the QoS CF-Ack+CF-Poll frame, subtype 1111; the QoS Data+CF-Poll frame, subtype 1010; and the QoS Data+CF-Ack+CF-Poll frame, subtype 1011
- QoS (+)Null frame refers to all three QoS data subtypes with “no data”: the QoS Null (no data) frame, subtype 1100; the QoS CF-Poll (no data) frame, subtype 1110; and the QoS CF-Ack+CF-Poll frame, subtype 1111
- QoS +CF-Ack frame refers to the three QoS data subtypes that include “+CF-Ack”: the QoS Data+CF-Ack frame, subtype 1001; QoS Data+CF-Ack+CF-Poll frame, subtype 1011; and QoS CF-Ack+CF-Poll frame, subtype 1111
- Whereas (QoS) CF-Poll frame refers to the QoS CF-Poll frame, subtype 1110, and the CF-Poll frame, subtype 0110.

Reserved fields and subfields are set to 0 upon transmission and are ignored upon reception.

General frame format

The MAC frame format comprises a set of fields that occur in a fixed order in all frames. Figure8-1 depicts the general MAC frame format. The first three fields (Frame Control, Duration/ID, and Address 1) and the last field (FCS) in Figure8-1 constitute the minimal frame format and are present in all frames, including reserved types and subtypes. The fields Address 2, Address 3, Sequence Control, Address 4, QoS Control, HT Control, and Frame Body are present only in certain frame types and subtypes. Each field is defined in 8.2.4. The format of each of the individual subtypes of each frame type is defined in 8.3. The components of management frame bodies are defined in 8.4. The formats of management frames of subtype Action are defined in 8.5.

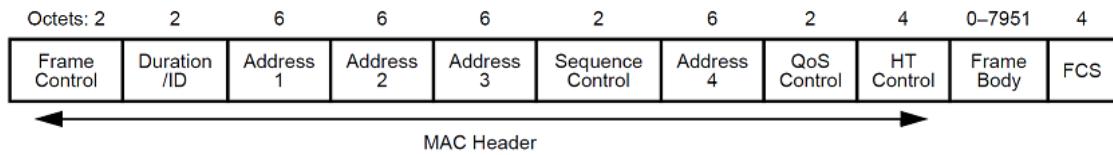


Figure 8-1—MAC frame format

Figure 7.146: General Frame Format for Wifi Packets

The Frame Body field is of variable size. The maximum frame body size is determined by the maximum MSDU size (2304 octets), plus the length of the Mesh Control field (6, 12, or 18 octets) if present, the maximum unencrypted MMPDU size excluding the MAC header and FCS (2304 octets) or the maximum AMSDU size (3839 or 7935 octets, depending upon the STA's capability), plus any overhead from security encapsulation.

Frame fields

The Frame Control field consists of the following subfields: Protocol Version, Type, Subtype, To DS, FromDS, More Fragments, Retry, Power Management, More Data, Protected Frame, and Order. The format of the Frame Control field is illustrated in Figure 8-2.

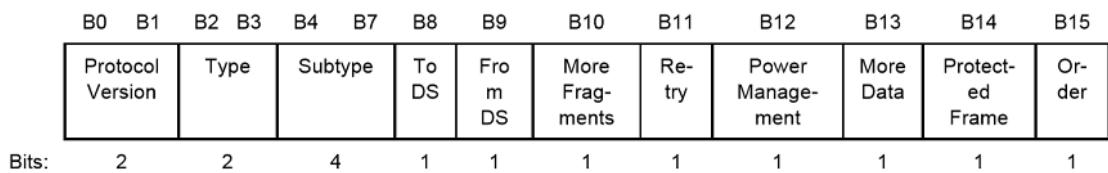


Figure 8-2—Frame Control field

Figure 7.147: Frame Control Field Format for Wifi Packets

Protocol Version field

The Protocol Version field is 2 bits in length and is invariant in size and placement across all revisions of this standard. For this standard, the value of the protocol version is 0. All other values are reserved. The revision level will be incremented only when a fundamental incompatibility exists between a new revision and the prior edition of the standard. See 9.24.2.

Type and Subtype fields

The Type field is 2 bits in length, and the Subtype field is 4 bits in length. The Type and Subtype fields together identify the function of the frame. There are three frame types: control, data, and management. Each of the frame types has several defined subtypes. In data frames, the most significant bit (MSB) of the Subtype field, b7, is defined as the QoS subfield. Table 8-1 defines the valid combinations of type and subtype. (The numeric values in Table 8-1 are shown in binary, see IEEE802.11-2012)

Each Subtype field bit position is used to indicate a specific modification of the basic data frame (subtype 0). Frame Control bit 4 is set to 1 in data subtypes that include +CF-Ack, bit 5 is set to 1 in data subtypes that include +CF-Poll, bit 6 is set to 1 in data subtypes that contain no Frame Body field, and bit 7 is set to 1 in the QoS data subtypes, which have QoS Control fields in their MAC headers.

Table 8-1—Valid type and subtype combinations

Type value b3 b2	Type description	Subtype value b7 b6 b5 b4	Subtype description
00	Management	0000	Association request
00	Management	0001	Association response
00	Management	0010	Reassociation request
00	Management	0011	Reassociation response
00	Management	0100	Probe request
00	Management	0101	Probe response
00	Management	0110	Timing Advertisement

Table 8-1—Valid type and subtype combinations (continued)

Type value b3 b2	Type description	Subtype value b7 b6 b5 b4	Subtype description
00	Management	0111	Reserved
00	Management	1000	Beacon
00	Management	1001	ATIM
00	Management	1010	Disassociation
00	Management	1011	Authentication
00	Management	1100	Deauthentication
00	Management	1101	Action
00	Management	1110	Action No Ack
00	Management	1111	Reserved
01	Control	0000–0110	Reserved
01	Control	0111	Control Wrapper
01	Control	1000	Block Ack Request (BlockAckReq)
01	Control	1001	Block Ack (BlockAck)
01	Control	1010	PS-Poll
01	Control	1011	RTS
01	Control	1100	CTS
01	Control	1101	ACK
01	Control	1110	CF-End
01	Control	1111	CF-End + CF-Ack
10	Data	0000	Data
10	Data	0001	Data + CF-Ack
10	Data	0010	Data + CF-Poll
10	Data	0011	Data + CF-Ack + CF-Poll
10	Data	0100	Null (no data)
10	Data	0101	CF-Ack (no data)
10	Data	0110	CF-Poll (no data)
10	Data	0111	CF-Ack + CF-Poll (no data)
10	Data	1000	QoS Data
10	Data	1001	QoS Data + CF-Ack
10	Data	1010	QoS Data + CF-Poll
10	Data	1011	QoS Data + CF-Ack + CF-Poll
10	Data	1100	QoS Null (no data)
10	Data	1101	Reserved
10	Data	1110	QoS CF-Poll (no data)
10	Data	1111	QoS CF-Ack + CF-Poll (no data)
11	Reserved	0000–1111	Reserved

Figure 7.148: Wifi Type and SubType Values
 Copyright 2017-2019 John Peter Greninger : Generated on Sat Aug 31 2019 22:28:11 for Protocol++ (ProtocolPP) 3.0.1 by Doxygen

To DS and From DS fields

The meaning of the combinations of values for the To DS and From DS fields are shown in Table8-2

Table 8-2—To/From DS combinations in data frames

To DS and From DS values	Meaning
To DS = 0 From DS = 0	A data frame direct from one STA to another STA within the same IBSS, a data frame direct from one non-AP STA to another non-AP STA within the same BSS, or a data frame outside the context of a BSS, as well as all management and control frames.
To DS = 1 From DS = 0	A data frame destined for the DS or being sent by a STA associated with an AP to the Port Access Entity in that AP.
To DS = 0 From DS = 1	A data frame exiting the DS or being sent by the Port Access Entity in an AP, or a group addressed Mesh Data frame with Mesh Control field present using the three-address MAC header format.
To DS = 1 From DS = 1	A data frame using the four-address MAC header format. This standard defines procedures for using this combination of field values only in a mesh BSS.

Figure 7.149: Wifi To and From Data Combinations

More Fragments field

The More Fragments field is 1 bit in length and is set to 1 in all data or management type frames that have another fragment of the current MSDU or current MMPDU to follow. It is set to 0 in all other frames.

Retry field

The Retry field is 1 bit in length and is set to 1 in any data or management type frame that is a retransmission of an earlier frame. It is set to 0 in all other frames. A receiving STA uses this indication to aid in the process of eliminating duplicate frames.

Power Management field

The Power Management field is 1 bit in length and is used to indicate the power management mode of a STA. The value of this field is either reserved (as defined below) or remains constant in each frame from a particular STA within a frame exchange sequence (see AnnexG). The value indicates the mode of the STA after the successful completion of the frame exchange sequence.

In an infrastructure BSS, the following applies:

- The Power Management field is reserved in all management frames that are not bufferable management frames
- The Power Management field is reserved in all management frames transmitted by a STA to an AP with which it is not associated
- The Power Management field is reserved in all frames transmitted by the AP
- Otherwise, a value of 1 indicates that the STA will be in PS mode. A value of 0 indicates that the STA will be in active mode.

In an IBSS, the following applies:

- The Power Management field is reserved in all management frames that are not bufferable management frames and that are not individually addressed Probe Request frames
- Otherwise, a value of 1 indicates that the STA will be in PS mode. A value of 0 indicates that the STA will be in active mode.

In an MBSS, the following applies:

- A value of 0 in group addressed frames, in management frames transmitted to nonpeer STAs, and in Probe Response frames indicates that the mesh STA will be in active mode towards all neighbor mesh STAs. A value of 1 in group addressed frames, in management frames transmitted to nonpeer STAs, and in Probe Response frames indicates that the mesh STA will be in deep sleep mode towards all nonpeer mesh power STAs
- A value of 0 in individually addressed frames transmitted to a peer mesh STA indicates that the mesh STA will be in active mode towards this peer mesh STA. A value of 1 in individually addressed frames transmitted to a peer mesh STA, except Probe Response frames, indicates that the mesh STA will be in either light sleep mode or deep sleep mode towards this peer mesh STA. When the QoS Control field is present in the frame, the Mesh Power Save Level subfield in the QoS Control field indicates whether the mesh STA will be in light sleep mode or in deep sleep mode for the recipient mesh STA as specified in 8.2.4.5.11.

The mesh power mode transition rules are described in 13.14.3.

More Data field

The More Data field is 1 bit in length and is used to indicate to a STA in PS mode that more BUs are buffered for that STA at the AP. The More Data field is valid in individually addressed data or management type frames transmitted by an AP to a STA in PS mode. A value of 1 indicates that at least one additional buffered BU is present for the same STA.

The More Data field is optionally set to 1 in individually addressed data type frames transmitted by a CF Pollable STA to the PC in response to a CF-Poll to indicate that the STA has at least one additional buffered MSDU available for transmission in response to a subsequent CF-Poll.

For a STA in which the More Data Ack subfield of its QoS Capability element is 1 and that has APSD enabled, an AP optionally sets the More Data field to 1 in ACK frames to this STA to indicate that the AP has a pending transmission for the STA.

For a STA with TDLS peer PSM enabled and the More Data Ack subfield equal to 1 in the QoS Capability element of its transmitted TDLS Setup Request frame or TDLS Setup Response frame, a TDLS peer STA optionally sets the More Data field to 1 in ACK frames to this STA to indicate that it has a pending transmission for the STA.

The More Data field is 1 in individually addressed frames transmitted by a mesh STA to a peer mesh STA that is either in light sleep mode or in deep sleep mode for the corresponding mesh peering, when additional BUs remain to be transmitted to this peer mesh STA.

The More Data field is set to 0 in all other individually addressed frames.

The More Data field is set to 1 in group addressed frames transmitted by the AP when additional group addressed bufferable units (BUs) remain to be transmitted by the AP during this beacon interval. The More Data field is set to 0 in group addressed frames transmitted by the AP when no more group addressed BUs remain to be transmitted by the AP during this beacon interval and in all group addressed frames transmitted by non-AP STAs.

The More Data field is 1 in group addressed frames transmitted by a mesh STA when additional group addressed BUs remain to be transmitted. The More Data field is 0 in group addressed frames transmitted by a mesh STA when no more group addressed BUs remain to be transmitted.

Protected Frame field

The Protected Frame field is 1 bit in length. The Protected Frame field is set to 1 if the Frame Body field contains information that has been processed by a cryptographic encapsulation algorithm. The Protected Frame field is set to 1 only within data frames and within management frames of subtype Authentication, and individually addressed robust management frames. The Protected Frame field is set to 0 in all other frames. When the Protected Frame field is equal to 1, the Frame Body field is protected utilizing the cryptographic encapsulation algorithm and expanded as defined in Clause 11. The Protected Frame field is set to 0 in Data frames of subtype Null Function, CF-ACK (no data), CF-Poll (no data), CF-ACK+CF-Poll (no data), QoS Null (no data), QoS CF-Poll (no data), and QoS CF-ACK+CF-Poll (no data) (see, for example, 11.4.2.2 and 11.4.3.1 that show that the frame body needs to be 1 octet or longer to apply the encapsulation).

Order field

The Order field is 1 bit in length. It is used for two purposes:

- It is set to 1 in a non-QoS data frame transmitted by a non-QoS STA to indicate that the frame contains an MSDU, or fragment thereof, that is being transferred using the StrictlyOrdered service class
- It is set to 1 in a QoS data or management frame transmitted with a value of HT_GF or HT_MF for the FO-RMAT parameter of the TXVECTOR to indicate that the frame contains an HT Control field. Otherwise, the Order field is set to 0.

Duration/ID field

The Duration/ID field is 16 bits in length. The contents of this field vary with frame type and subtype, with whether the frame is transmitted during the CFP, and with the QoS capabilities of the sending STA. The contents of the field are defined as follows:

- In control frames of subtype PS-Poll, the Duration/ID field carries the association identifier (AID) of the STA that transmitted the frame in the 14 least significant bits (LSB), and the 2 most significant bits (MSB) both set to 1. The value of the AID is in the range 1–2007
- In frames transmitted by the PC and non-QoS STAs, during the CFP, the Duration/ID field is set to a fixed value of 32768
- In all other frames sent by non-QoS STAs and control frames sent by QoS STAs, the Duration/ID field contains a duration value as defined for each frame type in 8.3
- In data and management frames sent by QoS STAs, the Duration/ID field contains a duration value as defined for each frame type in 8.2.5.

See 9.24.3 on the processing of this field in received frames.

The encoding of the Duration/ID field is given in Table 8-3.

Table 8-3—Duration/ID field encoding

Bits 0–13	Bit 14	Bit 15	Usage
0–32 767		0	Duration value (in microseconds) within all frames other than PS-Poll frames transmitted during the CP, and under HCF for frames transmitted during the CFP
0	0	1	Fixed value under point coordination function (PCF) within frames transmitted during the CFP
1–16 383	0	1	Reserved
0	1	1	Reserved
1–2007	1	1	AID in PS-Poll frames
2008–16 383	1	1	Reserved

Figure 7.150: Wifi Duration or Identification Field Formatting

The Duration/ID fields in the MAC headers of MPDUs in an A-MPDU all carry the same value NOTE—The reference point for the Duration/ID field is the end of the PPDU carrying the MPDU Setting the Duration/ ID field to the same value in the case of A-MPDU aggregation means that each MPDU consistently specifies the same NAV setting.

Address fields

There are four address fields in the MAC frame format. These fields are used to indicate the basic service set identifier (BSSID), source address (SA), destination address (DA), transmitting STA address (TA), and receiving STA address (RA). Certain frames may not contain some of the address fields.

Certain address field usage is specified by the relative position of the address field (1–4) within the MAC header, independent of the type of address present in that field. For example, receiver address matching is always performed on the contents of the Address 1 field in received frames, and the receiver address of CTS and ACK frames is always obtained from the Address 2 field in the corresponding RTS frame, or from the frame being acknowledged.

Address representation

Each Address field contains a 48-bit address as defined in 9.2 of IEEE Std 802-2001

Address designation

A MAC sublayer address is one of the following two types:

- a. Individual address. The address assigned to a particular STA on the network
- b. Group address. A multidestination address, which may be in use by one or more STAs on a given network. The two kinds of group addresses are as follows:
 - 1. Multicast-group address. An address associated by higher level convention with a group of logically related STAs
 - 2. Broadcast address. A distinguished, predefined group address that always denotes the set of all STAs on a given LAN. All ones are interpreted to be the broadcast address. This group is predefined for each communication medium to consist of all STAs actively connected to that medium; it is used to broadcast to all the active STAs on that medium.

The address space is also partitioned into locally administered and universal (globally administered) addresses. The nature of a body and the procedures by which it administers these universal (globally administered) addresses is beyond the scope of this standard. See IEEE Std 802-2001 for more information

Sequence Control field

The Sequence Control field is 16 bits in length and consists of two subfields, the Sequence Number and the Fragment Number. The format of the Sequence Control field is illustrated in Figure 8-3. The sequence Control field is not present in control frames.

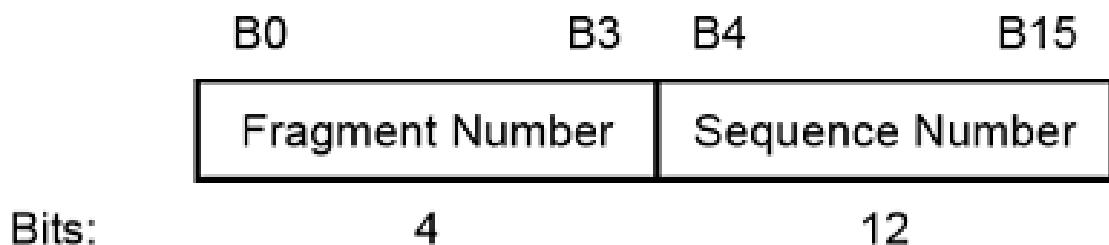


Figure 8-3—Sequence Control field

Figure 7.151: Wifi Sequence Control Field Formatting

Sequence Number field

The Sequence Number field is a 12-bit field indicating the sequence number of an MSDU, A-MSDU, or MMPDU. Each MSDU, A-MSDU, or MMPDU transmitted by a STA is assigned a sequence number. Sequence numbers are not assigned to control frames, as the Sequence Control field is not present.

Each fragment of an MSDU or MMPDU contains a copy of the sequence number assigned to that MSDU or MMPDU. The sequence number remains constant in all retransmissions of an MSDU, MMPDU, or fragment thereof.

Fragment Number field

The Fragment Number field is a 4-bit field indicating the number of each fragment of an MSDU or MMPDU. The fragment number is set to 0 in the first or only fragment of an MSDU or MMPDU and is incremented by one for each successive fragment of that MSDU or MMPDU. The fragment number is set to 0 in the only fragment of an A-MSDU. The fragment number remains constant in all retransmissions of the fragment.

QoS Control field

The QoS Control field is a 16-bit field that identifies the TC or TS to which the frame belongs as well as various other QoS-related, A-MSDU related, and mesh-related information about the frame that varies by frame type, subtype, and type of transmitting STA. The QoS Control field is present in all data frames in which the QoS subfield of the Subtype field is equal to 1 (see 8.2.4.1.3). Each QoS Control field comprises five or eight subfields, as defined for the particular sender (HC or non-AP STA) and frame type and subtype. The usage of these subfields and the various possible layouts of the QoS Control field are described 8.2.4.5.2 to 8.2.4.5.12 and illustrated in Table 8-4 of the IEEE802.11-2012 specification

Table 8-4—QoS Control field

Applicable frame (sub) types	Bits 0–3	Bit 4	Bits 5–6	Bit 7	Bits 8	Bit 9	Bit 10	Bits 11– 15
QoS CF-Poll and QoS CF-Ack+CF-Poll frames sent by HC	TID	EOSP	Ack Policy	Reserved	TXOP Limit			
QoS Data+CF-Poll and QoS Data+CF-Ack+CF-Poll frames sent by HC	TID	EOSP	Ack Policy	A-MSDU Present	TXOP Limit			
QoS Data and QoS Data+CF-Ack frames sent by HC	TID	EOSP	Ack Policy	A-MSDU Present	AP PS Buffer State			
QoS Null frames sent by HC	TID	EOSP	Ack Policy	Reserved	AP PS Buffer State			
QoS Data and QoS Data+CF-Ack frames sent by non-AP STAs that are not a TPU buffer STA or a TPU sleep STA in a nonmesh BSS	TID	0	Ack Policy	A-MSDU Present	TXOP Duration Requested			
	TID	1	Ack Policy	A-MSDU Present	Queue Size			

Table 8-4—QoS Control field (continued)

Applicable frame (sub) types	Bits 0–3	Bit 4	Bits 5–6	Bit 7	Bits 8	Bit 9	Bit 10	Bits 11– 15
QoS Null frames sent by non-AP STAs that are not a TPU buffer STA or a TPU sleep STA in a nonmesh BSS	TID	0	Ack Policy	Reserve d	TXOP Duration Requested			
	TID	1	Ack Policy	Reserve d	Queue Size			
QoS Data and QoS Data+CF-Ack frames sent by TPU buffer STAs in a nonmesh BSS	TID	EOSP	Ack Policy	A-MSDU Present	Reserved			
QoS Null frames sent by TPU buffer STAs in a nonmesh BSS	TID	EOSP	Ack Policy	Reserve d	Reserved			
QoS Data and QoS Data+CF-Ack frames sent by TPU sleep STAs in a nonmesh BSS	TID	Reserv ed	Ack Policy	A-MSDU Present	Reserved			
QoS Null frames sent by TPU sleep STAs in a nonmesh BSS	TID	Reserv ed	Ack Policy	Reserve d	Reserved			
All frames sent by mesh STAs in a mesh BSS	TID	EOSP	Ack Policy	A-MSDU Present	Mesh Control Present	Mesh Power Save Level	RSPI	Reserved

Figure 7.152: Wifi Quality of Service (QoS) Field Formatting

HT Control field

The HT Control field is always present in a Control Wrapper frame and is present in QoS Data and management frames as determined by the Order bit of the Frame Control field as defined in 8.2.4.1.10. NOTE—The only Control frame subtype for which HT Control field is present is the Control Wrapper frame. A control frame that is described as +HTC (e.g., RTS+HTC, CTS+HTC, BlockAck+HTC or BlockAckReq+HTC) implies the use of the Control Wrapper frame to carry that control frame. The format of the 4-octet HT Control field is shown in Figure8-5. Subfields of the HT control field are defined in the 8.2.4.6 section of the IEEE802.11-2012 specification.

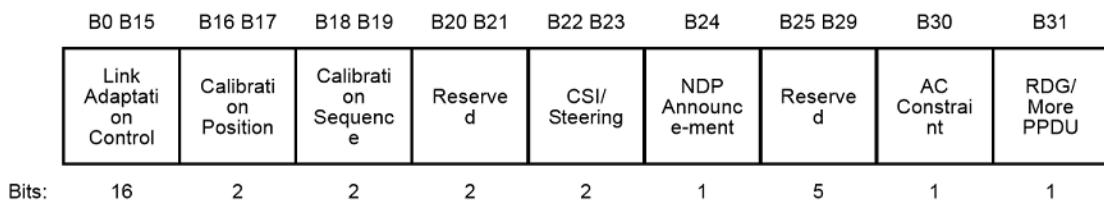
**Figure 8-5—HT Control field**

Figure 7.153: Wifi HT Control Field Formatting

Frame Body field

The Frame Body is a variable-length field that contains information specific to individual frame types and subtypes. The minimum length of the frame body is 0 octets. The maximum length of the frame body is defined by the maximum length MSDU plus the length of Mesh Control field as defined in 8.2.4.7.3, if present, plus any overhead for encryption as defined in Clause11, or by the maximum length A-MSDU plus any overhead for encryption as

defined in Clause11.

Overhead for encryption

The overhead for encryption is described in Clause11. When the Mesh Control field is present in the frame body, the Mesh Control field is encrypted as a part of data.

Mesh Control field

The Mesh Control field is present in the unfragmented Mesh Data frame, in the first fragment of the Mesh Data frame, and in the management frame of subtype Action, Category Multihop Action (Multihop Action frame) transmitted by a mesh STA.

In Mesh Data frames, when the Mesh Control Present subfield in the QoS Control field is 1, the Mesh Control field is prepended to the MSDU and located as follows:

- When the frame body contains an MSDU (or a fragment thereof) and the frame is not encrypted, the Mesh Control field is located in the first octets of the frame body
- When the frame body contains an MSDU (or a fragment thereof) and the frame is encrypted, the Mesh Control field is located in the first octets of the encrypted data portion
- When the frame body contains an A-MSDU, the Mesh Control field is located in the Aggregate MSDU sub-frame header as shown in Figure8-33.

In the Multihop Action frame, the Mesh Control field is present as specified in 8.5.18.

The Mesh Control field is of variable length (6, 12, or 18 octets). The structure of the Mesh Control field is defined in Figure8-9.

FCS field

The FCS field is a 32-bit field containing a 32-bit CRC. The FCS is calculated over all the fields of the MAC header and the Frame Body field. These are referred to as the calculation fields.

The FCS is calculated using the following standard generator polynomial of degree 32:

$$G(x) = x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1$$

The FCS is the ones complement of the sum (modulo 2) of the following:

- a. The remainder of $x^k(x^{31} + x^{30} + x^{29} + \dots + x^2 + x + 1)$ divided (modulo 2) by G(x), where k is the number of bits in the calculation fields, and
- b. The remainder after multiplication of the contents (treated as a polynomial) of the calculation fields by x^{32} and then division by G(x).

The FCS field is transmitted commencing with the coefficient of the highest-order term.

As a typical implementation, at the transmitter, the initial remainder of the division is preset to all ones and is then modified by division of the calculation fields by the generator polynomial G(x). The ones complement of this remainder is transmitted, with the highest-order bit first, as the FCS field.

At the receiver, the initial remainder is preset to all ones and the serial incoming bits of the calculation fields and FCS, when divided by G(x), results (in the absence of transmission errors) in a unique nonzero remainder value. The unique remainder value is the polynomial:

$$G(x) = x^{31} + x^{30} + x^{26} + x^{25} + x^{24} + x^{18} + x^{15} + x^{14} + x^{12} + x^{11} + x^{10} + x^8 + x^6 + x^5 + x^4 + x^3 + x + 1$$

Data frame format

The format of a data frame is defined in Figure8-30. The Frame Control, Duration/ID, Address 1, Address 2, Address 3, and Sequence Control fields are present in all data frame subtypes. The presence of the Address 4 field is determined by the setting of the To DS and From DS subfields of the Frame Control field (see below). The QoS Control field is present when the QoS subfield of the Subtype field is set to 1.

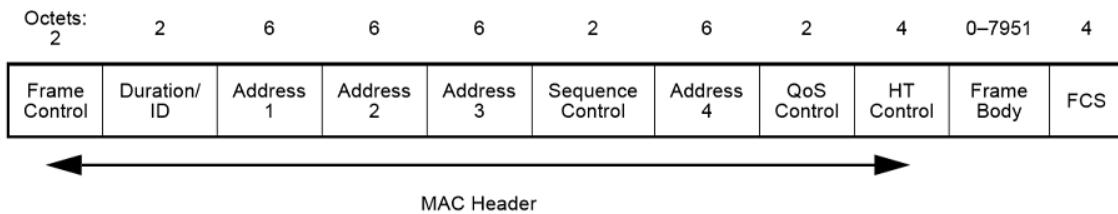
**Figure 8-30—Data frame**

Figure 7.154: WiFi Frame Formatting With Optional Fields

WiGig Frame Control Field format when type=1 and subtype=6

When the value of the Type subfield is equal to 1 and the value of the Subtype subfield is equal to 6, the remaining subfields within the Frame Control field are the following: Control Frame Extension, Power Management, More Data, Protected Frame, and Order. In this case, the format of the Frame Control field is illustrated in Figure 8-2a.

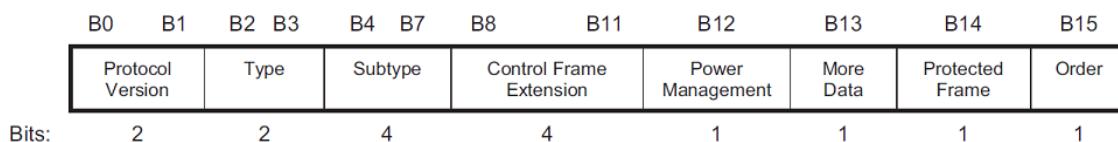
**Figure 8-2a—Frame Control field when Type is equal to 1 and Subtype is equal to 6**

Figure 7.155: Additional Formatting for WiGig Packets

For API Documentation:**See Also**

[ProtocolPP::jprotocol](#)
[ProtocolPP::jmodes](#)
[ProtocolPP::jwifisa](#)
[ProtocolPP::sm4](#)

For Additional Documentation:**See Also**

[jprotocol](#)
[jmodes](#)
[jwifisa](#)
[sm4](#)

Protocol++ written by : John Peter Greninger © John Peter Greninger 2015-2019 All Rights Reserved

All copyrights and trademarks are the property of their respective owners

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer

- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution
- Any and all modifications must be returned to John Peter Greninger at GitHub.com <https://github.com/jpgreninger/protocolpp> for evaluation. Inclusion of modifications in the source code shall be determined solely by John Peter Greninger. Failure to provide modifications shall render this license NULL and VOID and revoke any rights to use of Protocol++
- Commercial use requires a fee-based license obtainable at www.protocolpp.com
- Academic use requires written and notarized permission from John Peter and Sheila Rocha Greninger
- US Copyrights at <https://www.copyright.gov/>
 - TXU002059872 (Version 1.0.0)
 - TXU002066632 (Version 1.2.7)
 - TXU002082674 (Version 1.4.0)
 - TXU002097880 (Version 2.0.0)

The names of its contributors may not be used to endorse or promote products derived from this software without specific prior written permission

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE

The documentation for this class was generated from the following file:

- [include/jwifisa.h](#)

7.139 jwimax Class Reference

```
#include "include/jwimax.h"
```

7.139.1 Detailed Description

7.139.2 WiMax Protocol (WiMax)

See IEEE802.16e-2005 and IEEE802.16-2004 for the following

WiMAX MAC PDUS (See Joint Source-Channel Decoding Appendix Z)

As seen in Section Z.2, in DL subframes, several MAC PDUs are aggregated in bursts. Section 8.3, page 232, presents techniques for performing a reliable burst segmentation in the presence of noise. The content of each MAC PDU is introduced below.

Each MAC PDU begins with a xed-size header, followed by a variable length payload and an optional CRC. There are two types of MAC PDU headers

- The generic MAC header (GMH) is the header of MAC frames containing either MAC management messages or CS data. The CS data may be user data or other higher layer management data. The generic MAC header frame is the only one used in downlink

- The bandwidth request header (BRH) is not followed by any MAC PDU payload or CRC. This frame name has been introduced by the 802.16e amendment. Previously, in 802.16-2004, the BRH was denied to request additional bandwidths. The content of these headers is presented in what follows.

Generic MAC Header (IEEE802.16-2004)

As we are considering only the downlink case, where the connection is already established and MAC PDUs inside the BS contain only CS data, so only the

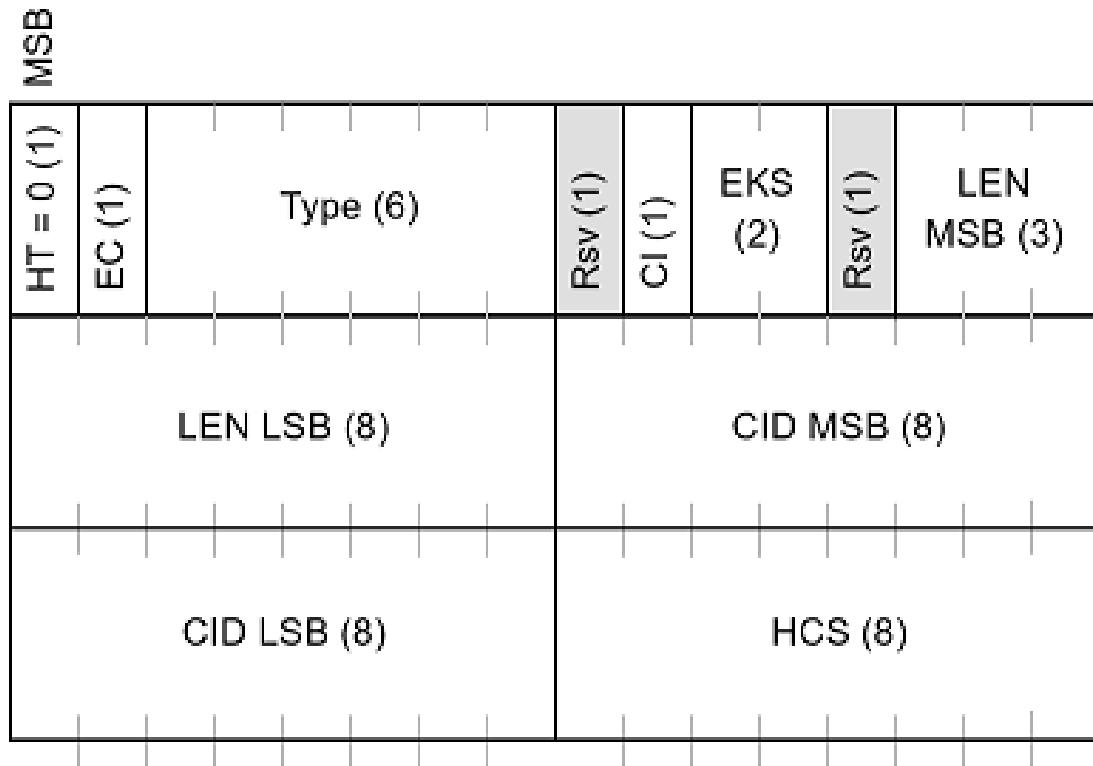


Figure 19—Generic MAC header format

Figure 7.156: Generic MAC Header (GMH) for WiMax Packets

GMH is possible inside BB. Format of GMH as specified in IEEE 802.16-2004 (IEEE, 2004) is illustrated in Figure Z.2. The various fields of a GMH are as follows

- The Header Type (HT) consists of a single bit set to 0 for GMH
 - The Encryption Control(EC) eld species whether the payload is encrypted or not and is set to 0 when the payload is not encrypted and to 1 when it is encrypted
 - The Type eld indicates the subheaders and special payload types present in the message payload (ve possible subheader types)
 - The Reserved (Rsv) eld is of one bit and is set to 0
 - The CRC Indicator (CI) eld is a single bit set to 1 if a CRC is included and is set to 0 if no CRC is included
 - The Encryption Key Sequence (EKS) eld is two bits. It is the index of the Trafic Encryption Key (TEK) and initialization vector used to encrypt the payload. Obviously, this eld is only meaningful if the EC eld is set to 1
 - The Reserved (Rsv) eld is of one bit and is set to 0

- The Length (LEN) eld is 11 bits long. It species the length in bytes of the MAC PDU including the MAC header and the CRC, if present
- The Connection IDentier (CID) eld is 16 bits long and represents the connection identier of the user
- The Header Check Sequence (HCS) eld is 8 bits long and is used to detect errors in the header.

Generic MAC Header (IEEE802.16e-2005 with ESF bit)

As we are considering only the downlink case, where the connection is already established and MAC PDUs inside the BS contain only CS data, so only the

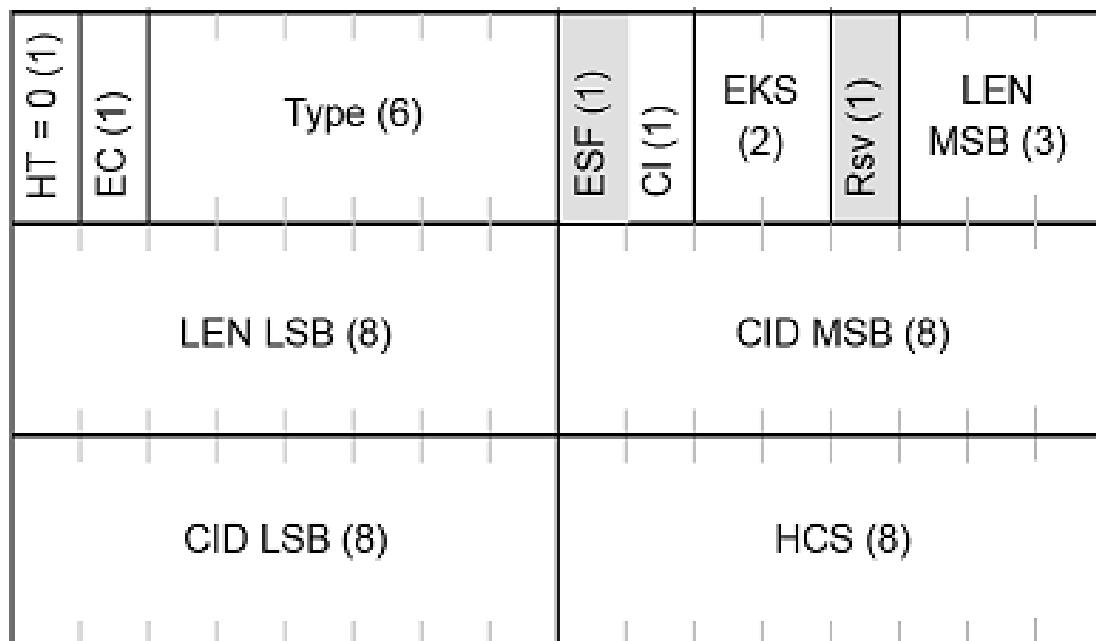


Figure 19—Generic MAC header format

Figure 7.157: ESF Bit Usage in WiMax Packets

GMH is possible inside BB. Format of GMH as specied in IEEE 802.16-2004 (IEEE, 2004) is illustrated in Figure Z.2. The various elds of a GMH are as follows

- The Header Type (HT) consists of a single bit set to 0 for GMH
- The Encryption Control(EC) eld species whether the payload is encrypted or not and is set to 0 when the payload is not encrypted and to 1 when it is encrypted
- The Type eld indicates the subheaders and special payload types present in the message payload (ve possible subheader types)
- The Extended Subheader Field(ESF) eld consists of a single bit set to 1 if the extended subheader is present and follows the GMH immediately (applicable in both the downlink and uplink)
- The CRC Indicator (CI) eld is a single bit set to 1 if a CRC is included and is set to 0 if no CRC is included
- The Encryption Key Sequence (EKS) eld is two bits. It is the index of the Trafic Encryption Key (TEK) and initialization vector used to encrypt the payload. Obviously, this eld is only meaningful if the EC eld is set to 1
- The Reserved (Rsv) eld is one bit and is set to 0

- The Length (LEN) eld is 11 bits long. It species the length in bytes of the MAC PDU including the MAC header and the CRC, if present
- The Connection IDentifier (CID) eld is 16 bits long and represents the connection identier of the user
- The Header Check Sequence (HCS) eld is 8 bits long and is used to detect errors in the header.

Bandwidth Request Header

Bandwidth request headers, shown in Figure Z.3, have the same size as generic MAC frame headers, but their content differs. A bandwidth request PDU consists only of a header and does not contain any payload. The elds of the header are provided below.

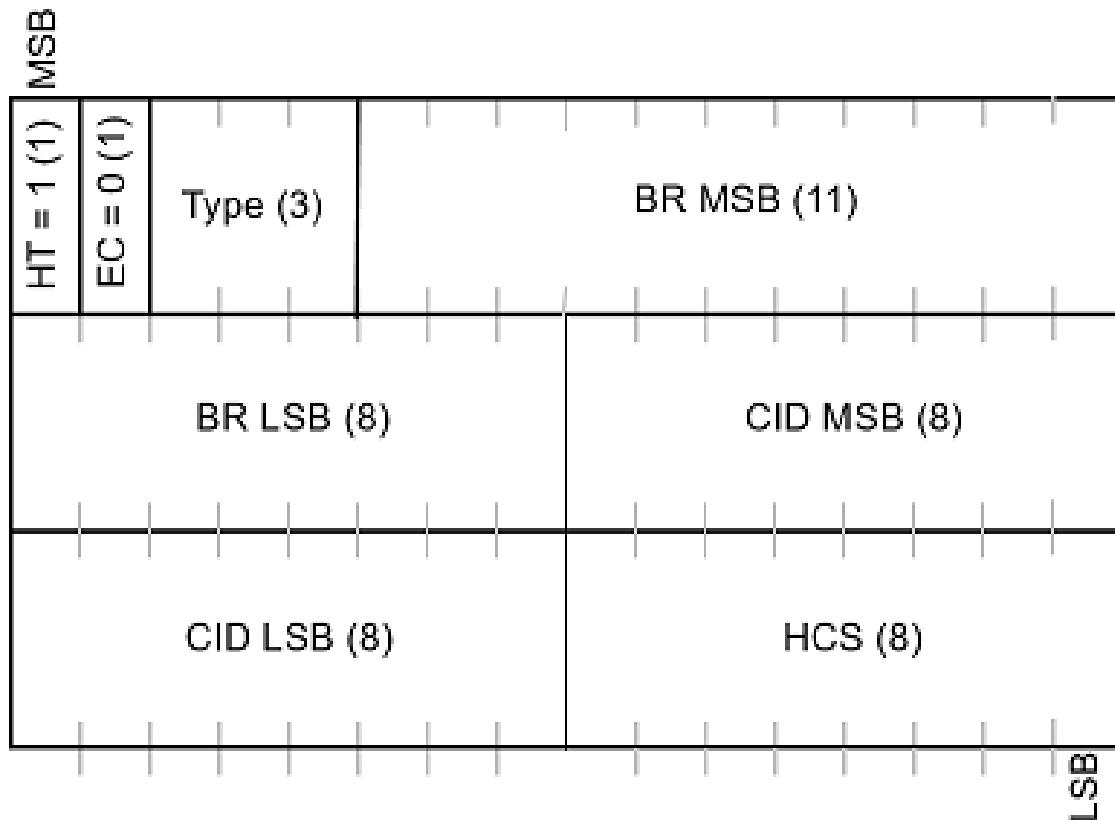


Figure 20—Bandwidth request header format

Figure 7.158: Header format for Bandwidth Request Packets in WiMax

- The HT eld is a single bit set to 1 for BRHs.
- The EC is a single bit set to 0, indicating no encryption
- The type eld is 3 bits long and indicates the type of BRH. It can take two values, namely 000 for incremental and 001 for aggregate bandwidth request
- The bandwidth request(BR) eld is 19 bits long and indicates the number of bytes requested
- The CID eld is 16 bits long and represents the CID of the connection for which uplink bandwidth is requested
- The HCS eld is 8 bits long and is used to detect errors in the header.

Flow identifier (FID)

See IEEE802.16.1-2012 for the following

Each AMS connection is assigned a 4-bit FID that uniquely identifies the connection within the AMS. FIDs identify control connection and unicast transport connections. The FID for E-MBS connection is used along with a 12-bit E-MBS ID to uniquely identify a specific E-MBS flow in the domain of an E-MBS zone (see 6.9.3.2). The FID for multicast connection is used along with a 12-bit Multicast Group ID to uniquely identify a specific multicast flow in the domain of an ABS. DL and UL Transport FIDs are allocated from the transport FID space as defined in Table 6-1. An FID that has been assigned to one DL transport connection shall not be assigned to another DL transport connection belonging to the same AMS. An FID that has been assigned to one UL transport connection shall not be assigned to another UL transport connection belonging to the same AMS. An FID that has been used for a DL transport connection can be assigned to another UL transport connection belonging to the same AMS, or vice versa. Some specific FIDs are preassigned. The values of 0000 and 0001 are used to indicate control FIDs. The values of 0010 and 0011 are used to indicate the FID for the signaling header and the FID for the default service flow, respectively. See Table 6-1 for the specific allocation of FIDs. An FID in combination with an STID uniquely identifies any connection in an ABS

Table 6-1—Flow identifiers

Value	Description
0000	Control FID for unencrypted control connection payload in the MAC PDU (unicast control FID when PDU is allocated by unicast assignment A-MAP IE; broadcast control FID when PDU is allocated by broadcast assignment A-MAP IE)
0001	Unicast Control FID for encrypted control connection payload in the MAC PDU
0010	FID for Signaling header
0011	FID for transport connection associated with default service flow (UL and DL)
0100–1111	FID for transport connection

Figure 7.159: Flow Identifier (FID) for WiMax Packets

MAC header formats

There are three defined MAC header formats: the advanced generic MAC header, the Short-Packet MAC header, and the MAC signaling header. At any connection, only one of the following formats shall be used: advanced generic MAC header, short-packet MAC header, and MAC signaling header

Advanced generic MAC header (AGMH)

The AGMH format is defined in Table 6-2. For E-MBS services, the FID shall be ignored by the receiver. For multicast service, the multicast-specific FID associated with Multicast Group ID shall be used in the FID field of AGMH

Table 6-2—AGMH format

Syntax	Size (bits)	Notes
Advanced Generic MAC header() {		
FID	4	Flow identifier.
EH	1	Extended header group presence indicator; When set to '1', this field indicates that an Extended Header group is present following this AGMH.
Length	11	This field indicates the length in bytes of MAC PDU including the AGMH and extended header if present. If EH is set to '1', this represents the 11 LSBs of 14-bit MAC PDU length; otherwise, it represents the 11-bit MAC PDU length.
}		

Figure 7.160: Advanced Generic MAC Header for WiMax Packets

Short-packet MAC header (SPMH)

The SPMH is defined to support applications, such as Voice over IP (VoIP), that utilize small data packets and non-ARQ connections. The extended header group may be piggybacked on the SPMH, if allowed by its length field. The SPMH is identified by the specific FID that is provisioned statically, or created dynamically via AAI-DSA-REQ/RSP

Table 6-3—SPMH format

Syntax	Size (bits)	Notes
Short-Packet MAC Header() {		
FID	4	Flow identifier.
EH	1	Extended header group presence indicator; When set to '1', this field indicates that an Extended Header group is present following this SPMH.
Length	7	This field indicates the length in bytes of MAC PDU including the SPMH and extended header if present.
SN	4	MAC PDU payload sequence number increments by one for each MAC PDU (modulo 16).
}		

Figure 7.161: Formatting for Short-Packet MAC Header (SPMH) in WiMax

MAC signaling header

The signaling header shall be sent standalone or concatenated with other MAC PDUs in either DL or UL. One FID is reserved for the MAC signaling header. The value of FID for the MAC signaling header is 0010

All MAC signaling header formats follow the layout defined in Table 6-4.

Table 6-4—MAC signaling header format

Syntax	Size (bits)	Notes
MAC Signaling Header() {		
FID	4	Flow Identifier. Set to 0010.
Type	5	MAC signaling header type.
Length	3	Indicates the length of the signaling header (includes the FID, Type, Length, and contents): 0b000 and 0b001: <i>Reserved</i> 0b010: 2 bytes 0b011: 3 bytes 0b100: 4 bytes 0b101: 5 bytes 0b110: 6 bytes 0b111: <i>Reserved</i>
Contents	<i>variable; ≤ 36</i>	MAC signaling header contents, with the size indicated by the length field. The size in bits is Length × 8 – 12.
}		

Figure 7.162: Formatting of MAC Signaling Header in WiMax

Cryptographic methods (See IEEE802.16.1-2012)**Payload encryption methods**

AES-CCM [refer to NIST Special Publication 800-38C and FIPS 197 Advanced Encryption Standard (AES)] shall be used as an encryption method when PDUs on the unicast control connection are encrypted. Unicast transport connections may be encrypted with AES-CTR (refer to NIST Special Publication 800-38A) or AES-CCM.

AES-CCM PDU payload format

The MAC PDU payload shall be prepended with a 2-bit EKS and a 22-bit PN (Packet Number). The EKS and PN shall not be encrypted. The plaintext PDU shall be encrypted and authenticated using the active TEK, according to the CCM specification. This includes appending an integrity check value (ICV) to the end of the payload and encrypting both the plaintext payload and the appended ICV where the size of ICV is decided by either 4 bytes or 8 bytes during the key agreement procedure in network entry. The processing yields a payload that is 7 bytes or 11 bytes longer than the plaintext payload.

Packet number (PN)

The PN associated with an SA shall be set to 0x000001 for DL and 0x200001 for UL when the SA is established and when a new TEK is installed. After each PDU transmission, the PN shall be incremented by 1. Any pair value of {PN, TEK} shall not be used more than once for the purposes of transmitting data. The AMS shall ensure that a new TEK is derived on both sides before the PN paired with either TEK for downlink or TEK for uplink reaches maximum value 0xFFFF or 0x3FFFF, respectively. If the PN reaches maximum value without new TEKs being installed, transport communications on that SA shall be halted until new TEKs are installed.

CCM algorithm

NIST Special Publication 800-38C defines a number of algorithm parameters. Those parameters shall be fixed to specific values as follows:

- The payload, P, shall be the connection payload, i.e., the PDU excluding the MAC header and any extended headers as well as the EKS, PN, and the Integrity Check Value (ICV) fields (refer to Figure 6-11)
- The message authentication code, T, is referred herein as the ICV. The length of T in bytes, t, is either 4 or 8

- The cipher block key, K, is the current TEK as defined herein
- The associated data, A, is the empty string. The length of the associated data, a, is zero.

The nonce, N, shall be as shown in Table 6-112. The MAC header, which fills bytes 1 and 2, may be the AGMH or the SPMH. If the STID or the FID has not been assigned, then the corresponding field shall be set to all zeroes. When payloads with different FIDs are multiplexed into the MAC PDU, the FID field of the nonce shall be set to the FID of the first payload connection.

Table 6-112—Nonce construction

Byte Number	0 1	2 3	4 9	10 12
Field	MAC Header	STID and Flow ID	<i>Reserved</i>	EKS and Packet Number
Contents	AGMH or SPMH	STID FID	0X00000000000000	EKS PN

Figure 7.163: Format of NONCE in WiMax Security Packets

The block cipher algorithm, CIPHK, is AES as specified in FIPS 197 Advanced Encryption Standard (AES).

The formatting function and the counter generation function are specified in NIST Special Publication 800-38C, Appendix A. In adopting Appendix A, the Length field Q shall be 2 bytes long; i.e., the octet length of the Length field, q, is 2. It follows that the Flags byte in the initial block, B0, has value 0x09 if a 4-byte ICV is used, or 0x19 if an 8-byte ICV is used. The initial block B0 is shown in Figure 6-21.

Byte Number	0	1	13	14	15
Byte significance				MSB	LSB
Number of bytes	1		13		2
Field	Flags		Nonce		Q
Contents	0x09 or 0x19	As specified in Table 6-109			Length of Plaintext payload

Figure 6-21—Construction of initial CCM Block B₀

Figure 7.164: Format of BZERO Field for WiMax Security Packets

The j-th counter block, Ctrj, is shown in Figure 6-22. Note that the Flags byte in the counter blocks, which is different from the Flags byte in the initial block B0, has constant value 0x01.

Byte Number	0	1	13	14	15
Byte significance				MSB	LSB
Number of bytes	1		13		2
Field	Flags		Nonce		Counter
Contents	0x1		As specified in Table 6-109		j

Figure 6-22—Construction of counter blocks Ctr_j

Figure 7.165: Format of CNTR0 Field for WiMax Security Packets

Receive Processing rules

On receipt of a PDU, the receiving AMS or ABS shall decrypt and authenticate the PDU consistent with the NIST CCM specification configured as specified previously. Packets that are found to be not authentic shall be discarded. Receiving ABSS or AMSs shall maintain a record of the highest value PN received for each SA.

The receiver shall maintain a PN window whose size is specified by the PN_WINDOW_SIZE parameter for SAs. Any received PDU with a PN lower than the beginning of the PN window shall be discarded as a replay attempt. The receiver shall track PNs within the PN window. Any PN that is received more than once shall be discarded as a replay attempt. Upon reception of a PN, which is greater than the end of the PN window, the PN window shall be advanced to cover this PN.

TEK update should be completed before MAC PDUs with ICV error are detected over the MaxInvalid times for the same TEK. If AMS recognizes that TEKDLE update is required due to ICV errors, it initiates TEK update by sending a PKMv3 TEK-Invalid message to the ABS. On receiving the PKMv3 TEK-Invalid message, the ABS discards current TEKDLE and uses TEKULE as TEKDLE, and derives a new TEK for TEKULE.

If ABS recognizes that TEKULE update is required due to ICV errors, it discards current TEKDLE and uses TEKULE as TEKDLE, and derives a new TEK for TEKULE. After recognizing ABS's TEK update, AMS performs the TEK update procedure. To expedite the TEK update procedure, the ABS may transmit the PKMv3 TEK-invalid message.

When the ABS detects that EKS is not synchronized yet, the ABS transmits the PKMv3 TEK-Invalid message in order for the AMS to send the PKMv3 TEK-Request message to the ABS. On receiving the PKMv3 TEK-Request message, the ABS responds with a PKMv3 TEK-reply message notifying the current TEKs.

AES-CTR

The MAC PDU payload shall be prepended with a 2-bit EKS and a 22-bit PN. The EKS and PN shall not be encrypted. Construction of the counter blocks is the same as that for counter blocks of AES-CCM (i.e., the counter blocks CTR_j and NONCE are formatted as shown in Figure6-22 and Figure6-21, respectively).

Calculation of cipher-based message authentication code (CMAC)

An ABS or AMS may support MAC control message integrity protection based on CMAC, together with the AES block cipher. The CMAC construction as specified in NIST Special Publication 800-38B shall be used.

The calculation of the keyed hash value contained in the CMAC Digest attribute and the CMAC Tuple shall use the CMAC algorithm with AES. The DL authentication key CMAC_KEY_D shall be used for authenticating messages in the DL direction. The UL authentication key CMAC_KEY_U shall be used for authenticating messages in the UL direction. UL and DL message authentication keys are derived from the CMAC-TEK prekey (see 6.2.5.2.1.1.3 for details).

The CMAC Packet Number Counter, CMAC_PN_*, is a 3-byte sequential counter that is incremented for each MAC Control Message that contains a CMAC Tuple or CMAC Digest in the context of UL messages by the AMS, and in the context of DL messages by the ABS.

If STID or TSTID is not assigned yet, then the STID field shall be stuffed with zeroes. The CMAC_PN_* is part of the AK context and shall be unique for each MAC control message with the CMAC tuple or digest. Any tuple value of {CMAC_PN_*, CMAC_KEY_*} shall not be used more than once. The reauthorization process should be initiated (by the ABS or the AMS) to establish a new PMK/AK before the CMAC_PN_* reaches the end of its number space.

The CMAC value shall be calculated over a field consisting of the AK ID followed by the CMAC_PN_*, expressed as an unsigned 24-bit number, followed by the 12-bit STID and 4-bit FID on which the message is sent, followed by 24-bit zero padding (for the header to be aligned with AES block size) and followed by the entire ASN.1 encoded MAC control message.

The LSB 64-bit of the outcome of AES-CMAC calculation shall be used for CMAC value.

NOTE—This is different from the recommendation in NIST Special Publication 800-38B where the MSB is used to derive the CMAC value.¹⁰

In other words, if CMAC_KEY_* is derived from CMAC-TEK prekey:

CMAC value Truncate(CMAC (CMAC_KEY_*, AK ID | CMAC_PN |STID|FID|24-bit zero padding | ASN.1 encoded MAC_Control_Message), 64), where STID ‘000000000000’ should be used if STID is not assigned yet.

Only the MAC control message of CMAC_PN that arrives in order at the receiver side can be accepted. MAC control messages with out-of-order CMAC_PN shall be discarded

For API Documentation:

See Also

[ProtocolPP::jprotocol](#)
[ProtocolPP::jmodes](#)
[ProtocolPP::jwimaxsa](#)
[ProtocolPP::sm4](#)

For Additional Documentation:

See Also

[jprotocol](#)
[jmodes](#)
[jwimaxsa](#)
[sm4](#)

Protocol++ written by : John Peter Greninger © John Peter Greninger 2015-2019 All Rights Reserved

All copyrights and trademarks are the property of their respective owners

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer
- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution
- Any and all modifications must be returned to John Peter Greninger at GitHub.com <https://github.com/jpgreninger/protocolpp> for evaluation. Inclusion of modifications in the source code shall be determined solely by John Peter Greninger. Failure to provide modifications shall render this license NULL and VOID and revoke any rights to use of Protocol++
- Commercial use requires a fee-based license obtainable at www.protocolpp.com
- Academic use requires written and notarized permission from John Peter and Sheila Rocha Greninger
- US Copyrights at <https://www.copyright.gov/>
 - TXu002059872 (Version 1.0.0)
 - TXu002066632 (Version 1.2.7)
 - TXu002082674 (Version 1.4.0)

- TXu002097880 (Version 2.0.0)

The names of its contributors may not be used to endorse or promote products derived from this software without specific prior written permission

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE

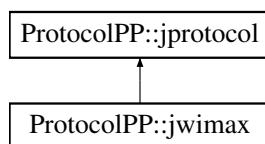
The documentation for this class was generated from the following file:

- [include/jwimax.h](#)

7.140 ProtocolPP::jwimax Class Reference

```
#include <jwimax.h>
```

Inheritance diagram for ProtocolPP::jwimax:



Public Member Functions

- [jwimax \(std::shared_ptr< \[jwimaxsa\]\(#\) > &security\)](#)
 - virtual [~jwimax \(\)](#)
- Standard deconstructor.*
- void [encap_packet \(std::shared_ptr< \[jarray\]\(#\)< uint8_t >> &input, std::shared_ptr< \[jarray\]\(#\)< uint8_t >> &output\)](#)
 - void [decap_packet \(std::shared_ptr< \[jarray\]\(#\)< uint8_t >> &input, std::shared_ptr< \[jarray\]\(#\)< uint8_t >> &output\)](#)
 - void [set_hdr \(\[jarray\]\(#\)< uint8_t > &hdr\)](#)
 - void [set_field \(field_t field, uint64_t value\)](#)
 - [jarray< uint8_t > get_hdr \(\)](#)
 - uint64_t [get_field \(field_t field, \[jarray\]\(#\)< uint8_t > &header\)](#)
 - void [to_xml \(tinyxml2::XMLPrinter &myxml, direction_t direction\)](#)

Additional Inherited Members

7.140.1 Constructor & Destructor Documentation

7.140.1.1 ProtocolPP::jwimax::jwimax (std::shared_ptr< [jwimaxsa](#) > & security)

Constructor for wimax

Parameters

<i>security</i>	- Security association (SA) for this wimax flow
-----------------	---

7.140.1.2 virtual ProtocolPP::jwimax::~jwimax() [inline], [virtual]

Standard deconstructor.

7.140.2 Member Function Documentation

7.140.2.1 void ProtocolPP::jwimax::decap_packet(std::shared_ptr< jarray< uint8_t >> &*input*, std::shared_ptr< jarray< uint8_t >> &*output*) [virtual]

Decap will produce a payload from the packet passed

Parameters

<i>input</i>	- packet to decapsulate with WEP/WPA
<i>output</i>	- packet encapsulated with WEP/WPA

Implements [ProtocolPP::jprotocol](#).

7.140.2.2 void ProtocolPP::jwimax::encap_packet(std::shared_ptr< jarray< uint8_t >> &*input*, std::shared_ptr< jarray< uint8_t >> &*output*) [virtual]

Encap will produce a packet from the payload passed

Parameters

<i>input</i>	- payload to protect with WEP/WPA
<i>output</i>	- packet encapsulated with WEP/WPA

Implements [ProtocolPP::jprotocol](#).

7.140.2.3 uint64_t ProtocolPP::jwimax::get_field(field_t *field*, jarray< uint8_t > &*header*) [virtual]

Retrieve the field from the wimax header

Parameters

<i>field</i>	- field to retrieve
<i>header</i>	- header to extract the field from

Returns

field

Implements [ProtocolPP::jprotocol](#).

7.140.2.4 jarray<uint8_t> ProtocolPP::jwimax::get_hdr() [virtual]

Retrieve the wimax header

Returns

current wimax header

Implements [ProtocolPP::jprotocol](#).

7.140.2.5 void ProtocolPP::jwimax::set_field (*field_t field*, *uint64_t value*) [virtual]

Update wimax field with the new value

Parameters

<i>field</i>	- field to update
<i>value</i>	- new value for the field

Implements [ProtocolPP::jprotocol](#).

7.140.2.6 void ProtocolPP::jwimax::set_hdr (*jarray< uint8_t > & hdr*) [virtual]

Update the current wimax header with a new header

Parameters

<i>hdr</i>	- new wimax header for this flow
------------	----------------------------------

Implements [ProtocolPP::jprotocol](#).

7.140.2.7 void ProtocolPP::jwimax::to_xml (*tinyxml2::XMLPrinter & myxml*, *direction_t direction*) [virtual]

Print the protocol and security objects to XML

Parameters

<i>myxml</i>	- XMLPrinter object for printing
<i>direction</i>	- facilitator for random generation

Implements [ProtocolPP::jprotocol](#).

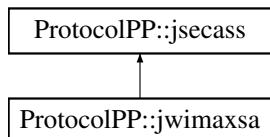
The documentation for this class was generated from the following file:

- [include/jwimax.h](#)

7.141 ProtocolPP::jwimaxsa Class Reference

```
#include <jwimaxsa.h>
```

Inheritance diagram for ProtocolPP::jwimaxsa:



Public Member Functions

- [jwimaxsa \(\)](#)
 - [jwimaxsa \(*direction_t* dir, *wimaxmode_t* mode, *cipher_t* cipher, *uint8_t* type, *uint8_t* eks, *uint8_t* fid, *uint16_t* cid, *uint32_t* pn, *unsigned int* ckeylen, *std::shared_ptr< jarray< uint8_t >>* cipherkey, *unsigned int* arlen, *jarray< uint8_t >* arwin, *bool* eh, *bool* ht, *bool* ec, *bool* esf, *bool* ci\)](#)
 - [jwimaxsa \(*jwimaxsa &rhs*\)](#)
 - [jwimaxsa \(*std::shared_ptr< jwimaxsa > &rhs*\)](#)
 - virtual [~jwimaxsa \(\)](#)
- Standard deconstructor.*
- template<typename T>
[void set_field \(*field_t* field, T fieldval\)](#)

- template<typename T >
T [get_field](#) (field_t field)
- void [to_xml](#) (tinyxml2::XMLPrinter &myxml, direction_t direction)

7.141.1 Constructor & Destructor Documentation

7.141.1.1 ProtocolPP::jwimaxsa()

Standard Constructor

7.141.1.2 ProtocolPP::jwimaxsa(direction_t dir, wimaxmode_t mode, cipher_t cipher, uint8_t type, uint8_t eks, uint8_t fid, uint16_t cid, uint32_t pn, unsigned int ckeylen, std::shared_ptr<jarray<uint8_t>> cipherkey, unsigned int arlen, jarray<uint8_t> arwin, bool eh, bool ht, bool ec, bool esf, bool ci)

Security Association for Wimax

Parameters

<i>dir</i>	- Direction of processing (ENCAP or DECAP)
<i>mode</i>	- Mode of operation (OFDM, OFDMA, AGHM_OFDM, AGHM_OFDMA)
<i>cipher</i>	- Encryption (AES_CCM, SM4_CCM, NULL_CIPHER)
<i>type</i>	- types and subheaders in payload
<i>eks</i>	- Encryption key sequence
<i>cid</i>	- Connection Identifier
<i>fid</i>	- Flow Identifier (AGHM header only)
<i>pn</i>	- Packet number (for counter modes PN=ROC[7:0] PN[23:0])
<i>ckeylen</i>	- Length of the key
<i>cipherkey</i>	- Key for the encryption algorithm
<i>arlen</i>	- Number of packets to track in replay window
<i>arwin</i>	- Anti-replay window for tracking packets
<i>eh</i>	- Extended header group presence identifier (AGHM header only)
<i>ht</i>	- Header type
<i>ec</i>	- Encryption control
<i>esf</i>	- Extended subheader present
<i>ci</i>	- CRC Indicator

7.141.1.3 ProtocolPP::jwimaxsa(jwimaxsa &rhs)

Copy constructor for wimax security association

Parameters

<i>rhs</i>	- Security association (SA) for this wimax flow
------------	---

7.141.1.4 ProtocolPP::jwimaxsa(std::shared_ptr<jwimaxsa> &rhs)

Copy constructor for wimax security association from shared pointer

Parameters

<i>rhs</i>	- Security association (SA) for this wimax flow
------------	---

Table 7 Copyright 2017-2019 John Peter Greninger : Generated on Sat Aug 31 2019 22:28:11 for Protocol++ (ProtocolIPP) 3.0.1 by Doxygen

Each MAC PDU begins with a fixed-size header, followed by a variable length payload and an optional CRC. There are two types of MAC PDU headers

- The generic MAC header (GMH) is the header of MAC frames containing either MAC management messages or CS data. The CS data may be user data or other higher layer management data. The generic MAC header frame is the only one used in downlink
- The bandwidth request header (BRH) is not followed by any MAC PDU payload or CRC. This frame name has been introduced by the 802.16e amendment. Previously, in 802.16-2004, the BRH was dened to request additional bandwidths. The content of these headers is presented in what follows.

Generic MAC Header (IEEE802.16-2004)

As we are considering only the downlink case, where the connection is already established and MAC PDUs inside the BS contain only CS data, so only the

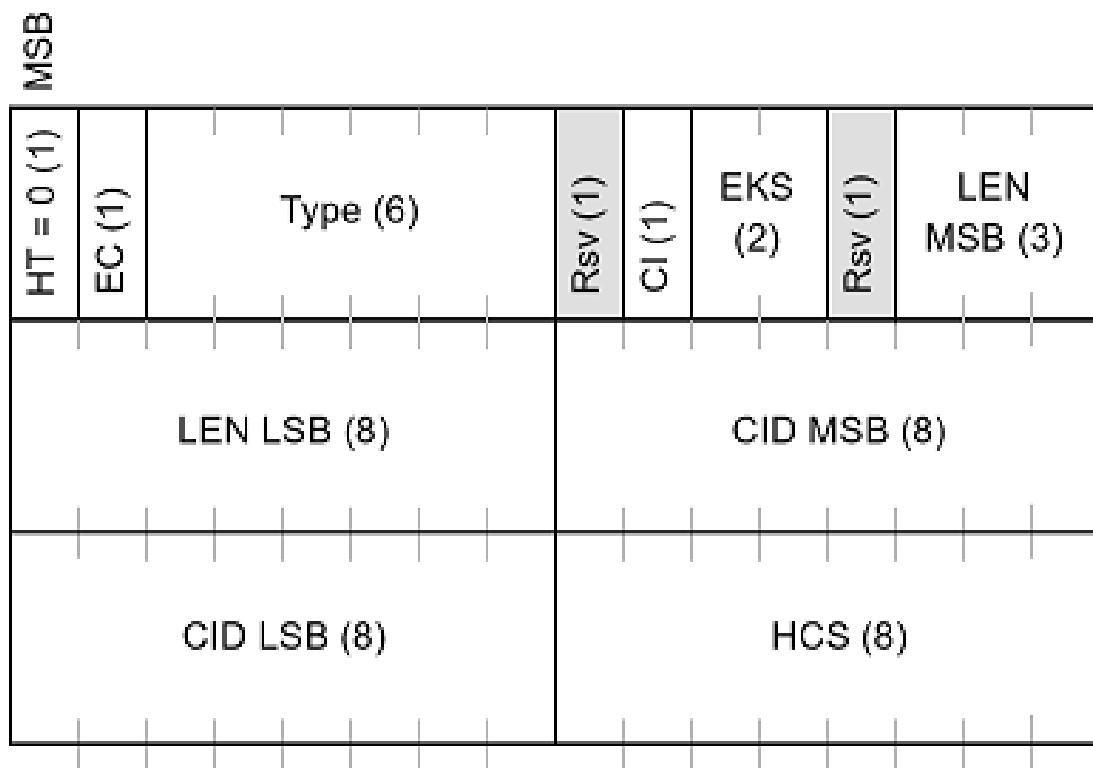


Figure 19—Generic MAC header format

Figure 7.166: Generic MAC Header (GMH) for WiMax Packets

GMH is possible inside BB. Format of GMH as specied in IEEE 802.16-2004 (IEEE, 2004) is illustrated in Figure Z.2. The various elds of a GMH are as follows

- The Header Type (HT) consists of a single bit set to 0 for GMH
- The Encryption Control(EC) eld species whether the payload is encrypted or not and is set to 0 when the payload is not encrypted and to 1 when it is encrypted
- The Type eld indicates the subheaders and special payload types present in the message payload (ve possible subheader types)

- The Reserved (Rsv) eld is of one bit and is set to 0
- The CRC Indicator (CI) eld is a single bit set to 1 if a CRC is included and is set to 0 if no CRC is included
- The Encryption Key Sequence (EKS) eld is two bits. It is the index of the Trafic Encryption Key (TEK) and initialization vector used to encrypt the payload. Obviously, this eld is only meaningful if the EC eld is set to 1
- The Reserved (Rsv) eld is of one bit and is set to 0
- The Length (LEN) eld is 11 bits long. It species the length in bytes of the MAC PDU including the MAC header and the CRC, if present
- The Connection IDentier (CID) eld is 16 bits long and represents the connection identier of the user
- The Header Check Sequence (HCS) eld is 8 bits long and is used to detect errors in the header.

Generic MAC Header (IEEE802.16e-2005 with ESF bit)

As we are considering only the downlink case, where the connection is already established and MAC PDUs inside the BS contain only CS data, so only the

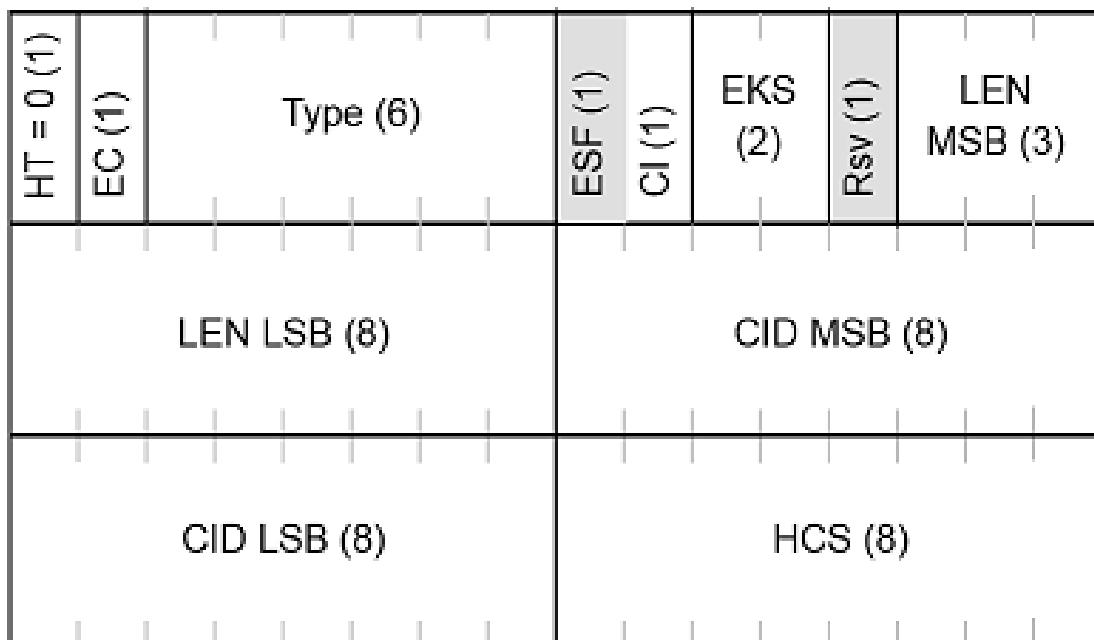


Figure 19—Generic MAC header format

Figure 7.167: ESF Bit Usage and Formatting in WiMax Packets

GMH is possible inside BB. Format of GMH as specied in IEEE 802.16-2004 (IEEE, 2004) is illustrated in Figure Z.2. The various elds of a GMH are as follows

- The Header Type (HT) consists of a single bit set to 0 for GMH
- The Encryption Control(EC) eld species whether the payload is encrypted or not and is set to 0 when the payload is not encrypted and to 1 when it is encrypted
- The Type eld indicates the subheaders and special payload types present in the message payload (ve possible subheader types)
- The Extended Subheader Field(ESF) eld consists of a single bit set to 1 if the extended subheader is present and follows the GMH immediately (applicable in both the downlink and uplink)

- The CRC Indicator (CI) eld is a single bit set to 1 if a CRC is included and is set to 0 if no CRC is included
- The Encryption Key Sequence (EKS) eld is two bits. It is the index of the Trafic Encryption Key (TEK) and initialization vector used to encrypt the payload. Obviously, this eld is only meaningful if the EC eld is set to 1
- The Reserved (Rsv) eld is one bit and is set to 0
- The Length (LEN) eld is 11 bits long. It species the length in bytes of the MAC PDU including the MAC header and the CRC, if present
- The Connection IDentier (CID) eld is 16 bits long and represents the connection identier of the user
- The Header Check Sequence (HCS) eld is 8 bits long and is used to detect errors in the header.

MAC header formats

There are three defined MAC header formats: the advanced generic MAC header, the Short-Packet MAC header, and the MAC signaling header. At any connection, only one of the following formats shall be used: advanced generic MAC header, short-packet MAC header, and MAC signaling header

Advanced generic MAC header (AGMH)

The AGMH format is defined in Table 6-2. For E-MBS services, the FID shall be ignored by the receiver. For multicast service, the multicast-specific FID associated with Multicast Group ID shall be used in the FID field of AGMH

Table 6-2—AGMH format

Syntax	Size (bits)	Notes
Advanced Generic MAC header() {		
FID	4	Flow identifier.
EH	1	Extended header group presence indicator; When set to '1', this field indicates that an Extended Header group is present following this AGMH.
Length	11	This field indicates the length in bytes of MAC PDU including the AGMH and extended header if present. If EH is set to '1', this represents the 11 LSBs of 14-bit MAC PDU length; otherwise, it represents the 11-bit MAC PDU length.
}		

Figure 7.168: Advanced Generic MAC (AGMH) Header for WiMax Packets

Short-packet MAC header (SPMH)

The SPMH is defined to support applications, such as Voice over IP (VoIP), that utilize small data packets and non-ARQ connections. The extended header group may be piggybacked on the SPMH, if allowed by its length field. The SPMH is identified by the specific FID that is provisioned statically, or created dynamically via AAI-DSA-REQ/RSP

Table 6-3—SPMH format

Syntax	Size (bits)	Notes
Short-Packet MAC Header() {		
FID	4	Flow identifier.
EH	1	Extended header group presence indicator; When set to '1', this field indicates that an Extended Header group is present following this SPMH.
Length	7	This field indicates the length in bytes of MAC PDU including the SPMH and extended header if present.
SN	4	MAC PDU payload sequence number increments by one for each MAC PDU (modulo 16).
}		

Figure 7.169: Formatting for Short-Packet MAC Header (SPMH) in WiMax Packets

MAC signaling header

The signaling header shall be sent standalone or concatenated with other MAC PDUs in either DL or UL. One FID is reserved for the MAC signaling header. The value of FID for the MAC signaling header is 0010

All MAC signaling header formats follow the layout defined in Table 6-4.

Table 6-4—MAC signaling header format

Syntax	Size (bits)	Notes
MAC Signaling Header() {		
FID	4	Flow Identifier. Set to 0010.
Type	5	MAC signaling header type.
Length	3	Indicates the length of the signaling header (includes the FID, Type, Length, and contents): 0b000 and 0b001: <i>Reserved</i> 0b010: 2 bytes 0b011: 3 bytes 0b100: 4 bytes 0b101: 5 bytes 0b110: 6 bytes 0b111: <i>Reserved</i>
Contents	<i>variable</i> ; ≤ 36	MAC signaling header contents, with the size indicated by the length field. The size in bits is $\text{Length} \times 8 - 12$.
}		

Figure 7.170: Formatting for MAC Signaling Header in WiMax Packets

For API Documentation:

See Also

[ProtocolPP::jprotocol](#)
[ProtocolPP::jmodes](#)
[ProtocolPP::jwimaxsa](#)
[ProtocolPP::sm4](#)
[ProtocolPP::jsecass](#)

For Additional Documentation:**See Also**

[jprotocol](#)
[jmodes](#)
[jwimaxsa](#)
[sm4](#)
[jsecass](#)

Protocol++ written by : John Peter Greninger © John Peter Greninger 2015-2019 All Rights Reserved

All copyrights and trademarks are the property of their respective owners

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer
- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution
- Any and all modifications must be returned to John Peter Greninger at GitHub.com <https://github.com/jpgreninger/protocolpp> for evaluation. Inclusion of modifications in the source code shall be determined solely by John Peter Greninger. Failure to provide modifications shall render this license NULL and VOID and revoke any rights to use of Protocol++
- Commercial use requires a fee-based license obtainable at www.protocolpp.com
- Academic use requires written and notarized permission from John Peter and Sheila Rocha Greninger
- US Copyrights at [https://www.copyright.gov/](http://www.copyright.gov/)
 - TXu002059872 (Version 1.0.0)
 - TXu002066632 (Version 1.2.7)
 - TXu002082674 (Version 1.4.0)
 - TXu002097880 (Version 2.0.0)

The names of its contributors may not be used to endorse or promote products derived from this software without specific prior written permission

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE

The documentation for this class was generated from the following file:

- [include/jwimaxsa.h](#)

7.143 jzuc Class Reference

```
#include "include/jzuc.h"
```

7.143.1 Detailed Description

7.143.2 3GPP ZUC Protocol

Code for the ZUC algorithm was derived from

ETSI/SAGE Specification Version: 1.7 Date: 30th Dec, 2011

Specification of the 3GPP Confidentiality and Integrity Algorithms 128-EEA3 & 128-EIA3

Document 1: 128-EEA3 and 128-EIA3 Specification

For API Documentation:

See Also

[ProtocolPP::jarray](#)

For Additional Documentation:

See Also

[jarray](#)

Protocol++ written by : John Peter Greninger © John Peter Greninger 2015-2019 All Rights Reserved

All copyrights and trademarks are the property of their respective owners

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer
- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution
- Any and all modifications must be returned to John Peter Greninger at GitHub.com <https://github.com/jpgreninger/protocolpp> for evaluation. Inclusion of modifications in the source code shall be determined solely by John Peter Greninger. Failure to provide modifications shall render this license NULL and VOID and revoke any rights to use of Protocol++
- Commercial use requires a fee-based license obtainable at www.protocolpp.com
- Academic use requires written and notarized permission from John Peter and Sheila Rocha Greninger
- US Copyrights at <https://www.copyright.gov/>
 - TXu002059872 (Version 1.0.0)
 - TXu002066632 (Version 1.2.7)
 - TXu002082674 (Version 1.4.0)
 - TXu002097880 (Version 2.0.0)

The names of its contributors may not be used to endorse or promote products derived from this software without specific prior written permission

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE

The documentation for this class was generated from the following file:

- [include/jzuc.h](#)

7.144 ProtocolPP::jzuc Class Reference

```
#include <jzuc.h>
```

Public Types

- enum [dir_t](#) { [UPLINK](#), [DOWNLINK](#) }

Public Member Functions

- [jzuc \(dir_t dir, uint8_t *key, unsigned int keylen, uint32_t count, uint8_t bearer\)](#)
- [jzuc \(dir_t dir, uint8_t *key, unsigned int keylen, uint32_t count, uint32_t bearer\)](#)
- virtual [~jzuc \(\)](#)
Standard deconstructor.
- void [ProcessData \(const uint8_t *input, uint8_t *output, uint32_t length\)](#)
- void [ProcessData \(const uint8_t *input, uint32_t length\)](#)
- void [result \(uint8_t *result\)](#)
- unsigned int [result_size \(\)](#)

7.144.1 Member Enumeration Documentation

7.144.1.1 enum ProtocolPP::jzuc::dir_t

Direction of processing for ZUC

Parameters

UPLINK	- User to tower
DOWNLINK	- Tower to user

Enumerator

[UPLINK](#)

[DOWNLINK](#)

7.144.2 Constructor & Destructor Documentation

7.144.2.1 `ProtocolPP::jzuc::jzuc (dir_t dir, uint8_t * key, unsigned int keylen, uint32_t count, uint8_t bearer)`

Constructor for ZUC encryption

Parameters

<i>dir</i>	- direction of processing
<i>key</i>	- encryption key
<i>keylen</i>	- length of the encryption key (16 bytes)
<i>count</i>	- 32-bit COUNT value
<i>bearer</i>	- bearer value

7.144.2.2 `ProtocolPP::jzuc::jzuc (dir_t dir, uint8_t * key, unsigned int keylen, uint32_t count, uint32_t bearer)`

Constructor for ZUC authentication

Parameters

<i>dir</i>	- direction of processing (DLINK, UPLINK)
<i>key</i>	- authentication key
<i>keylen</i>	- length of the authentication key (16 bytes)
<i>count</i>	- 32-bit COUNT value
<i>bearer</i>	- bearer value

7.144.2.3 `virtual ProtocolPP::jzuc::~jzuc () [inline], [virtual]`

Standard deconstructor.

7.144.3 Member Function Documentation

7.144.3.1 `void ProtocolPP::jzuc::ProcessData (const uint8_t * input, uint8_t * output, uint32_t length)`

calculates the ciphertext/plaintext

Parameters

<i>input</i>	- input data
<i>output</i>	- output data
<i>length</i>	- length of the input data

7.144.3.2 `void ProtocolPP::jzuc::ProcessData (const uint8_t * input, uint32_t length)`

calculates the MAC (icv)

Parameters

<i>input</i>	- input authentication data
<i>length</i>	- length of the input data

7.144.3.3 `void ProtocolPP::jzuc::result (uint8_t * result)`

returns the MAC (icv)

Parameters

<i>result</i>	- ICV of length 4-bytes
---------------	-------------------------

7.144.3.4 unsigned int ProtocolPP::jzuc::result_size() [inline]

returns the length of the MAC (icv)

Returns

- length of the ICV (4 bytes)

The documentation for this class was generated from the following file:

- include/jzuc.h

7.145 option::PrintUsageImplementation::LinePartIterator Class Reference

```
#include <optionparser.h>
```

Public Member Functions

- **LinePartIterator** (const [Descriptor](#) usage[])

Creates an iterator for usage.
- **bool nextTable()**

Moves iteration to the next table (if any). Has to be called once on a new [LinePartIterator](#) to move to the 1st table.
- **void restartTable()**

Reset iteration to the beginning of the current table.
- **bool nextRow()**

Moves iteration to the next row (if any). Has to be called once after each call to [nextTable\(\)](#) to move to the 1st row of the table.
- **void restartRow()**

Reset iteration to the beginning of the current row.
- **bool next()**

Moves iteration to the next part (if any). Has to be called once after each call to [nextRow\(\)](#) to move to the 1st part of the row.
- **int column()**

Returns the index (counting from 0) of the column in which the part pointed to by [data\(\)](#) is located.
- **int line()**

Returns the index (counting from 0) of the line within the current column this part belongs to.
- **int length()**

Returns the length of the part pointed to by [data\(\)](#) in raw chars (not UTF-8 characters).
- **int screenLength()**

Returns the width in screen columns of the part pointed to by [data\(\)](#). Takes multi-byte UTF-8 sequences and wide characters into account.
- **const char * data()**

Returns the current part of the iteration.

7.145.1 Constructor & Destructor Documentation

7.145.1.1 option::PrintUsageImplementation::LinePartIterator (const [Descriptor](#) usage[]) [inline]

Creates an iterator for usage.

7.145.2 Member Function Documentation

7.145.2.1 `int option::PrintUsageImplementation::LinePartIterator::column() [inline]`

Returns the index (counting from 0) of the column in which the part pointed to by `data()` is located.

7.145.2.2 `const char* option::PrintUsageImplementation::LinePartIterator::data() [inline]`

Returns the current part of the iteration.

7.145.2.3 `int option::PrintUsageImplementation::LinePartIterator::length() [inline]`

Returns the length of the part pointed to by `data()` in raw chars (not UTF-8 characters).

7.145.2.4 `int option::PrintUsageImplementation::LinePartIterator::line() [inline]`

Returns the index (counting from 0) of the line within the current column this part belongs to.

7.145.2.5 `bool option::PrintUsageImplementation::LinePartIterator::next() [inline]`

Moves iteration to the next part (if any). Has to be called once after each call to `nextRow()` to move to the 1st part of the row.

Return values

<code>false</code>	if moving to next part failed because no further part exists.
--------------------	---

See [LinePartIterator](#) for details about the iteration.

7.145.2.6 `bool option::PrintUsageImplementation::LinePartIterator::nextRow() [inline]`

Moves iteration to the next row (if any). Has to be called once after each call to `nextTable()` to move to the 1st row of the table.

Return values

<code>false</code>	if moving to next row failed because no further row exists.
--------------------	---

7.145.2.7 `bool option::PrintUsageImplementation::LinePartIterator::nextTable() [inline]`

Moves iteration to the next table (if any). Has to be called once on a new [LinePartIterator](#) to move to the 1st table.

Return values

<code>false</code>	if moving to next table failed because no further table exists.
--------------------	---

7.145.2.8 `void option::PrintUsageImplementation::LinePartIterator::restartRow() [inline]`

Reset iteration to the beginning of the current row.

7.145.2.9 `void option::PrintUsageImplementation::LinePartIterator::restartTable() [inline]`

Reset iteration to the beginning of the current table.

7.145.2.10 int option::PrintUsageImplementation::LinePartIterator::screenLength () [inline]

Returns the width in screen columns of the part pointed to by [data\(\)](#). Takes multi-byte UTF-8 sequences and wide characters into account.

The documentation for this class was generated from the following file:

- jtestbench/include/[optionparser.h](#)

7.146 option::PrintUsageImplementation::LineWrapper Class Reference

```
#include <optionparser.h>
```

Public Member Functions

- void [flush \(IStringWriter &write\)](#)
Writes out all remaining data from the [LineWrapper](#) using `write`. Unlike [process\(\)](#) this method indents all lines including the first and will output a \n at the end (but only if something has been written).
- void [process \(IStringWriter &write, const char *data, int len\)](#)
Process, wrap and output the next piece of data.
- [LineWrapper \(int x1, int x2\)](#)
Constructs a [LineWrapper](#) that wraps its output to fit into screen columns `x1` (incl.) to `x2` (excl.).

7.146.1 Constructor & Destructor Documentation

7.146.1.1 option::PrintUsageImplementation::LineWrapper (int x1, int x2) [inline]

Constructs a [LineWrapper](#) that wraps its output to fit into screen columns `x1` (incl.) to `x2` (excl.).

`x1` gives the indentation [LineWrapper](#) uses if it needs to indent.

7.146.2 Member Function Documentation

7.146.2.1 void option::PrintUsageImplementation::LineWrapper::flush (IStringWriter & write) [inline]

Writes out all remaining data from the [LineWrapper](#) using `write`. Unlike [process\(\)](#) this method indents all lines including the first and will output a \n at the end (but only if something has been written).

7.146.2.2 void option::PrintUsageImplementation::LineWrapper::process (IStringWriter & write, const char * data, int len) [inline]

Process, wrap and output the next piece of data.

[process\(\)](#) will output at least one line of output. This is not necessarily the `data` passed in. It may be data queued from a prior call to [process\(\)](#). If the internal buffer is full, more than 1 line will be output.

[process\(\)](#) assumes that the a proper amount of indentation has already been output. It won't write any further indentation before the 1st line. If more than 1 line is written due to buffer constraints, the lines following the first will be indented by this method, though.

No \n is written by this method after the last line that is written.

Parameters

<i>write</i>	where to write the data.
<i>data</i>	the new chunk of data to write.
<i>len</i>	the length of the chunk of data to write.

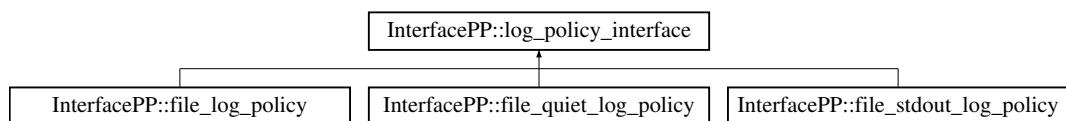
The documentation for this class was generated from the following file:

- jtestbench/include/optionparser.h

7.147 InterfacePP::log_policy_interface Class Reference

```
#include <jlogger.h>
```

Inheritance diagram for InterfacePP::log_policy_interface:



Public Member Functions

- virtual void [open_ostream](#) (const std::string &name)=0
open_ostream - pure virtual function to override in your testbench
- virtual void [close_ostream](#) ()=0
close_ostream - pure virtual function to override in your testbench
- virtual void [write](#) (const std::string &msg)=0
write - pure virtual function to override in your testbench

7.147.1 Member Function Documentation

7.147.1.1 virtual void [InterfacePP::log_policy_interface::close_ostream](#)() [pure virtual]

close_ostream - pure virtual function to override in your testbench

Implemented in [InterfacePP::file_quiet_log_policy](#), [InterfacePP::file_stdout_log_policy](#), and [InterfacePP::file_log_policy](#).

7.147.1.2 virtual void [InterfacePP::log_policy_interface::open_ostream](#)(const std::string & name) [pure virtual]

open_ostream - pure virtual function to override in your testbench

Implemented in [InterfacePP::file_quiet_log_policy](#), [InterfacePP::file_stdout_log_policy](#), and [InterfacePP::file_log_policy](#).

7.147.1.3 virtual void [InterfacePP::log_policy_interface::write](#)(const std::string & msg) [pure virtual]

write - pure virtual function to override in your testbench

Implemented in [InterfacePP::file_quiet_log_policy](#), [InterfacePP::file_stdout_log_policy](#), and [InterfacePP::file_log_policy](#).

The documentation for this class was generated from the following file:

- jlogger/include/jlogger.h

7.148 log_policy_interface Class Reference

7.148.1 Detailed Description

7.148.2 logger class

Supports logging levels for INFO, DEBUG, WARN, ERROR, and FATAL. Can also be colorized by defining COLORIZE in this file. Output will appear as follows

DARK_THEME

- INFO - Light blue (cyan)
- DEBUG - Dark blue
- WARN - Yellow
- ERROR - Red
- FATAL - Magenta

LIGHT_THEME

- INFO - Black
- DEBUG - Dark blue
- WARN - Blue
- ERROR - Red
- FATAL - Magenta

```
0000566 < Mon Dec 18 22:00:53 2017 - 0125000 > ~ <INFO> :jexec::exec() writing to memory addr=0x7fc2180126e0 size=314
0000567 < Mon Dec 18 22:00:53 2017 - 0125000 > ~ <INFO> :jexec::exec() writing to memory finished
0000568 < Mon Dec 18 22:00:53 2017 - 0125000 > ~ <INFO> :jresponder::push() push to oring=0x7fc2180126e0
0000569 < Mon Dec 18 22:00:53 2017 - 0125000 > ~ <INFO> :jproducer::run() state=process packet=t8q15kdg6ya_last PASSED
0000570 < Mon Dec 18 22:00:53 2017 - 0125000 > ~ <INFO> :jexec::exec() read from memory addr=0x7fc218012830 size=1037
0000571 < Mon Dec 18 22:00:53 2017 - 0125000 > ~ <INFO> :jexec::exec() read finished
0000572 < Mon Dec 18 22:00:53 2017 - 0125000 > ~ <INFO> :jexec::exec() processing finished
0000573 < Mon Dec 18 22:00:53 2017 - 0125000 > ~ <INFO> :jexec::exec() processing...
0000574 < Mon Dec 18 22:00:53 2017 - 0125000 > ~ <INFO> :jexec::exec() writing to memory addr=0x7fc218012c50 size=1019
0000575 < Mon Dec 18 22:00:53 2017 - 0125000 > ~ <INFO> :jexec::exec() writing to memory finished
0000576 < Mon Dec 18 22:00:53 2017 - 0125000 > ~ <INFO> :jresponder::push() push to oring=0x7fc218012c50
0000577 < Mon Dec 18 22:00:53 2017 - 0125000 > ~ <INFO> :jproducer::run() state=process packet=xqv6bdjq64 PASSED
0000578 < Mon Dec 18 22:00:53 2017 - 0125000 > ~ <INFO> :jexec::exec() read from memory addr=0x7fc218013140 size=572
0000579 < Mon Dec 18 22:00:54 2017 - 0125000 > ~ <INFO> :jexec::exec() read finished
0000580 < Mon Dec 18 22:00:54 2017 - 0125000 > ~ <INFO> :jexec::exec() processing...
0000581 < Mon Dec 18 22:00:54 2017 - 0125000 > ~ <INFO> :jexec::exec() processing finished
0000582 < Mon Dec 18 22:00:54 2017 - 0125000 > ~ <INFO> :jexec::exec() writing to memory addr=0x7fc218013390 size=554
0000583 < Mon Dec 18 22:00:54 2017 - 0125000 > ~ <INFO> :jexec::exec() writing to memory finished
0000584 < Mon Dec 18 22:00:54 2017 - 0125000 > ~ <INFO> :jresponder::push() push to oring=0x7fc218013390
0000585 < Mon Dec 18 22:00:54 2017 - 0125000 > ~ <INFO> :jproducer::run() state=process packet=wj4pegp4o2s_last PASSED
0000586 < Mon Dec 18 22:00:54 2017 - 0125000 > ~ <INFO> :jresponder::run() responder terminating...
0000587 < Mon Dec 18 22:00:54 2017 - 0125000 > ~ <PASS> :
*****
PPPPPPP AA SSSSS SSSSS EEEEEEE DDDDDDD
PP PP AAAA S S S S EE DD DD
PP PP A A S S EE DD DD
PPPPPPP AAAAAAA SSSSS SSSSS EEEEEEE DD DD
PP AA AA S S S S EE DD DD
PP AA AA SSSSS SSSSS EEEEEEE DDDDDDD
*****
SEED : 15763005553963583039
ERRORS : 0
FLOWS : 0
PACKETS : 61
START : 2017-12-18 22:00:51
FINISH : 2017-12-18 22:00:54
*****
```

```
0000311 < Mon Dec 18 22:00:01 2017 - 0093750 > ~ <INFO> :jresponder::push() push to oring=0x7fca5c0120d0
0000312 < Mon Dec 18 22:00:01 2017 - 0093750 > ~ <WARN> :jproducer::run() state=process status dummy packet=qfdtbpxb6pr received : 0x321405bd expected : 0x32140354
0000313 < Mon Dec 18 22:00:01 2017 - 0093750 > ~ <INFO> :jexec::exec() read from memory addr=0x7fca5c0122b0 size=2372
0000314 < Mon Dec 18 22:00:01 2017 - 0093750 > ~ <INFO> :jexec::exec() read finished
0000315 < Mon Dec 18 22:00:01 2017 - 0093750 > ~ <INFO> :jexec::exec() processing...
0000316 < Mon Dec 18 22:00:01 2017 - 0093750 > ~ <INFO> :jexec::exec() processing finished
0000317 < Mon Dec 18 22:00:01 2017 - 0093750 > ~ <INFO> :jexec::exec() writing to memory addr=0x7fca5c012c00 size=1688
0000318 < Mon Dec 18 22:00:01 2017 - 0093750 > ~ <INFO> :jexec::exec() writing to memory finished
0000319 < Mon Dec 18 22:00:01 2017 - 0093750 > ~ <INFO> :jresponder::push() push to oring=0x7fca5c012c00
0000320 < Mon Dec 18 22:00:01 2017 - 0093750 > ~ <WARN> :jproducer::run() state=process status dummy packet=xxti015hnd_last received : 0x321405bd expected : 0x32140354
0000321 < Mon Dec 18 22:00:01 2017 - 0093750 > ~ <INFO> :jexec::exec() read from memory addr=0x7fca5c013320 size=409
0000322 < Mon Dec 18 22:00:01 2017 - 0093750 > ~ <INFO> :jexec::exec() read finished
0000323 < Mon Dec 18 22:00:01 2017 - 0093750 > ~ <INFO> :jexec::exec() processing...
0000324 < Mon Dec 18 22:00:01 2017 - 0093750 > ~ <INFO> :jexec::exec() processing finished
0000325 < Mon Dec 18 22:00:01 2017 - 0093750 > ~ <INFO> :jexec::exec() writing to memory addr=0x7fca5c013510 size=820
0000326 < Mon Dec 18 22:00:01 2017 - 0093750 > ~ <INFO> :jexec::exec() writing to memory finished
0000327 < Mon Dec 18 22:00:01 2017 - 0093750 > ~ <INFO> :jexec::exec() processing finished
```

```
0000117 < Mon Dec 18 22:00:05 2017 - 0000000 > ~ <INFO> :jresponder::push() push to oring=0x7f2a38001c00
0000118 < Mon Dec 18 22:00:05 2017 - 0000000 > ~ <ERROR> :jproducer::run() state=process packet=nsz53tc799y output length mismatch received : 0 expected : 194
0000119 < Mon Dec 18 22:00:05 2017 - 0000000 > ~ <INFO> :jresponder::dequeue(), dequeued flow=f87nr184rff
0000120 < Mon Dec 18 22:00:05 2017 - 0000000 > ~ <INFO> :jexec::exec() read from memory addr=0x7f2a38001d30 size=203
0000121 < Mon Dec 18 22:00:05 2017 - 0000000 > ~ <INFO> :jexec::exec() read finished
0000122 < Mon Dec 18 22:00:05 2017 - 0000000 > ~ <INFO> :jexec::exec() processing...
0000123 < Mon Dec 18 22:00:05 2017 - 0000000 > ~ <INFO> :jexec::exec() processing finished
0000124 < Mon Dec 18 22:00:05 2017 - 0000000 > ~ <INFO> :jexec::exec() writing to memory addr=0x7f2a38001e10 size=0
0000125 < Mon Dec 18 22:00:05 2017 - 0000000 > ~ <INFO> :jexec::exec() writing to memory finished
0000126 < Mon Dec 18 22:00:05 2017 - 0000000 > ~ <INFO> :jresponder::push() push to oring=0x7f2a38001e10
0000127 < Mon Dec 18 22:00:05 2017 - 0000000 > ~ <ERROR> :jproducer::run() state=process packet=6c08szb7nn output length mismatch received : 0 expected : 200
0000128 < Mon Dec 18 22:00:05 2017 - 0000000 > ~ <INFO> :jresponder::dequeue(), dequeued flow=fv086vgkbph
0000129 < Mon Dec 18 22:00:05 2017 - 0000000 > ~ <INFO> :jexec::exec() read from memory addr=0x7f2a38001f40 size=329
0000130 < Mon Dec 18 22:00:05 2017 - 0000000 > ~ <INFO> :jexec::exec() read finished
0000131 < Mon Dec 18 22:00:05 2017 - 0000000 > ~ <INFO> :jexec::exec() processing...
0000132 < Mon Dec 18 22:00:05 2017 - 0000000 > ~ <INFO> :jexec::exec() processing finished
0000133 < Mon Dec 18 22:00:05 2017 - 0000000 > ~ <INFO> :jexec::exec() writing to memory addr=0x7f2a380020a0 size=0
0000134 < Mon Dec 18 22:00:05 2017 - 0000000 > ~ <INFO> :jexec::exec() writing to memory finished
0000135 < Mon Dec 18 22:00:05 2017 - 0000000 > ~ <INFO> :jresponder::push() push to oring=0x7f2a380020a0
0000136 < Mon Dec 18 22:00:05 2017 - 0000000 > ~ <ERROR> :jproducer::run() state=process packet=mffegq3bqfu output length mismatch received : 0 expected : 326
0000137 < Mon Dec 18 22:00:05 2017 - 0000000 > ~ <INFO> :jresponder::dequeue(), dequeued flow=jrm0tpetbf1
0000138 < Mon Dec 18 22:00:05 2017 - 0000000 > ~ <INFO> :jexec::exec() read from memory addr=0x7f2a38002250 size=316
0000139 < Mon Dec 18 22:00:05 2017 - 0000000 > ~ <INFO> :jexec::exec() read finished
0000140 < Mon Dec 18 22:00:05 2017 - 0000000 > ~ <INFO> :jexec::exec() processing...
0000141 < Mon Dec 18 22:00:05 2017 - 0000000 > ~ <INFO> :jexec::exec() processing finished
```

```
0000515 < Mon Dec 18 22:00:07 2017 - 0015625 > ~ <INFO> :jexec::exec() processing finished
0000516 < Mon Dec 18 22:00:07 2017 - 0015625 > ~ <INFO> :jexec::exec() writing to memory addr=0x7f2a38009e70 size=152
0000517 < Mon Dec 18 22:00:07 2017 - 0015625 > ~ <INFO> :jexec::exec() writing to memory finished
0000518 < Mon Dec 18 22:00:07 2017 - 0015625 > ~ <INFO> :jresponder::push() push to oring=0x7f2a38009e70
0000519 < Mon Dec 18 22:00:07 2017 - 0015625 > ~ <INFO> :jproducer::run() state=process packet=aeqvmal7rnq__last PASSED
0000520 < Mon Dec 18 22:00:07 2017 - 0015625 > ~ <INFO> :jresponder::run() responder terminating...
0000521 < Mon Dec 18 22:00:07 2017 - 0015625 > ~ <ERROR> :*****  
*****  
FFFFFFFFFF AA II LL EEEEEEEE DDDDDDD  
FF AAAA II LL EE DD DD  
FF A A II LL EE DD DD  
FFFFFFFFFF AAAAAAAA II LL EEEEEE DO DD  
FF AA AA II LL EE DO DD  
FF AA AA II LLLL EEEEEE DDDDDDD  
*****  
SEED : 16910110653950264800  
ERRORS : 19  
FLOWS : 0  
PACKETS : 54  
START : 2017-12-18,22:00:05  
FINISH : 2017-12-18,22:00:07  
*****
```

Protocol++ modified by : John Peter Greninger © John Peter Greninger 2015-2019 All Rights Reserved

All copyrights and trademarks are the property of their respective owners

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer
- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution
- Any and all modifications must be returned to John Peter Greninger at GitHub.com <https://github.com/jpgreninger/protocolpp> for evaluation. Inclusion of modifications in the source code shall be determined solely by John Peter Greninger. Failure to provide modifications shall render this license NULL and VOID and revoke any rights to use of Protocol++
- Commercial use requires a fee-based license obtainable at www.protocolpp.com
- Academic use requires written and notarized permission from John Peter and Sheila Rocha Greninger

- US Copyrights at <https://www.copyright.gov/>
 - TXU002059872 (Version 1.0.0)
 - TXU002066632 (Version 1.2.7)
 - TXU002082674 (Version 1.4.0)
 - TXU002097880 (Version 2.0.0)

The names of its contributors may not be used to endorse or promote products derived from this software without specific prior written permission

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE

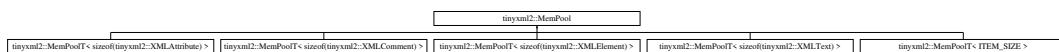
The documentation for this class was generated from the following file:

- [jlogger/include/jlogger.h](#)

7.149 tinyxml2::MemPool Class Reference

```
#include <tinyxml2.h>
```

Inheritance diagram for tinyxml2::MemPool:



Public Member Functions

- [MemPool \(\)](#)
- [virtual ~MemPool \(\)](#)
- [virtual int ItemSize \(\) const =0](#)
- [virtual void * Alloc \(\)=0](#)
- [virtual void Free \(void *\)=0](#)
- [virtual void SetTracked \(\)=0](#)

7.149.1 Constructor & Destructor Documentation

7.149.1.1 [tinyxml2::MemPool::MemPool\(\) \[inline\]](#)

7.149.1.2 [virtual tinyxml2::MemPool::~MemPool\(\) \[inline\], \[virtual\]](#)

7.149.2 Member Function Documentation

7.149.2.1 [virtual void* tinyxml2::MemPool::Alloc\(\) \[pure virtual\]](#)

Implemented in [tinyxml2::MemPoolT< ITEM_SIZE >](#), [tinyxml2::MemPoolT< sizeof\(tinyxml2::XMLComment\) >](#), [tinyxml2::MemPoolT< sizeof\(tinyxml2::XMLText\) >](#), [tinyxml2::MemPoolT< sizeof\(tinyxml2::XmlAttribute\) >](#), and [tinyxml2::MemPoolT< sizeof\(tinyxml2::XMLElement\) >](#).

7.149.2.2 virtual void tinyxml2::MemPool::Free (void *) [pure virtual]

Implemented in `tinyxml2::MemPoolT< ITEM_SIZE >`, `tinyxml2::MemPoolT< sizeof(tinyxml2::XMLComment) >`, `tinyxml2::MemPoolT< sizeof(tinyxml2::XMLText) >`, `tinyxml2::MemPoolT< sizeof(tinyxml2::XmlAttribute) >`, and `tinyxml2::MemPoolT< sizeof(tinyxml2::XMLElement) >`.

7.149.2.3 virtual int tinyxml2::MemPool::ItemSize () const [pure virtual]

Implemented in `tinyxml2::MemPoolT< ITEM_SIZE >`, `tinyxml2::MemPoolT< sizeof(tinyxml2::XMLComment) >`, `tinyxml2::MemPoolT< sizeof(tinyxml2::XMLText) >`, `tinyxml2::MemPoolT< sizeof(tinyxml2::XmlAttribute) >`, and `tinyxml2::MemPoolT< sizeof(tinyxml2::XMLElement) >`.

7.149.2.4 virtual void tinyxml2::MemPool::SetTracked () [pure virtual]

Implemented in `tinyxml2::MemPoolT< ITEM_SIZE >`, `tinyxml2::MemPoolT< sizeof(tinyxml2::XMLComment) >`, `tinyxml2::MemPoolT< sizeof(tinyxml2::XMLText) >`, `tinyxml2::MemPoolT< sizeof(tinyxml2::XmlAttribute) >`, and `tinyxml2::MemPoolT< sizeof(tinyxml2::XMLElement) >`.

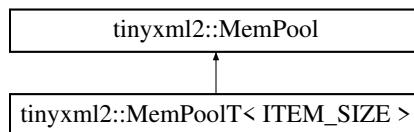
The documentation for this class was generated from the following file:

- `include/tinyxml2.h`

7.150 tinyxml2::MemPoolT< ITEM_SIZE > Class Template Reference

```
#include <tinyxml2.h>
```

Inheritance diagram for `tinyxml2::MemPoolT< ITEM_SIZE >`:



Public Types

- enum { `ITEMS_PER_BLOCK` = $(4 * 1024) / \text{ITEM_SIZE}$ }

Public Member Functions

- `MemPoolT ()`
- `~MemPoolT ()`
- void `Clear ()`
- virtual int `ItemSize () const`
- int `CurrentAllocs () const`
- virtual void * `Alloc ()`
- virtual void `Free (void *mem)`
- void `Trace (const char *name)`
- void `SetTracked ()`
- int `Untracked () const`

7.150.1 Member Enumeration Documentation

7.150.1.1 `template<int ITEM_SIZE> anonymous enum`

Enumerator

ITEMS_PER_BLOCK

7.150.2 Constructor & Destructor Documentation

7.150.2.1 `template<int ITEM_SIZE> tinyxml2::MemPoolT< ITEM_SIZE >::MemPoolT() [inline]`

7.150.2.2 `template<int ITEM_SIZE> tinyxml2::MemPoolT< ITEM_SIZE >::~MemPoolT() [inline]`

7.150.3 Member Function Documentation

7.150.3.1 `template<int ITEM_SIZE> virtual void* tinyxml2::MemPoolT< ITEM_SIZE >::Alloc() [inline], [virtual]`

Implements [tinyxml2::MemPool](#).

7.150.3.2 `template<int ITEM_SIZE> void tinyxml2::MemPoolT< ITEM_SIZE >::Clear() [inline]`

7.150.3.3 `template<int ITEM_SIZE> int tinyxml2::MemPoolT< ITEM_SIZE >::CurrentAllocs() const [inline]`

7.150.3.4 `template<int ITEM_SIZE> virtual void tinyxml2::MemPoolT< ITEM_SIZE >::Free(void * mem) [inline], [virtual]`

Implements [tinyxml2::MemPool](#).

7.150.3.5 `template<int ITEM_SIZE> virtual int tinyxml2::MemPoolT< ITEM_SIZE >::ItemSize() const [inline], [virtual]`

Implements [tinyxml2::MemPool](#).

7.150.3.6 `template<int ITEM_SIZE> void tinyxml2::MemPoolT< ITEM_SIZE >::SetTracked() [inline], [virtual]`

Implements [tinyxml2::MemPool](#).

7.150.3.7 `template<int ITEM_SIZE> void tinyxml2::MemPoolT< ITEM_SIZE >::Trace(const char * name) [inline]`

7.150.3.8 `template<int ITEM_SIZE> int tinyxml2::MemPoolT< ITEM_SIZE >::Untracked() const [inline]`

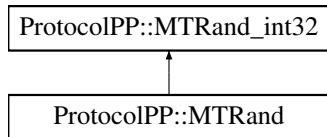
The documentation for this class was generated from the following file:

- [include/tinyxml2.h](#)

7.151 ProtocolPP::MTRand Class Reference

```
#include <mtrand.h>
```

Inheritance diagram for ProtocolPP::MTRand:



Public Member Functions

- [MTRand \(\)](#)
- [MTRand \(unsigned long seed\)](#)
- [MTRand \(const unsigned long *seed, int size\)](#)
- [~MTRand \(\)](#)
- [double operator\(\) \(\)](#)

Additional Inherited Members

7.151.1 Constructor & Destructor Documentation

7.151.1.1 [ProtocolPP::MTRand::MTRand \(\) \[inline\]](#)

7.151.1.2 [ProtocolPP::MTRand::MTRand \(unsigned long seed \) \[inline\]](#)

7.151.1.3 [ProtocolPP::MTRand::MTRand \(const unsigned long * seed, int size \) \[inline\]](#)

7.151.1.4 [ProtocolPP::MTRand::~MTRand \(\) \[inline\]](#)

7.151.2 Member Function Documentation

7.151.2.1 [double ProtocolPP::MTRand::operator\(\) \(\) \[inline\]](#)

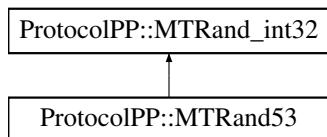
The documentation for this class was generated from the following file:

- [include/mtrand.h](#)

7.152 ProtocolPP::MTRand53 Class Reference

#include <mtrand.h>

Inheritance diagram for ProtocolPP::MTRand53:



Public Member Functions

- [MTRand53 \(\)](#)
- [MTRand53 \(unsigned long seed\)](#)

- `MTRand53` (const unsigned long *`seed`, int `size`)
- `~MTRand53` ()
- double `operator()` ()

Additional Inherited Members

7.152.1 Constructor & Destructor Documentation

7.152.1.1 `ProtocolPP::MTRand53::MTRand53()` [inline]

7.152.1.2 `ProtocolPP::MTRand53::MTRand53(unsigned long seed)` [inline]

7.152.1.3 `ProtocolPP::MTRand53::MTRand53(const unsigned long * seed, int size)` [inline]

7.152.1.4 `ProtocolPP::MTRand53::~MTRand53()` [inline]

7.152.2 Member Function Documentation

7.152.2.1 double `ProtocolPP::MTRand53::operator()` () [inline]

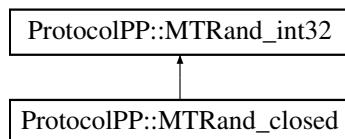
The documentation for this class was generated from the following file:

- include/mtrand.h

7.153 ProtocolPP::MTRand_closed Class Reference

```
#include <mtrand.h>
```

Inheritance diagram for ProtocolPP::MTRand_closed:



Public Member Functions

- `MTRand_closed()`
- `MTRand_closed(unsigned long seed)`
- `MTRand_closed(const unsigned long *seed, int size)`
- `~MTRand_closed()`
- double `operator()` ()

Additional Inherited Members

7.153.1 Constructor & Destructor Documentation

7.153.1.1 `ProtocolPP::MTRand_closed::MTRand_closed()` [inline]

7.153.1.2 `ProtocolPP::MTRand_closed::MTRand_closed(unsigned long seed)` [inline]

7.153.1.3 `ProtocolPP::MTRand_closed::MTRand_closed (const unsigned long * seed, int size)` [inline]

7.153.1.4 `ProtocolPP::MTRand_closed::~MTRand_closed ()` [inline]

7.153.2 Member Function Documentation

7.153.2.1 `double ProtocolPP::MTRand_closed::operator() ()` [inline]

The documentation for this class was generated from the following file:

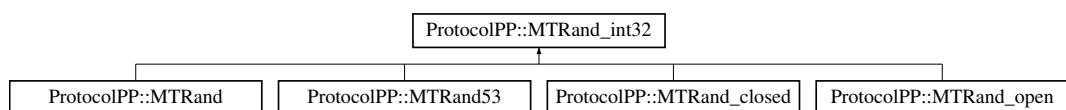
- `include/mtrand.h`

7.154 ProtocolPP::MTRand_int32 Class Reference

Mersenne Twister random number generator.

`#include <mtrand.h>`

Inheritance diagram for ProtocolPP::MTRand_int32:



Public Member Functions

- `MTRand_int32 ()`
default constructor: uses default seed only if this is the first instance
- `MTRand_int32 (unsigned long s)`
constructor with 32 bit int as seed
- `MTRand_int32 (const unsigned long *array, int size)`
constructor with array of size 32 bit ints as seed
- `void seed (unsigned long)`
the two seed functions
- `void seed (const unsigned long *, int size)`
- `unsigned long operator() ()`
overload operator() to make this a generator (functor)
- `virtual ~MTRand_int32 ()`
2007-02-11: made the destructor virtual; thanks "double more" for pointing this out

Protected Member Functions

- `unsigned long rand_int32 ()`

7.154.1 Detailed Description

Mersenne Twister random number generator.

7.154.2 Constructor & Destructor Documentation

7.154.2.1 `ProtocolPP::MTRand_int32::MTRand_int32() [inline]`

default constructor: uses default seed only if this is the first instance

7.154.2.2 `ProtocolPP::MTRand_int32::MTRand_int32(unsigned long s) [inline]`

constructor with 32 bit int as seed

7.154.2.3 `ProtocolPP::MTRand_int32::MTRand_int32(const unsigned long * array, int size) [inline]`

constructor with array of size 32 bit ints as seed

7.154.2.4 `virtual ProtocolPP::MTRand_int32::~MTRand_int32() [inline], [virtual]`

2007-02-11: made the destructor virtual; thanks "double more" for pointing this out

7.154.3 Member Function Documentation

7.154.3.1 `unsigned long ProtocolPP::MTRand_int32::operator()() [inline]`

overload operator() to make this a generator (functor)

7.154.3.2 `unsigned long ProtocolPP::MTRand_int32::rand_int32() [inline], [protected]`

7.154.3.3 `void ProtocolPP::MTRand_int32::seed(unsigned long)`

the two seed functions

7.154.3.4 `void ProtocolPP::MTRand_int32::seed(const unsigned long *, int size)`

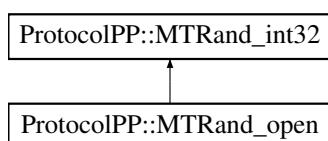
The documentation for this class was generated from the following file:

- [include/mtrand.h](#)

7.155 ProtocolPP::MTRand_open Class Reference

```
#include <mtrand.h>
```

Inheritance diagram for ProtocolPP::MTRand_open:



Public Member Functions

- `MTRand_open()`
- `MTRand_open(unsigned long seed)`
- `MTRand_open(const unsigned long *seed, int size)`
- `~MTRand_open()`
- `double operator()()`

Additional Inherited Members

7.155.1 Constructor & Destructor Documentation

7.155.1.1 `ProtocolPP::MTRand_open::MTRand_open()` [inline]

7.155.1.2 `ProtocolPP::MTRand_open::MTRand_open(unsigned long seed)` [inline]

7.155.1.3 `ProtocolPP::MTRand_open::MTRand_open(const unsigned long *seed, int size)` [inline]

7.155.1.4 `ProtocolPP::MTRand_open::~MTRand_open()` [inline]

7.155.2 Member Function Documentation

7.155.2.1 `double ProtocolPP::MTRand_open::operator()()` [inline]

The documentation for this class was generated from the following file:

- `include/mtrand.h`

7.156 option::Option Class Reference

A parsed option from the command line together with its argument if it has one.

```
#include <optionparser.h>
```

Public Member Functions

- `int type() const`
Returns `Descriptor::type` of this `Option`'s `Descriptor`, or 0 if this `Option` is invalid (unused).
- `int index() const`
Returns `Descriptor::index` of this `Option`'s `Descriptor`, or -1 if this `Option` is invalid (unused).
- `int count()`
Returns the number of times this `Option` (or others with the same `Descriptor::index`) occurs in the argument vector.
- `bool isFirst() const`
Returns true iff this is the first element of the linked list.
- `bool isLast() const`
Returns true iff this is the last element of the linked list.
- `Option * first()`
Returns a pointer to the first element of the linked list.
- `Option * last()`
Returns a pointer to the last element of the linked list.
- `Option * prev()`
Returns a pointer to the previous element of the linked list or NULL if called on `first()`.

- **Option * prevwrap ()**
Returns a pointer to the previous element of the linked list with wrap-around from `first()` to `last()`.
- **Option * next ()**
Returns a pointer to the next element of the linked list or NULL if called on `last()`.
- **Option * nextwrap ()**
Returns a pointer to the next element of the linked list with wrap-around from `last()` to `first()`.
- **void append (Option *new_last)**
Makes `new_last` the new `last()` by chaining it into the list after `last()`.
- **operator const Option * () const**
Casts from `Option` to `const Option*` but only if this `Option` is valid.
- **operator Option * ()**
Casts from `Option` to `Option*` but only if this `Option` is valid.
- **Option ()**
Creates a new `Option` that is a one-element linked list and has NULL `desc`, `name`, `arg` and `namelen`.
- **Option (const Descriptor *desc_, const char *name_, const char *arg_)**
Creates a new `Option` that is a one-element linked list and has the given values for `desc`, `name` and `arg`.
- **void operator= (const Option &orig)**
Makes `*this` a copy of `orig` except for the linked list pointers.
- **Option (const Option &orig)**
Makes `*this` a copy of `orig` except for the linked list pointers.

Public Attributes

- **const Descriptor * desc**
Pointer to this `Option`'s `Descriptor`.
- **const char * name**
The name of the option as used on the command line.
- **const char * arg**
Pointer to this `Option`'s argument (if any).
- **int namelen**
The length of the option `name`.

7.156.1 Detailed Description

A parsed option from the command line together with its argument if it has one.

The `Parser` chains all parsed options with the same `Descriptor::index` together to form a linked list. This allows you to easily implement all of the common ways of handling repeated options and enable/disable pairs.

- Test for presence of a switch in the argument vector:

```
if ( options[QUIET] ) ...
```
- Evaluate –enable-foo/-disable-foo pair where the last one used wins:

```
if ( options[FOO].last() ->type() == DISABLE ) ...
```
- Cumulative option (-v verbose, -vv more verbose, -vvv even more verbose):

```
int verbosity = options[VERBOSE].count();
```
- Iterate over all –file=<fname> arguments:

```
for (Option* opt = options[FILE]; opt; opt = opt->next())
*   fname = opt->arg; ...
```

7.156.2 Constructor & Destructor Documentation

7.156.2.1 option::Option::Option() [inline]

Creates a new [Option](#) that is a one-element linked list and has NULL `desc`, `name`, `arg` and `namelen`.

7.156.2.2 option::Option::Option(const Descriptor * *desc_*, const char * *name_*, const char * *arg_*) [inline]

Creates a new [Option](#) that is a one-element linked list and has the given values for `desc`, `name` and `arg`.

If `name_` points at a character other than '-' it will be assumed to refer to a short option and `namelen` will be set to 1. Otherwise the length will extend to the first '=' character or the string's 0-terminator.

7.156.2.3 option::Option::Option(const Option & *orig*) [inline]

Makes `*this` a copy of `orig` except for the linked list pointers.

After this operation `*this` will be a one-element linked list.

7.156.3 Member Function Documentation

7.156.3.1 void option::Option::append(Option * *new_last*) [inline]

Makes `new_last` the new [last\(\)](#) by chaining it into the list after [last\(\)](#).

It doesn't matter which element you call [append\(\)](#) on. The new element will always be appended to [last\(\)](#).

Attention

`new_last` must not yet be part of a list, or that list will become corrupted, because this method does not unchain `new_last` from an existing list.

7.156.3.2 int option::Option::count() [inline]

Returns the number of times this [Option](#) (or others with the same [Descriptor::index](#)) occurs in the argument vector.

This corresponds to the number of elements in the linked list this [Option](#) is part of. It doesn't matter on which element you call [count\(\)](#). The return value is always the same.

Use this to implement cumulative options, such as -v, -vv, -vvv for different verbosity levels.

Returns 0 when called for an unused/invalid option.

7.156.3.3 Option* option::Option::first() [inline]

Returns a pointer to the first element of the linked list.

Use this when you want the first occurrence of an option on the command line to take precedence. Note that this is not the way most programs handle options. You should probably be using [last\(\)](#) instead.

Note

This method may be called on an unused/invalid option and will return a pointer to the option itself.

7.156.3.4 int option::Option::index() const [inline]

Returns [Descriptor::index](#) of this [Option](#)'s [Descriptor](#), or -1 if this [Option](#) is invalid (unused).

7.156.3.5 `bool option::Option::isFirst() const [inline]`

Returns true iff this is the first element of the linked list.

The first element in the linked list is the first option on the command line that has the respective `Descriptor::index` value.

Returns true for an unused/invalid option.

7.156.3.6 `bool option::Option::isLast() const [inline]`

Returns true iff this is the last element of the linked list.

The last element in the linked list is the last option on the command line that has the respective `Descriptor::index` value.

Returns true for an unused/invalid option.

7.156.3.7 `Option* option::Option::last() [inline]`

Returns a pointer to the last element of the linked list.

Use this when you want the last occurrence of an option on the command line to take precedence. This is the most common way of handling conflicting options.

Note

This method may be called on an unused/invalid option and will return a pointer to the option itself.

Tip:

If you have options with opposite meanings (e.g. `-enable-foo` and `-disable-foo`), you can assign them the same `Descriptor::index` to get them into the same list. Distinguish them by `Descriptor::type` and all you have to do is check `last() ->type()` to get the state listed last on the command line.

7.156.3.8 `Option* option::Option::next() [inline]`

Returns a pointer to the next element of the linked list or NULL if called on `last()`.

If called on `last()` this method returns NULL. Otherwise it will return the option with the same `Descriptor::index` that follows this option on the command line.

7.156.3.9 `Option* option::Option::nextwrap() [inline]`

Returns a pointer to the next element of the linked list with wrap-around from `last()` to `first()`.

If called on `last()` this method returns `first()`. Otherwise it will return the option with the same `Descriptor::index` that follows this option on the command line.

7.156.3.10 `option::Option::operator const Option *() const [inline]`

Casts from `Option` to `const Option*` but only if this `Option` is valid.

If this `Option` is valid (i.e. `desc !=NULL`), returns this. Otherwise returns NULL. This allows testing an `Option` directly in an if-clause to see if it is used:

```
* if (options[CREATE])
* {
*   ...
* }
```

It also allows you to write loops like this:

```
* for (Option* opt = options[FILE]; opt; opt = opt->next())
*   fname = opt->arg; ...
```

7.156.3.11 option::Option::operator Option*()

Casts from [Option](#) to [Option*](#) but only if this [Option](#) is valid.

If this [Option](#) is valid (i.e. `desc!=NULL`), returns this. Otherwise returns `NULL`. This allows testing an [Option](#) directly in an if-clause to see if it is used:

```
* if (options[CREATE])
* {
*   ...
* }
```

It also allows you to write loops like this:

```
* for (Option* opt = options[FILE]; opt; opt = opt->next())
*   fname = opt->arg; ...
```

7.156.3.12 void option::operator=(const Option & orig)

Makes `*this` a copy of `orig` except for the linked list pointers.

After this operation `*this` will be a one-element linked list.

7.156.3.13 Option* option::prev()

Returns a pointer to the previous element of the linked list or `NULL` if called on [first\(\)](#).

If called on [first\(\)](#) this method returns `NULL`. Otherwise it will return the option with the same [Descriptor::index](#) that precedes this option on the command line.

7.156.3.14 Option* option::prevwrap()

Returns a pointer to the previous element of the linked list with wrap-around from [first\(\)](#) to [last\(\)](#).

If called on [first\(\)](#) this method returns [last\(\)](#). Otherwise it will return the option with the same [Descriptor::index](#) that precedes this option on the command line.

7.156.3.15 int option::type()

Returns [Descriptor::type](#) of this [Option](#)'s [Descriptor](#), or 0 if this [Option](#) is invalid (unused).

Because this method (and [last\(\)](#), too) can be used even on unused Options with `desc==0`, you can (provided you arrange your types properly) switch on [type\(\)](#) without testing validity first.

```
* enum OptionType { UNUSED=0, DISABLED=0, ENABLED=1 };
* enum OptionIndex { FOO };
* const Descriptor usage[] = {
*   { FOO, ENABLED, "", "enable-foo", Arg::None, 0 },
*   { FOO, DISABLED, "", "disable-foo", Arg::None, 0 },
*   { 0, 0, 0, 0, 0, 0 } };
* ...
* switch(options[FOO].last()->type()) // no validity check required!
* {
*   case ENABLED: ...
*   case DISABLED: ... // UNUSED==DISABLED !
* }
```

7.156.4 Member Data Documentation

7.156.4.1 const char* option::Option::arg

Pointer to this [Option](#)'s argument (if any).

NULL if this option has no argument. Do not confuse this with the empty string which is a valid argument.

7.156.4.2 const Descriptor* option::Option::desc

Pointer to this [Option](#)'s [Descriptor](#).

Remember that the first dummy descriptor (see [Descriptor::longopt](#)) is used for unknown options.

Attention

`desc==NULL` signals that this [Option](#) is unused. This is the default state of elements in the result array. You don't need to test `desc` explicitly. You can simply write something like this:

```
* if (options[CREATE])
* {
*   ...
* }
```

This works because of `operator const Option*()` .

7.156.4.3 const char* option::Option::name

The name of the option as used on the command line.

The main purpose of this string is to be presented to the user in messages.

In the case of a long option, this is the actual `argv` pointer, i.e. the first character is a '-'. In the case of a short option this points to the option character within the `argv` string.

Note that in the case of a short option group or an attached option argument, this string will contain additional characters following the actual name. Use [namelen](#) to filter out the actual option name only.

7.156.4.4 int option::Option::namelen

The length of the option `name`.

Because `name` points into the actual `argv` string, the option name may be followed by more characters (e.g. other short options in the same short option group). This value is the number of bytes (not characters!) that are part of the actual name.

For a short option, this length is always 1. For a long option this length is always at least 2 if single minus long options are permitted and at least 3 if they are disabled.

Note

In the pathological case of a minus within a short option group (e.g. `-xf-z`), this length is incorrect, because this case will be misinterpreted as a long option and the name will therefore extend to the string's 0-terminator or a following '=' character if there is one. This is irrelevant for most uses of `name` and `namelen`. If you really need to distinguish the case of a long and a short option, compare `name` to the `argv` pointers. A long option's `name` is always identical to one of them, whereas a short option's is never.

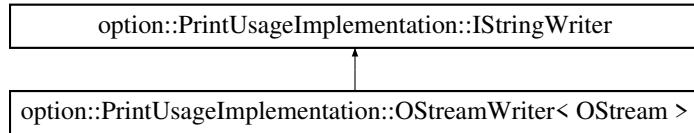
The documentation for this class was generated from the following file:

- [jtestbench/include/optionparser.h](#)

7.157 option::PrintUsageImplementation::OStreamWriter< OStream > Struct Template Reference

```
#include <optionparser.h>
```

Inheritance diagram for option::PrintUsageImplementation::OStreamWriter< OStream >:



Public Member Functions

- virtual void [operator\(\)](#) (const char *str, int size)
Writes the given number of chars beginning at the given pointer somewhere.
- [OStreamWriter](#) (OStream &o)

Public Attributes

- OStream & [ostream](#)

7.157.1 Constructor & Destructor Documentation

7.157.1.1 template<typename OStream> option::PrintUsageImplementation::OStreamWriter< OStream >::OStreamWriter (OStream & o) [inline]

7.157.2 Member Function Documentation

7.157.2.1 template<typename OStream> virtual void option::PrintUsageImplementation::OStreamWriter< OStream >::operator() (const char *, int) [inline], [virtual]

Writes the given number of chars beginning at the given pointer somewhere.

Reimplemented from [option::PrintUsageImplementation::IStringWriter](#).

7.157.3 Member Data Documentation

7.157.3.1 template<typename OStream> OStream& option::PrintUsageImplementation::OStreamWriter< OStream >::ostream

The documentation for this struct was generated from the following file:

- jtestbench/include/[optionparser.h](#)

7.158 option::Parser Class Reference

Checks argument vectors for validity and parses them into data structures that are easier to work with.

```
#include <optionparser.h>
```

Classes

- struct [Action](#)
- class [StoreOptionAction](#)

Public Member Functions

- **Parser ()**
Creates a new Parser.
- **Parser (bool gnu, const Descriptor usage[], int argc, const char **argv, Option options[], Option buffer[], int min_abbr_len=0, bool single_minus_longopt=false, int bufmax=-1)**
Creates a new Parser and immediately parses the given argument vector.
- **Parser (bool gnu, const Descriptor usage[], int argc, char **argv, Option options[], Option buffer[], int min_abbr_len=0, bool single_minus_longopt=false, int bufmax=-1)**
Parser(...) with non-const argv.
- **Parser (const Descriptor usage[], int argc, const char **argv, Option options[], Option buffer[], int min_abbr_len=0, bool single_minus_longopt=false, int bufmax=-1)**
POSIX Parser(...) (gnu==false).
- **Parser (const Descriptor usage[], int argc, char **argv, Option options[], Option buffer[], int min_abbr_len=0, bool single_minus_longopt=false, int bufmax=-1)**
POSIX Parser(...) (gnu==false) with non-const argv.
- **void parse (bool gnu, const Descriptor usage[], int argc, const char **argv, Option options[], Option buffer[], int min_abbr_len=0, bool single_minus_longopt=false, int bufmax=-1)**
Parses the given argument vector.
- **void parse (bool gnu, const Descriptor usage[], int argc, char **argv, Option options[], Option buffer[], int min_abbr_len=0, bool single_minus_longopt=false, int bufmax=-1)**
parse() with non-const argv.
- **void parse (const Descriptor usage[], int argc, const char **argv, Option options[], Option buffer[], int min_abbr_len=0, bool single_minus_longopt=false, int bufmax=-1)**
POSIX parse() (gnu==false).
- **void parse (const Descriptor usage[], int argc, char **argv, Option options[], Option buffer[], int min_abbr_len=0, bool single_minus_longopt=false, int bufmax=-1)**
POSIX parse() (gnu==false) with non-const argv.
- **int optionsCount ()**
Returns the number of valid Option objects in buffer[].
- **int nonOptionsCount ()**
Returns the number of non-option arguments that remained at the end of the most recent parse() that actually encountered non-option arguments.
- **const char ** nonOptions ()**
Returns a pointer to an array of non-option arguments (only valid if nonOptionsCount () >0).
- **const char * nonOption (int i)**
Returns nonOptions () [i] (without checking if i is in range!).
- **bool error ()**
Returns true if an unrecoverable error occurred while parsing options.

Friends

- struct [Stats](#)

7.158.1 Detailed Description

Checks argument vectors for validity and parses them into data structures that are easier to work with.

Example:

```
* int main(int argc, char* argv[])
* {
*     argc-=(argc>0); argv+=(argc>0); // skip program name argv[0] if present
*     option::Stats stats(usage, argc, argv);
*     option::Option options[stats.options_max], buffer[stats.buffer_max];
*     option::Parser parse(usage, argc, argv, options, buffer);
*
*     if (parse.error())
*         return 1;
*
*     if (options[HELP])
*     ...
*
```

7.158.2 Constructor & Destructor Documentation

7.158.2.1 option::Parser::Parser () [inline]

Creates a new [Parser](#).

7.158.2.2 option::Parser::Parser (bool gnu, const Descriptor usage[], int argc, const char ** argv, Option options[], Option buffer[], int min_abbr_len = 0, bool single_minus_longopt = false, int bufmax = -1) [inline]

Creates a new [Parser](#) and immediately parses the given argument vector.

Parameters

<i>gnu</i>	if true, parse() will not stop at the first non-option argument. Instead it will reorder arguments so that all non-options are at the end. This is the default behaviour of GNU getopt() but is not conforming to POSIX. Note, that once the argument vector has been reordered, the <i>gnu</i> flag will have no further effect on this argument vector. So it is enough to pass <i>gnu==true</i> when creating Stats .
<i>usage</i>	Array of Descriptor objects that describe the options to support. The last entry of this array must have 0 in all fields.
<i>argc</i>	The number of elements from <i>argv</i> that are to be parsed. If you pass -1, the number will be determined automatically. In that case the <i>argv</i> list must end with a NULL pointer.
<i>argv</i>	The arguments to be parsed. If you pass -1 as <i>argc</i> the last pointer in the <i>argv</i> list must be NULL to mark the end.
<i>options</i>	Each entry is the first element of a linked list of Options. Each new option that is parsed will be appended to the list specified by that Option 's Descriptor::index . If an entry is not yet used (i.e. the Option is invalid), it will be replaced rather than appended to. The minimum length of this array is the greatest Descriptor::index value that occurs in <i>usage</i> PLUS ONE.
<i>buffer</i>	Each argument that is successfully parsed (including unknown arguments, if they have a Descriptor whose CheckArg does not return ARG_ILLEGAL) will be stored in this array. parse() scans the array for the first invalid entry and begins writing at that index. You can pass <i>bufmax</i> to limit the number of options stored.

<code>min_abbr_len</code>	<p>Passing a value <code>min_abbr_len > 0</code> enables abbreviated long options. The parser will match a prefix of a long option as if it was the full long option (e.g. <code>-foob=10</code> will be interpreted as if it was <code>-foobar=10</code>), as long as the prefix has at least <code>min_abbr_len</code> characters (not counting the <code>-</code>) and is unambiguous.</p> <p>Be careful if combining <code>min_abbr_len=1</code> with <code>single_minus_longopt=true</code> because the ambiguity check does not consider short options and abbreviated single minus long options will take precedence over short options.</p>
<code>single_minus_longopt</code>	<p>Passing <code>true</code> for this option allows long options to begin with a single minus. The double minus form will still be recognized. Note that single minus long options take precedence over short options and short option groups. E.g. <code>-file</code> would be interpreted as <code>-file</code> and not as <code>-f -i -l -e</code> (assuming a long option named "file" exists).</p>
<code>bufmax</code>	<p>The greatest index in the <code>buffer[]</code> array that <code>parse()</code> will write to is <code>bufmax-1</code>. If there are more options, they will be processed (in particular their <code>CheckArg</code> will be called) but not stored.</p> <p>If you used <code>Stats::buffer_max</code> to dimension this array, you can pass <code>-1</code> (or not pass <code>bufmax</code> at all) which tells <code>parse()</code> that the buffer is "large enough".</p>

Attention

Remember that `options` and `buffer` store *Option objects*, not pointers. Therefore it is not possible for the same object to be in both arrays. For those options that are found in both `buffer[]` and `options[]` the respective objects are independent copies. And only the objects in `options[]` are properly linked via `Option::next()` and `Option::prev()`. You can iterate over `buffer[]` to process all options in the order they appear in the argument vector, but if you want access to the other Options with the same `Descriptor::index`, then you *must* access the linked list via `options[]`. You can get the linked list in `options` from a `buffer` object via something like `options[buffer[i].index()]`.

7.158.2.3 `option::Parser::Parser (bool gnu, const Descriptor usage[], int argc, char ** argv, Option options[], Option buffer[], int min_abbr_len = 0, bool single_minus_longopt = false, int bufmax = -1) [inline]`

`Parser(...)` with non-const `argv`.

7.158.2.4 `option::Parser::Parser (const Descriptor usage[], int argc, const char ** argv, Option options[], Option buffer[], int min_abbr_len = 0, bool single_minus_longopt = false, int bufmax = -1) [inline]`

POSIX `Parser(...)` (`gnu==false`).

7.158.2.5 `option::Parser::Parser (const Descriptor usage[], int argc, char ** argv, Option options[], Option buffer[], int min_abbr_len = 0, bool single_minus_longopt = false, int bufmax = -1) [inline]`

POSIX `Parser(...)` (`gnu==false`) with non-const `argv`.

7.158.3 Member Function Documentation

7.158.3.1 `bool option::Parser::error () [inline]`

Returns `true` if an unrecoverable error occurred while parsing options.

An illegal argument to an option (i.e. `CheckArg` returns `ARG_ILLEGAL`) is an unrecoverable error that aborts the parse. Unknown options are only an error if their `CheckArg` function returns `ARG_ILLEGAL`. Otherwise they are collected. In that case if you want to exit the program if either an illegal argument or an unknown option has been passed, use code like this

```
* if (parser.error() || options[UNKNOWN])
*   exit(1);
*
```

7.158.3.2 const char* option::Parser::nonOption(int i) [inline]

Returns [nonOptions\(\)](#) [i] (*without* checking if i is in range!).

7.158.3.3 const char** option::Parser::nonOptions() [inline]

Returns a pointer to an array of non-option arguments (only valid if [nonOptionsCount\(\) >0](#)).

Note

- [parse\(\)](#) does not copy arguments, so this pointer points into the actual argument vector as passed to [parse\(\)](#).
- As explained at [nonOptionsCount\(\)](#) this pointer is only changed by [parse\(\)](#) calls that actually encounter non-option arguments. A [parse\(\)](#) call that encounters only options, will not change [nonOptions\(\)](#).

7.158.3.4 int option::Parser::nonOptionsCount() [inline]

Returns the number of non-option arguments that remained at the end of the most recent [parse\(\)](#) that actually encountered non-option arguments.

Note

A [parse\(\)](#) that does not encounter non-option arguments will leave this value as well as [nonOptions\(\)](#) undisturbed. This means you can feed the [Parser](#) a default argument vector that contains non-option arguments (e.g. a default filename). Then you feed it the actual arguments from the user. If the user has supplied at least one non-option argument, all of the non-option arguments from the default disappear and are replaced by the user's non-option arguments. However, if the user does not supply any non-option arguments the defaults will still be in effect.

7.158.3.5 int option::Parser::optionsCount() [inline]

Returns the number of valid [Option](#) objects in [buffer\[\]](#).

Note

- The returned value always reflects the number of Options in the [buffer\[\]](#) array used for the most recent call to [parse\(\)](#).
- The count (and the [buffer\[\]](#)) includes unknown options if they are collected (see [Descriptor::longopt](#)).

7.158.3.6 void option::Parser::parse(bool gnu, const Descriptor usage[], int argc, const char ** argv, Option options[], Option buffer[], int min_abbr_len = 0, bool single_minus_longopt = false, int bufmax = -1) [inline]

Parses the given argument vector.

Parameters

<i>gnu</i>	if true, parse() will not stop at the first non-option argument. Instead it will reorder arguments so that all non-options are at the end. This is the default behaviour of GNU getopt() but is not conforming to POSIX. Note, that once the argument vector has been reordered, the <i>gnu</i> flag will have no further effect on this argument vector. So it is enough to pass <i>gnu==true</i> when creating Stats .
------------	---

<i>usage</i>	Array of Descriptor objects that describe the options to support. The last entry of this array must have 0 in all fields.
<i>argc</i>	The number of elements from <i>argv</i> that are to be parsed. If you pass -1, the number will be determined automatically. In that case the <i>argv</i> list must end with a NULL pointer.
<i>argv</i>	The arguments to be parsed. If you pass -1 as <i>argc</i> the last pointer in the <i>argv</i> list must be NULL to mark the end.
<i>options</i>	Each entry is the first element of a linked list of Options. Each new option that is parsed will be appended to the list specified by that Option 's Descriptor::index . If an entry is not yet used (i.e. the Option is invalid), it will be replaced rather than appended to. The minimum length of this array is the greatest Descriptor::index value that occurs in <i>usage</i> PLUS ONE.
<i>buffer</i>	Each argument that is successfully parsed (including unknown arguments, if they have a Descriptor whose CheckArg does not return ARG_ILLEGAL) will be stored in this array. parse() scans the array for the first invalid entry and begins writing at that index. You can pass <i>bufmax</i> to limit the number of options stored.
<i>min_abbr_len</i>	Passing a value <i>min_abbr_len</i> > 0 enables abbreviated long options. The parser will match a prefix of a long option as if it was the full long option (e.g. <code>-foob=10</code> will be interpreted as if it was <code>-foobar=10</code>), as long as the prefix has at least <i>min_abbr_len</i> characters (not counting the -) and is unambiguous. Be careful if combining <i>min_abbr_len</i> =1 with <i>single_minus_longopt</i> =true because the ambiguity check does not consider short options and abbreviated single minus long options will take precedence over short options.
<i>single_minus_-longopt</i>	Passing <code>true</code> for this option allows long options to begin with a single minus. The double minus form will still be recognized. Note that single minus long options take precedence over short options and short option groups. E.g. <code>-file</code> would be interpreted as <code>-file</code> and not as <code>-f -i -l -e</code> (assuming a long option named "file" exists).
<i>bufmax</i>	The greatest index in the <i>buffer[]</i> array that parse() will write to is <i>bufmax</i> -1. If there are more options, they will be processed (in particular their CheckArg will be called) but not stored. If you used Stats::buffer_max to dimension this array, you can pass -1 (or not pass <i>bufmax</i> at all) which tells parse() that the buffer is "large enough".

Attention

Remember that *options* and *buffer* store [Option objects](#), not pointers. Therefore it is not possible for the same object to be in both arrays. For those options that are found in both *buffer[]* and *options[]* the respective objects are independent copies. And only the objects in *options[]* are properly linked via [Option::next\(\)](#) and [Option::prev\(\)](#). You can iterate over *buffer[]* to process all options in the order they appear in the argument vector, but if you want access to the other Options with the same [Descriptor::index](#), then you *must* access the linked list via *options[]*. You can get the linked list in *options* from a *buffer* object via something like *options[buffer[i].index()]*.

7.158.3.7 void [option::Parser::parse](#) (bool gnu, const [Descriptor](#) *usage*[], int *argc*, char ** *argv*, [Option](#) *options*[], [Option](#) *buffer*[], int *min_abbr_len* = 0, bool *single_minus_longopt* = false, int *bufmax* = -1) [inline]

[parse\(\)](#) with non-const *argv*.

7.158.3.8 void [option::Parser::parse](#) (const [Descriptor](#) *usage*[], int *argc*, const char ** *argv*, [Option](#) *options*[], [Option](#) *buffer*[], int *min_abbr_len* = 0, bool *single_minus_longopt* = false, int *bufmax* = -1) [inline]

POSIX [parse\(\)](#) (*gnu*==false).

7.158.3.9 void [option::Parser::parse](#) (const [Descriptor](#) *usage*[], int *argc*, char ** *argv*, [Option](#) *options*[], [Option](#) *buffer*[], int *min_abbr_len* = 0, bool *single_minus_longopt* = false, int *bufmax* = -1) [inline]

POSIX [parse\(\)](#) (*gnu*==false) with non-const *argv*.

7.158.4 Friends And Related Function Documentation

7.158.4.1 friend struct Stats [friend]

The documentation for this class was generated from the following file:

- jtestbench/include/[optionparser.h](#)

7.159 option::PrintUsageImplementation Struct Reference

```
#include <optionparser.h>
```

Classes

- struct [FunctionWriter](#)
- struct [IStringWriter](#)
- class [LinePartIterator](#)
- class [LineWrapper](#)
- struct [OStreamWriter](#)
- struct [StreamWriter](#)
- struct [SyscallWriter](#)
- struct [TemporaryWriter](#)

Static Public Member Functions

- static void [upmax](#) (int &i1, int i2)
- static void [indent](#) ([IStringWriter](#) &write, int &x, int want_x)
- static bool [isWideChar](#) (unsigned ch)

Returns true if ch is the unicode code point of a wide character.
- static void [printUsage](#) ([IStringWriter](#) &write, const [Descriptor](#) usage[], int width=80, int last_column_min_percent=50, int last_column_own_line_max_percent=75)

7.159.1 Member Function Documentation

7.159.1.1 static void option::PrintUsageImplementation::indent ([IStringWriter](#) & write, int & x, int want_x) [inline], [static]

7.159.1.2 static bool option::PrintUsageImplementation::isWideChar (unsigned ch) [inline], [static]

Returns true if ch is the unicode code point of a wide character.

Note

The following character ranges are treated as wide

- * 1100..115F
- * 2329..232A (just 2 characters!)
- * 2E80..A4C6 except [for](#) 303F
- * A960..A97C
- * AC00..D7FB
- * F900..FAFF
- * FE10..FE6B
- * FF01..FF60
- * FFE0..FFE6
- * 1B000.....
- *

7.159.1.3 `static void option::PrintUsageImplementation::printUsage (IStringWriter & write, const Descriptor usage[], int width = 80, int last_column_min_percent = 50, int last_column_own_line_max_percent = 75) [inline], [static]`

7.159.1.4 `static void option::PrintUsageImplementation::upmax (int & i1, int i2) [inline], [static]`

The documentation for this struct was generated from the following file:

- [jtestbench/include/optionparser.h](#)

7.160 protocolpp Class Reference

```
#include "schema/protocolpp.xsd"
```

7.160.1 Detailed Description

7.160.2 Protocol++(ProtocolPP) Configuration Schema

For API information:

See Also

- [ProtocolPP::jprotocolpp](#)
- [ProtocolPP::jprotocol](#)
- [ProtocolPP::protocolpp](#)

For Additional Documentation:

See Also

- [jprotocolpp](#)
- [jprotocol](#)
- [protocolpp](#)

Protocol++ written by : John Peter Greninger © John Peter Greninger 2015-2019 All Rights Reserved

All copyrights and trademarks are the property of their respective owners

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer
- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution
- Any and all modifications must be returned to John Peter Greninger at GitHub.com <https://github.com/jpgreninger/protocolpp> for evaluation. Inclusion of modifications in the source code shall be determined solely by John Peter Greninger. Failure to provide modifications shall render this license NULL and VOID and revoke any rights to use of Protocol++
- Commercial use requires a fee-based license obtainable at www.protocolpp.com
- Academic use requires written and notarized permission from John Peter and Sheila Rocha Greninger
- US Copyrights at [https://www.copyright.gov](http://www.copyright.gov)

- TXu002059872 (Version 1.0.0)
- TXu002066632 (Version 1.2.7)
- TXu002082674 (Version 1.4.0)
- TXu002097880 (Version 2.0.0)

The names of its contributors may not be used to endorse or promote products derived from this software without specific prior written permission

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE

The documentation for this class was generated from the following file:

- schema/protocolpp.xsd

7.161 InterfacePP::ringflow Class Reference

```
#include <jring.h>
```

Public Member Functions

- [ringflow](#) (uint64_t address, unsigned int size, std::string name)
- virtual [~ringflow](#) ()
standard deconstructor
- uint64_t [get_addr](#) ()
- unsigned int [get_size](#) ()
- std::string [get_name](#) ()
- void [set_addr](#) (uint64_t address)
- void [set_size](#) (unsigned int size)
- void [set_name](#) (std::string &name)

7.161.1 Constructor & Destructor Documentation

7.161.1.1 [InterfacePP::ringflow::ringflow\(uint64_t address, unsigned int size, std::string name \) \[inline\]](#)

constructor

Parameters

<code>address</code>	- address of flow
<code>size</code>	- size of flow
<code>name</code>	- name of the flow

7.161.1.2 [virtual InterfacePP::ringflow::~ringflow\(\) \[inline\], \[virtual\]](#)

standard deconstructor

7.161.2 Member Function Documentation

7.161.2.1 `uint64_t InterfacePP::ringflow::get_addr() [inline]`

return address

Returns

- address

7.161.2.2 `std::string InterfacePP::ringflow::get_name() [inline]`

return the name of the flow

Returns

- name of the flow

7.161.2.3 `unsigned int InterfacePP::ringflow::get_size() [inline]`

return size

Returns

- size

7.161.2.4 `void InterfacePP::ringflow::set_addr(uint64_t address) [inline]`

set the address

Parameters

<code>address</code>	- address of the input
----------------------	------------------------

7.161.2.5 `void InterfacePP::ringflow::set_name(std::string & name) [inline]`

set the name of the flow

Parameters

<code>name</code>	- name of the flow to add
-------------------	---------------------------

7.161.2.6 `void InterfacePP::ringflow::set_size(unsigned int size) [inline]`

set the size of the input

Parameters

<code>size</code>	- size of the input
-------------------	---------------------

The documentation for this class was generated from the following file:

- [jtestbench/include/jring.h](#)

7.162 InterfacePP::ringin Class Reference

```
#include <jring.h>
```

Public Member Functions

- `ringin (uint64_t address, unsigned int size, std::string stream, uint32_t protect=0, uint64_t outaddr=0, unsigned int outlen=0)`
- virtual `~ringin ()`
standard deconstructor
- `uint64_t get_addr ()`
- `unsigned int get_size ()`
- `uint32_t get_protect ()`
- `std::string get_stream ()`
- `uint64_t get_outaddr ()`
- `unsigned int get_outlen ()`
- `void set_addr (uint64_t address)`
- `void set_size (unsigned int size)`
- `void set_protect (uint32_t protect)`
- `void set_stream (std::string &stream)`
- `void set_outaddr (uint64_t outaddr)`
- `void set_outlen (unsigned int outlen)`

7.162.1 Constructor & Destructor Documentation

7.162.1.1 `InterfacePP::ringin::ringin (uint64_t address, unsigned int size, std::string stream, uint32_t protect = 0, uint64_t outaddr = 0, unsigned int outlen = 0) [inline]`

constructor

Parameters

<code>address</code>	- address of input
<code>size</code>	- size of input
<code>stream</code>	- name of the stream (flow)
<code>protect</code>	- protect (ENCAP) the data
<code>outaddr</code>	- address to write output
<code>outlen</code>	- length of the output memory

7.162.1.2 `virtual InterfacePP::ringin::~ringin () [inline], [virtual]`

standard deconstructor

7.162.2 Member Function Documentation

7.162.2.1 `uint64_t InterfacePP::ringin::get_addr () [inline]`

return address

Returns

- address

7.162.2.2 `uint64_t InterfacePP::ringin::get_outaddr() [inline]`

return the address of the output

Returns

- address of the output

7.162.2.3 `unsigned int InterfacePP::ringin::get_outlen() [inline]`

return the address of the output

Returns

- address of the output

7.162.2.4 `uint32_t InterfacePP::ringin::get_protect() [inline]`

return protection status

Returns

- protection status

7.162.2.5 `unsigned int InterfacePP::ringin::get_size() [inline]`

return size

Returns

- size

7.162.2.6 `std::string InterfacePP::ringin::get_stream() [inline]`

return name of the stream associated with this packet

Returns

- name of the stream

7.162.2.7 `void InterfacePP::ringin::set_addr(uint64_t address) [inline]`

set the address

Parameters

<code>address</code>	- address of the input
----------------------	------------------------

7.162.2.8 `void InterfacePP::ringin::set_outaddr(uint64_t outaddr) [inline]`

set the size of the input

Parameters

<i>outaddr</i>	- address to write output to
----------------	------------------------------

7.162.2.9 void InterfacePP::ringin::set_outlen (unsigned int *outlen*) [inline]

set the size of the output

Parameters

<i>outlen</i>	- length of the output memory
---------------	-------------------------------

7.162.2.10 void InterfacePP::ringin::set_protect (uint32_t *protect*) [inline]

set the size of the input

Parameters

<i>protect</i>	- protect (ENCAP) the data
----------------	----------------------------

7.162.2.11 void InterfacePP::ringin::set_size (unsigned int *size*) [inline]

set the size of the input

Parameters

<i>size</i>	- size of the input
-------------	---------------------

7.162.2.12 void InterfacePP::ringin::set_stream (std::string & *stream*) [inline]

set the name of the stream

Parameters

<i>stream</i>	- name of the stream
---------------	----------------------

The documentation for this class was generated from the following file:

- jtestbench/include/jring.h

7.163 InterfacePP::ringout Class Reference

```
#include <jring.h>
```

Public Member Functions

- [ringout](#) (uint64_t address, unsigned int size, uint32_t status)
- virtual [~ringout](#) ()
standard deconstructor
- uint64_t [get_addr](#) ()
- unsigned int [get_size](#) ()
- uint32_t [get_status](#) ()
- void [set_addr](#) (uint64_t address)
- void [set_size](#) (unsigned int size)
- void [set_status](#) (uint32_t status)

7.163.1 Constructor & Destructor Documentation

7.163.1.1 `InterfacePP::ringout::ringout(uint64_t address, unsigned int size, uint32_t status) [inline]`

constructor

Parameters

<code>address</code>	- address of output
<code>size</code>	- size of output
<code>status</code>	- status of the output

7.163.1.2 `virtual InterfacePP::ringout::~ringout() [inline], [virtual]`

standard deconstructor

7.163.2 Member Function Documentation

7.163.2.1 `uint64_t InterfacePP::ringout::get_addr() [inline]`

return address

Returns

- address

7.163.2.2 `unsigned int InterfacePP::ringout::get_size() [inline]`

return size

Returns

- size

7.163.2.3 `uint32_t InterfacePP::ringout::get_status() [inline]`

retrieve the status of the output

Returns

- 32-bit status word

7.163.2.4 `void InterfacePP::ringout::set_addr(uint64_t address) [inline]`

set the address

Parameters

<code>address</code>	- address of the input
----------------------	------------------------

7.163.2.5 `void InterfacePP::ringout::set_size(unsigned int size) [inline]`

set the size of the input

Parameters

<code>size</code>	- size of the input
-------------------	---------------------

7.163.2.6 `void InterfacePP::ringout::set_status(uint32_t status) [inline]`

set the status word

Parameters

<code>status</code>	- status value to set
---------------------	-----------------------

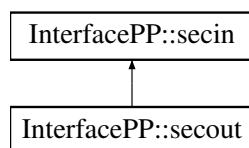
The documentation for this class was generated from the following file:

- `jtestbench/include/jring.h`

7.164 InterfacePP::secin Class Reference

#include <jring.h>

Inheritance diagram for InterfacePP::secin:



Public Member Functions

- `secin(uint64_t address, unsigned int size)`
- virtual `~secin()`
standard deconstructor
- `uint64_t get_addr()`
- `unsigned int get_size()`
- `void set_addr(uint64_t address)`
- `void set_size(unsigned int size)`

7.164.1 Constructor & Destructor Documentation

7.164.1.1 `InterfacePP::secin::secin(uint64_t address, unsigned int size) [inline]`

constructor

Parameters

<code>address</code>	- address of flow
<code>size</code>	- size of flow

7.164.1.2 `virtual InterfacePP::secin::~secin() [inline], [virtual]`

standard deconstructor

7.164.2 Member Function Documentation

7.164.2.1 `uint64_t InterfacePP::secin::get_addr() [inline]`

return address

Returns

- address

7.164.2.2 `unsigned int InterfacePP::secin::get_size() [inline]`

return size

Returns

- size

7.164.2.3 `void InterfacePP::secin::set_addr(uint64_t address) [inline]`

set the address

Parameters

<code>address</code>	- address of the input
----------------------	------------------------

7.164.2.4 `void InterfacePP::secin::set_size(unsigned int size) [inline]`

set the size of the input

Parameters

<code>size</code>	- size of the input
-------------------	---------------------

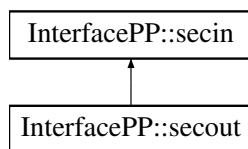
The documentation for this class was generated from the following file:

- [jtestbench/include/jring.h](#)

7.165 InterfacePP::secout Class Reference

#include <jring.h>

Inheritance diagram for InterfacePP::secout:



Public Member Functions

- `secout(uint64_t address, unsigned int size, uint32_t status)`
- virtual `~secout()`

- standard deconstructor*
- `uint32_t get_status ()`
 - `void set_status (uint32_t status)`

7.165.1 Constructor & Destructor Documentation

7.165.1.1 `InterfacePP::secout::secout (uint64_t address, unsigned int size, uint32_t status)` [inline]

constructor

Parameters

<code>address</code>	- address of output
<code>size</code>	- size of output
<code>status</code>	- status of the operation

7.165.1.2 `virtual InterfacePP::secout::~secout ()` [inline], [virtual]

standard deconstructor

7.165.2 Member Function Documentation

7.165.2.1 `uint32_t InterfacePP::secout::get_status ()` [inline]

return the name of the flow

Returns

- name of the flow

7.165.2.2 `void InterfacePP::secout::set_status (uint32_t status)` [inline]

set the status of the packet

Parameters

<code>status</code>	- status of the packet
---------------------	------------------------

The documentation for this class was generated from the following file:

- `jtestbench/include/jring.h`

7.166 ProtocolPP::sfmt Class Reference

```
#include <SFMT.h>
```

Public Member Functions

- `sfmt ()`
- `sfmt (uint32_t seed)`
- `sfmt (uint32_t *init_key, int key_length)`
- `virtual ~sfmt ()`

Standard deconstructor.

- void `sfmt_fill_array8` (uint8_t *array, int size)
- void `sfmt_fill_array16` (uint16_t *array, int size)
- void `sfmt_fill_array32` (uint32_t *array, int size)
- void `sfmt_fill_array64` (uint64_t *array, int size)
- void `sfmt_init_gen_rand` (uint32_t seed)
- void `sfmt_init_by_array` (uint32_t *init_key, int key_length)
- const char * `sfmt_get_idstring` ()
- int `sfmt_get_min_array_size8` ()
- int `sfmt_get_min_array_size16` ()
- int `sfmt_get_min_array_size32` ()
- int `sfmt_get_min_array_size64` ()
- void `sfmt_gen_rand_all` ()
- uint32_t `sfmt_genrand_uint32` ()
- uint64_t `sfmt_genrand_uint64` ()
- double `sfmt_to_real1` (uint32_t v)
- double `sfmt_genrand_real1` ()
- double `sfmt_to_real2` (uint32_t v)
- double `sfmt_genrand_real2` ()
- double `sfmt_to_real3` (uint32_t v)
- double `sfmt_genrand_real3` ()
- double `sfmt_to_res53` (uint64_t v)
- double `sfmt_genrand_res53` ()
- double `sfmt_to_res53_mix` (uint32_t x, uint32_t y)
- double `sfmt_genrand_res53_mix` ()
- int `idxof` (int i)
- void `gen_rand_array` (w128_t *array, int size)
- uint32_t `func1` (uint32_t x)
- uint32_t `func2` (uint32_t x)
- void `period_certification` ()

7.166.1 Constructor & Destructor Documentation

7.166.1.1 ProtocolPP::sfmt::sfmt()

Standard constructor for **SFMT** that initializes with a value of 0x01234567

7.166.1.2 ProtocolPP::sfmt::sfmt(uint32_t seed)

Standard constructor for **SFMT** that initializes with a value of seed

Parameters

<code>seed</code>	- seed to initialize SFMT with
-------------------	---------------------------------------

7.166.1.3 ProtocolPP::sfmt::sfmt(uint32_t * init_key, int key_length)

Standard constructor for **SFMT** that initializes with a value of init_key

Parameters

<code>init_key</code>	- array of uint32_t to initialize SFMT with
-----------------------	--

<i>key_length</i>	- size of the seed array
-------------------	--------------------------

7.166.1.4 virtual ProtocolPP::sfmt::~sfmt() [inline], [virtual]

Standard deconstructor.

7.166.2 Member Function Documentation

7.166.2.1 uint32_t ProtocolPP::sfmt::func1 (uint32_t x) [inline]

This function represents a function used in the initialization by init_by_array

Parameters

<i>x</i>	32-bit integer
----------	----------------

Returns

32-bit integer

7.166.2.2 uint32_t ProtocolPP::sfmt::func2 (uint32_t x) [inline]

This function represents a function used in the initialization by init_by_array

Parameters

<i>x</i>	32-bit integer
----------	----------------

Returns

32-bit integer

7.166.2.3 void ProtocolPP::sfmt::gen_rand_array (w128_t * array, int size) [inline]

This function fills the user-specified array with pseudorandom integers

Parameters

<i>array</i>	an 128-bit array to be filled by pseudorandom numbers
<i>size</i>	number of 128-bit pseudorandom numbers to be generated

7.166.2.4 int ProtocolPP::sfmt::idxof (int i) [inline]

This function simulate a 64-bit index of LITTLE ENDIAN in BIG ENDIAN machine.

Parameters

<i>i</i>	- index to value
----------	------------------

7.166.2.5 void ProtocolPP::sfmt::period_certification ()

This function represents a function used in the initialization by init_by_array

Returns

32-bit integer

7.166.2.6 ProtocolPP::sfmt_fill_array16 (*uint16_t* * *array*, *int size*)

Fills the user specified array with *uint16_t* values from the underlying *w128_t* array

Parameters

<i>array</i>	- array to fill with <i>uint16_t</i>
<i>size</i>	- number of <i>uint16_t</i> values requested

7.166.2.7 ProtocolPP::sfmt_fill_array32 (*uint32_t* * *array*, *int size*)

Fills the user specified array with *uint32_t* values from the underlying *w128_t* array

Parameters

<i>array</i>	- array to fill with <i>uint32_t</i>
<i>size</i>	- number of <i>uint32_t</i> values requested

7.166.2.8 ProtocolPP::sfmt_fill_array64 (*uint64_t* * *array*, *int size*)

Fills the user specified array with *uint64_t* values from the underlying *w128_t* array

Parameters

<i>array</i>	- array to fill with <i>uint64_t</i>
<i>size</i>	- number of <i>uint64_t</i> values requested

7.166.2.9 ProtocolPP::sfmt_fill_array8 (*uint8_t* * *array*, *int size*)

Fills the user specified array with *uint8_t* values from the underlying *w128_t* array

Parameters

<i>array</i>	- array to fill with <i>uint8_t</i>
<i>size</i>	- number of <i>uint8_t</i> values requested

7.166.2.10 void ProtocolPP::sfmt_gen_rand_all ()

This function fills the internal state array with pseudorandom integers

7.166.2.11 double ProtocolPP::sfmt_genrand_real1 () [inline]

generates a random number on [0,1]-real-interval

Returns

double on [0,1]-real-interval

7.166.2.12 double ProtocolPP::sfmt::sfmt_genrand_real2() [inline]

generates a random number on [0,1)-real-interval

Returns

double on [0,1)-real-interval

7.166.2.13 double ProtocolPP::sfmt::sfmt_genrand_real3() [inline]

generates a random number on (0,1)-real-interval

Returns

double on (0,1)-real-interval

7.166.2.14 double ProtocolPP::sfmt::sfmt_genrand_res53() [inline]

generates a random number on [0,1) with 53-bit resolution

Returns

double on [0,1) with 53-bit resolution

7.166.2.15 double ProtocolPP::sfmt::sfmt_genrand_res53_mix() [inline]

generates a random number on [0,1) with 53-bit resolution using two 32bit integers.

Returns

double on [0,1) with 53-bit resolution

7.166.2.16 uint32_t ProtocolPP::sfmt::sfmt_genrand_uint32() [inline]

This function generates and returns 32-bit pseudorandom number. init_gen_rand or init_by_array must be called before this function.

Returns

32-bit pseudorandom number

7.166.2.17 uint64_t ProtocolPP::sfmt::sfmt_genrand_uint64() [inline]

This function generates and returns 64-bit pseudorandom number. init_gen_rand or init_by_array must be called before this function. The function gen_rand64 should not be called after gen_rand32, unless an initialization is again executed.

Returns

64-bit pseudorandom number

7.166.2.18 ProtocolPP::sfmt::sfmt_get_idstring()

Returns unique identifier for this [Sfmt](#) including internal state values

Returns

string representation of this [Sfmt](#)

7.166.2.19 ProtocolPP::sfmt::sfmt_get_min_array_size16()

This function returns the minimum size of array used for fill_array16() function

Returns

minimum size of array used for fill_array16() function.

7.166.2.20 ProtocolPP::sfmt::sfmt_get_min_array_size32()

This function returns the minimum size of array used for fill_array32() function

Returns

minimum size of array used for fill_array32() function.

7.166.2.21 ProtocolPP::sfmt::sfmt_get_min_array_size64()

This function returns the minimum size of array used for fill_array64() function

Returns

minimum size of array used for fill_array64() function.

7.166.2.22 ProtocolPP::sfmt::sfmt_get_min_array_size8()

This function returns the minimum size of array used for fill_array8() function

Returns

minimum size of array used for fill_array8() function.

7.166.2.23 ProtocolPP::sfmt::sfmt_init_by_array(*uint32_t * init_key, int key_length*)

Reseeds the random number generator with the *uint32_t* array

Parameters

<i>init_key</i>	- array to initialize with
<i>key_length</i>	- length of the init array

7.166.2.24 ProtocolPP::sfmt::sfmt_init_gen_rand(*uint32_t seed*)

Reseeds the random number generator with the *uint32_t*

Parameters

seed	- new seed
------	------------

7.166.2.25 double ProtocolPP::sfmt::sfmt_to_real1 (uint32_t v) [inline]

converts an unsigned 32-bit number to a double on [0,1]-real-interval.

Parameters

v	32-bit unsigned integer
---	-------------------------

Returns

double on [0,1]-real-interval

7.166.2.26 double ProtocolPP::sfmt::sfmt_to_real2 (uint32_t v) [inline]

converts an unsigned 32-bit integer to a double on [0,1)-real-interval.

Parameters

v	32-bit unsigned integer
---	-------------------------

Returns

double on [0,1)-real-interval

7.166.2.27 double ProtocolPP::sfmt::sfmt_to_real3 (uint32_t v) [inline]

converts an unsigned 32-bit integer to a double on (0,1)-real-interval.

Parameters

v	32-bit unsigned integer
---	-------------------------

Returns

double on (0,1)-real-interval

7.166.2.28 double ProtocolPP::sfmt::sfmt_to_res53 (uint64_t v) [inline]

converts an unsigned 32-bit integer to double on [0,1) with 53-bit resolution.

Parameters

v	32-bit unsigned integer
---	-------------------------

Returns

double on [0,1)-real-interval with 53-bit resolution.

7.166.2.29 double ProtocolPP::sfmt::sfmt_to_res53_mix (uint32_t x, uint32_t y) [inline]

generates a random number on [0,1) with 53-bit resolution from two 32 bit integers

Parameters

<i>x</i>	- first uint32_t
<i>y</i>	- second uint32_t

Returns

random number on [0,1] with 53-bit resolution

The documentation for this class was generated from the following file:

- [include/SFMT.h](#)

7.167 SFMT Class Reference

SIMD oriented Fast Mersenne Twister(SFMT) pseudorandom number generator using C structure.

```
#include "include/SFMT.h"
```

7.167.1 Detailed Description

SIMD oriented Fast Mersenne Twister(SFMT) pseudorandom number generator using C structure.

```
@author Mutsuo Saito (Hiroshima University)
@author Makoto Matsumoto (The University of Tokyo)
```

```
Copyright (C) 2006, 2007 Mutsuo Saito, Makoto Matsumoto and Hiroshima
University.
Copyright (C) 2012 Mutsuo Saito, Makoto Matsumoto, Hiroshima
University and The University of Tokyo.
All rights reserved.
```

The 3-clause BSD License is applied to this software, see
LICENSE.txt

```
@note We assume that your system has inttypes.h. If your system
doesn't have inttypes.h, you have to typedef uint32_t and uint64_t,
and you have to define PRIu64 and PRIx64 in this file as follows:
```

```
/// typedef unsigned int uint32_t
/// typedef unsigned long long uint64_t
/// #define PRIu64 "llu"
/// #define PRIx64 "llx"
///
```

uint32_t must be exactly 32-bit unsigned integer type (no more, no less), and uint64_t must be exactly 64-bit unsigned integer type. PRIu64 and PRIx64 are used for printf function to print 64-bit unsigned int and 64-bit unsigned int in hexadecimal format.

<center>ported to C++ by : John Peter Greninger • © John Peter Greninger 2015-2019 • All Rights
<center>_{All copyrights and trademarks are the property of their respective owners}</center>

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- * Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer
- * Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution
- * Any and all modifications must be returned to John Peter Greninger at GitHub.com

<https://github.com/jpgreninger/protocolpp> for evaluation. Inclusion of modifications in the source code shall be determined solely by John Peter Greninger. Failure to provide modifications shall render this license NULL and VOID and revoke any rights to use of Protocol++

- * Commercial use requires a fee-based license obtainable at www.protocolpp.com
- * Academic use requires written and notarized permission from John Peter and Sheila Rocha Greninger
- * US Copyrights at <https://www.copyright.gov/>
 - * TXu002059872 (Version 1.0.0)
 - * TXu002066632 (Version 1.2.7)
 - * TXu002082674 (Version 1.4.0)
 - * TXu002097880 (Version 2.0.0)

The names of its contributors may not be used to endorse or promote products derived from this software without specific prior written permission

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE

The documentation for this class was generated from the following file:

- [include/SFMT.h](#)

7.168 PlatformPP::jsec::sgt_t Struct Reference

```
#include <jsec.h>
```

Public Member Functions

- [sgt_t \(\)](#)

Public Attributes

- std::string [chunksize](#)
- unsigned int [ptrsize](#)
- unsigned int [sgtsize](#)
- uint8_t [bufpool](#)
- std::string [offset](#)
- ProtocolPP::endian_t [endianess](#)
- unsigned int [maxentries](#)

7.168.1 Detailed Description

Structure for scatter-gather tables in SEC/CAAM QorIQ or Layerscape processors

Parameters

<i>chunksize</i>	- range of lengths to split the data into when creating a SG table, i.e., "256..1024"
<i>sgtsize</i>	- size of entries (either 8 or 16 for SEC/CAAM)
<i>bufpool</i>	- Buffer pool to use for memory
<i>offset</i>	- size of the offset into data buffers, i.e., "20..100"
<i>maxentries</i>	- maximum number of entries to read in a SG table

7.168.2 Constructor & Destructor Documentation

7.168.2.1 PlatformPP::jsec::sgt_t::sgt_t() [inline]

7.168.3 Member Data Documentation

7.168.3.1 `uint8_t PlatformPP::jsec::sgt_t::bufpool`

7.168.3.2 `std::string PlatformPP::jsec::sgt_t::chunksize`

7.168.3.3 `ProtocolPP::endian_t PlatformPP::jsec::sgt_t::endianess`

7.168.3.4 `unsigned int PlatformPP::jsec::sgt_t::maxentries`

7.168.3.5 `std::string PlatformPP::jsec::sgt_t::offset`

7.168.3.6 `unsigned int PlatformPP::jsec::sgt_t::ptrsize`

7.168.3.7 `unsigned int PlatformPP::jsec::sgt_t::sgtsize`

The documentation for this struct was generated from the following file:

- platforms/SEC/include/jsec.h

7.169 option::Stats Struct Reference

Determines the minimum lengths of the buffer and options arrays used for [Parser](#).

```
#include <optionparser.h>
```

Classes

- class [CountOptionsAction](#)

Public Member Functions

- [Stats \(\)](#)

Creates a [Stats](#) object with counts set to 1 (for the sentinel element).

- [Stats \(bool gnu, const Descriptor usage\[\], int argc, const char **argv, int min_abbr_len=0, bool single_minus_longopt=false\)](#)

Creates a new [Stats](#) object and immediately updates it for the given usage and argument vector. You may pass 0 for argc and/or argv, if you just want to update options_max.

- [Stats \(bool gnu, const Descriptor usage\[\], int argc, char **argv, int min_abbr_len=0, bool single_minus_longopt=false\)](#)

Stats(...) with non-const argv.

- `Stats (const Descriptor usage[], int argc, const char **argv, int min_abbr_len=0, bool single_minus_longopt=false)`
`POSIX Stats(...) (gnu==false).`
- `Stats (const Descriptor usage[], int argc, char **argv, int min_abbr_len=0, bool single_minus_longopt=false)`
`POSIX Stats(...) (gnu==false) with non-const argv.`
- `void add (bool gnu, const Descriptor usage[], int argc, const char **argv, int min_abbr_len=0, bool single_minus_longopt=false)`
`Updates this Stats object for the given usage and argument vector. You may pass 0 for argc and/or argv, if you just want to update options_max.`
- `void add (bool gnu, const Descriptor usage[], int argc, char **argv, int min_abbr_len=0, bool single_minus_longopt=false)`
`add() with non-const argv.`
- `void add (const Descriptor usage[], int argc, const char **argv, int min_abbr_len=0, bool single_minus_longopt=false)`
`POSIX add() (gnu==false).`
- `void add (const Descriptor usage[], int argc, char **argv, int min_abbr_len=0, bool single_minus_longopt=false)`
`POSIX add() (gnu==false) with non-const argv.`

Public Attributes

- `unsigned buffer_max`
`Number of elements needed for a buffer[] array to be used for parsing the same argument vectors that were fed into this Stats object.`
- `unsigned options_max`
`Number of elements needed for an options[] array to be used for parsing the same argument vectors that were fed into this Stats object.`

7.169.1 Detailed Description

Determines the minimum lengths of the buffer and options arrays used for [Parser](#).

Because [Parser](#) doesn't use dynamic memory its output arrays have to be pre-allocated. If you don't want to use fixed size arrays (which may turn out too small, causing command line arguments to be dropped), you can use [Stats](#) to determine the correct sizes. [Stats](#) work cumulative. You can first pass in your default options and then the real options and afterwards the counts will reflect the union.

7.169.2 Constructor & Destructor Documentation

7.169.2.1 option::Stats::Stats () [inline]

Creates a [Stats](#) object with counts set to 1 (for the sentinel element).

7.169.2.2 option::Stats::Stats (bool gnu, const Descriptor usage[], int argc, const char ** argv, int min_abbr_len = 0, bool single_minus_longopt = false) [inline]

Creates a new [Stats](#) object and immediately updates it for the given `usage` and argument vector. You may pass 0 for `argc` and/or `argv`, if you just want to update `options_max`.

Note

The calls to [Stats](#) methods must match the later calls to [Parser](#) methods. See [Parser::parse\(\)](#) for the meaning of the arguments.

7.169.2.3 `option::Stats::Stats (bool gnu, const Descriptor usage[], int argc, char ** argv, int min_abbr_len = 0, bool single_minus_longopt = false) [inline]`

[Stats\(...\)](#) with non-const argv.

7.169.2.4 `option::Stats::Stats (const Descriptor usage[], int argc, const char ** argv, int min_abbr_len = 0, bool single_minus_longopt = false) [inline]`

POSIX [Stats\(...\)](#) (gnu==false).

7.169.2.5 `option::Stats::Stats (const Descriptor usage[], int argc, char ** argv, int min_abbr_len = 0, bool single_minus_longopt = false) [inline]`

POSIX [Stats\(...\)](#) (gnu==false) with non-const argv.

7.169.3 Member Function Documentation

7.169.3.1 `void option::Stats::add (bool gnu, const Descriptor usage[], int argc, const char ** argv, int min_abbr_len = 0, bool single_minus_longopt = false) [inline]`

Updates this [Stats](#) object for the given `usage` and argument vector. You may pass 0 for `argc` and/or `argv`, if you just want to update [options_max](#).

Note

The calls to [Stats](#) methods must match the later calls to [Parser](#) methods. See [Parser::parse\(\)](#) for the meaning of the arguments.

7.169.3.2 `void option::Stats::add (bool gnu, const Descriptor usage[], int argc, char ** argv, int min_abbr_len = 0, bool single_minus_longopt = false) [inline]`

[add\(\)](#) with non-const argv.

7.169.3.3 `void option::Stats::add (const Descriptor usage[], int argc, const char ** argv, int min_abbr_len = 0, bool single_minus_longopt = false) [inline]`

POSIX [add\(\)](#) (gnu==false).

7.169.3.4 `void option::Stats::add (const Descriptor usage[], int argc, char ** argv, int min_abbr_len = 0, bool single_minus_longopt = false) [inline]`

POSIX [add\(\)](#) (gnu==false) with non-const argv.

7.169.4 Member Data Documentation

7.169.4.1 `unsigned option::Stats::buffer_max`

Number of elements needed for a `buffer[]` array to be used for [parsing](#) the same argument vectors that were fed into this [Stats](#) object.

Note

This number is always 1 greater than the actual number needed, to give you a sentinel element.

7.169.4.2 unsigned option::Stats::options_max

Number of elements needed for an `options[]` array to be used for parsing the same argument vectors that were fed into this `Stats` object.

Note

- This number is always 1 greater than the actual number needed, to give you a sentinel element.
- This number depends only on the `usage`, not the argument vectors, because the `options` array needs exactly one slot for each possible `Descriptor::index`.

The documentation for this struct was generated from the following file:

- `jtestbench/include/optionparser.h`

7.170 StdCapture Class Reference

```
#include <StdCapture.h>
```

Static Public Member Functions

- static void `Init ()`
- static void `BeginCapture ()`
- static bool `IsCapturing ()`
- static bool `EndCapture ()`
- static std::string `GetCapture ()`

7.170.1 Detailed Description

Protocol++ written by : John Peter Greninger © John Peter Greninger 2015-2019 All Rights Reserved

All copyrights and trademarks are the property of their respective owners

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer
- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution
- Any and all modifications must be returned to John Peter Greninger at GitHub.com <https://github.com/jpgreninger/protocolpp> for evaluation. Inclusion of modifications in the source code shall be determined solely by John Peter Greninger. Failure to provide modifications shall render this license NULL and VOID and revoke any rights to use of Protocol++
- Commercial use requires a fee-based license obtainable at www.protocolpp.com
- Academic use requires written and notarized permission from John Peter and Sheila Rocha Greninger
- US Copyrights at <https://www.copyright.gov/>
 - TXu002059872 (Version 1.0.0)
 - TXu002066632 (Version 1.2.7)

- TXu002082674 (Version 1.4.0)
- TXu002097880 (Version 2.0.0)

The names of its contributors may not be used to endorse or promote products derived from this software without specific prior written permission

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE

7.170.2 Member Function Documentation

- 7.170.2.1 `static void StdCapture::BeginCapture() [static]`
- 7.170.2.2 `static bool StdCapture::EndCapture() [static]`
- 7.170.2.3 `static std::string StdCapture::GetCapture() [static]`
- 7.170.2.4 `static void StdCapture::Init() [static]`
- 7.170.2.5 `static bool StdCapture::IsCapturing() [static]`

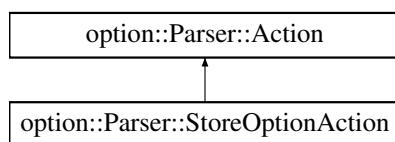
The documentation for this class was generated from the following file:

- drivers/include/[StdCapture.h](#)

7.171 option::Parser::StoreOptionAction Class Reference

```
#include <optionparser.h>
```

Inheritance diagram for option::Parser::StoreOptionAction:



Public Member Functions

- `StoreOptionAction (Parser &parser_, Option options_[], Option buffer_[], int bufmax_)`
Number of slots in buffer. -1 means "large enough".
- `bool perform (Option &option)`
Called by Parser::workhorse() for each Option that has been successfully parsed (including unknown options if they have a Descriptor whose Descriptor::check_arg does not return ARG_ILLEGAL).
- `bool finished (int numargs, const char **args)`
Called by Parser::workhorse() after finishing the parse.

7.171.1 Constructor & Destructor Documentation

7.171.1.1 `option::Parser::StoreOptionAction::StoreOptionAction (Parser & parser_, Option options_[], Option buffer_[], int bufmax_) [inline]`

Number of slots in `buffer`. -1 means "large enough".

Creates a new `StoreOption` action.

Parameters

<code>parser_</code>	the parser whose <code>op_count</code> should be updated.
<code>options_</code>	each <code>Option</code> <code>o</code> is chained into the linked list <code>options_[o.desc->index]</code>
<code>buffer_</code>	each <code>Option</code> is appended to this array as long as there's a free slot.
<code>bufmax_</code>	number of slots in <code>buffer_</code> . -1 means "large enough".

7.171.2 Member Function Documentation

7.171.2.1 `bool option::Parser::StoreOptionAction::finished (int numargs, const char ** args) [inline], [virtual]`

Called by `Parser::workhorse()` after finishing the parse.

Parameters

<code>numargs</code>	the number of non-option arguments remaining
<code>args</code>	pointer to the first remaining non-option argument (if <code>numargs > 0</code>).

Returns

`false` iff a fatal error has occurred.

Reimplemented from [option::Parser::Action](#).

7.171.2.2 `bool option::Parser::StoreOptionAction::perform (Option &) [inline], [virtual]`

Called by `Parser::workhorse()` for each `Option` that has been successfully parsed (including unknown options if they have a `Descriptor` whose `Descriptor::check_arg` does not return `ARG_ILLEGAL`).

Returns `false` iff a fatal error has occurred and the parse should be aborted.

Reimplemented from [option::Parser::Action](#).

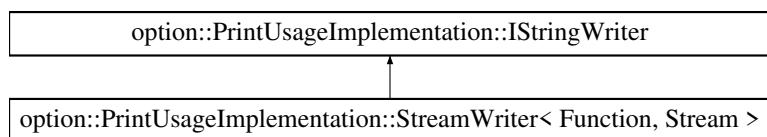
The documentation for this class was generated from the following file:

- jtestbench/include/[optionparser.h](#)

7.172 option::PrintUsageImplementation::StreamWriter< Function, Stream > Struct Template Reference

```
#include <optionparser.h>
```

Inheritance diagram for `option::PrintUsageImplementation::StreamWriter< Function, Stream >`:



Public Member Functions

- virtual void [operator\(\)](#) (const char *str, int size)
Writes the given number of chars beginning at the given pointer somewhere.
- [StreamWriter](#) (Function *w, Stream *s)

Public Attributes

- Function * [fwrite](#)
- Stream * [stream](#)

7.172.1 Constructor & Destructor Documentation

7.172.1.1 template<typename Function, typename Stream> option::PrintUsageImplementation::StreamWriter<Function, Stream >::StreamWriter (Function * w, Stream * s) [inline]

7.172.2 Member Function Documentation

7.172.2.1 template<typename Function, typename Stream> virtual void option::PrintUsageImplementation::StreamWriter<Function, Stream >::operator() (const char *, int) [inline],
[virtual]

Writes the given number of chars beginning at the given pointer somewhere.

Reimplemented from [option::PrintUsageImplementation::IStringWriter](#).

7.172.3 Member Data Documentation

7.172.3.1 template<typename Function, typename Stream> Function* option::PrintUsageImplementation::StreamWriter<Function, Stream >::fwrite

7.172.3.2 template<typename Function, typename Stream> Stream* option::PrintUsageImplementation::StreamWriter<Function, Stream >::stream

The documentation for this struct was generated from the following file:

- jtestbench/include/[optionparser.h](#)

7.173 tinyxml2::StrPair Class Reference

```
#include <tinyxml2.h>
```

Public Types

- enum {
 NEEDS_ENTITY_PROCESSING = 0x01, NEEDS_NEWLINE_NORMALIZATION = 0x02, NEEDS_WHITESPACE_COLLAPSING = 0x04, TEXT_ELEMENT = NEEDS_ENTITY_PROCESSING | NEEDS_NEWLINE_NORMALIZATION,
 TEXT_ELEMENT_LEAVE_ENTITIES = NEEDS_NEWLINE_NORMALIZATION, ATTRIBUTE_NAME = 0, ATTRIBUTE_VALUE = NEEDS_ENTITY_PROCESSING | NEEDS_NEWLINE_NORMALIZATION, ATTRIBUTE_VALUE_LEAVE_ENTITIES = NEEDS_NEWLINE_NORMALIZATION,
 COMMENT = NEEDS_NEWLINE_NORMALIZATION }

Public Member Functions

- `StrPair ()`
- `~StrPair ()`
- `void Set (char *start, char *end, int flags)`
- `const char * GetStr ()`
- `bool Empty () const`
- `void SetInternedStr (const char *str)`
- `void SetStr (const char *str, int flags=0)`
- `char * ParseText (char *in, const char *endTag, int strFlags, int *curLineNumPtr)`
- `char * ParseName (char *in)`
- `void TransferTo (StrPair *other)`
- `void Reset ()`

7.173.1 Member Enumeration Documentation

7.173.1.1 anonymous enum

Enumerator

```
NEEDS_ENTITY_PROCESSING
NEEDS_NEWLINE_NORMALIZATION
NEEDS_WHITESPACE_COLLAPSING
TEXT_ELEMENT
TEXT_ELEMENT_LEAVE_ENTITIES
ATTRIBUTE_NAME
ATTRIBUTE_VALUE
ATTRIBUTE_VALUE_LEAVE_ENTITIES
COMMENT
```

7.173.2 Constructor & Destructor Documentation

7.173.2.1 `tinyxml2::StrPair::StrPair () [inline]`

7.173.2.2 `tinyxml2::StrPair::~StrPair ()`

7.173.3 Member Function Documentation

7.173.3.1 `bool tinyxml2::StrPair::Empty () const [inline]`

7.173.3.2 `const char* tinyxml2::StrPair::GetStr ()`

7.173.3.3 `char* tinyxml2::StrPair::ParseName (char * in)`

7.173.3.4 `char* tinyxml2::StrPair::ParseText (char * in, const char * endTag, int strFlags, int * curLineNumPtr)`

7.173.3.5 `void tinyxml2::StrPair::Reset ()`

7.173.3.6 `void tinyxml2::StrPair::Set (char * start, char * end, int flags) [inline]`

7.173.3.7 `void tinyxml2::StrPair::SetInternedStr (const char * str) [inline]`

7.173.3.8 `void tinyxml2::StrPair::SetStr (const char * str, int flags = 0)`

7.173.3.9 void tinyxml2::StrPair::TransferTo (StrPair * other)

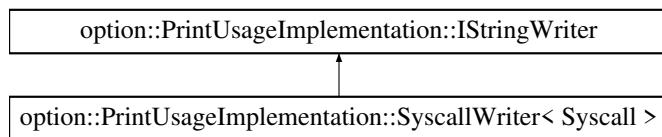
The documentation for this class was generated from the following file:

- [include/tinyxml2.h](#)

7.174 option::PrintUsageImplementation::SyscallWriter< Syscall > Struct Template Reference

#include <optionparser.h>

Inheritance diagram for option::PrintUsageImplementation::SyscallWriter< Syscall >:



Public Member Functions

- virtual void [operator\(\)](#) (const char *str, int size)
Writes the given number of chars beginning at the given pointer somewhere.
- [SyscallWriter](#) (Syscall *w, int f)

Public Attributes

- Syscall * [write](#)
- int [fd](#)

7.174.1 Constructor & Destructor Documentation

7.174.1.1 template<typename Syscall> option::PrintUsageImplementation::SyscallWriter< Syscall >::SyscallWriter (Syscall * w, int f) [inline]

7.174.2 Member Function Documentation

7.174.2.1 template<typename Syscall> virtual void option::PrintUsageImplementation::SyscallWriter< Syscall >::operator() (const char *, int) [inline], [virtual]

Writes the given number of chars beginning at the given pointer somewhere.

Reimplemented from [option::PrintUsageImplementation::IStringWriter](#).

7.174.3 Member Data Documentation

7.174.3.1 template<typename Syscall> int option::PrintUsageImplementation::SyscallWriter< Syscall >::fd

7.174.3.2 template<typename Syscall> Syscall* option::PrintUsageImplementation::SyscallWriter< Syscall >::write

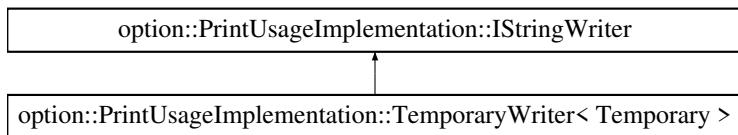
The documentation for this struct was generated from the following file:

- jtestbench/include/[optionparser.h](#)

7.175 option::PrintUsageImplementation::TemporaryWriter< Temporary > Struct Template Reference

#include <optionparser.h>

Inheritance diagram for option::PrintUsageImplementation::TemporaryWriter< Temporary >:



Public Member Functions

- virtual void [operator\(\)](#) (const char *str, int size)
Writes the given number of chars beginning at the given pointer somewhere.
- [TemporaryWriter](#) (const Temporary &u)

Public Attributes

- const Temporary & [userstream](#)

7.175.1 Constructor & Destructor Documentation

7.175.1.1 template<typename Temporary> option::PrintUsageImplementation::TemporaryWriter< Temporary >::TemporaryWriter (const Temporary & u) [inline]

7.175.2 Member Function Documentation

7.175.2.1 template<typename Temporary> virtual void option::PrintUsageImplementation::TemporaryWriter< Temporary >::operator() (const char *, int) [inline], [virtual]

Writes the given number of chars beginning at the given pointer somewhere.

Reimplemented from [option::PrintUsageImplementation::IStringWriter](#).

7.175.3 Member Data Documentation

7.175.3.1 template<typename Temporary> const Temporary& option::PrintUsageImplementation::TemporaryWriter< Temporary >::userstream

The documentation for this struct was generated from the following file:

- jtestbench/include/[optionparser.h](#)

7.176 ProtocolPP::W128_T Union Reference

#include <SFMT.h>

Public Attributes

- `uint32_t u [4]`
- `uint64_t u64 [2]`

7.176.1 Detailed Description

128-bit data structure

7.176.2 Member Data Documentation

7.176.2.1 `uint32_t ProtocolPP::W128_T::u[4]`

7.176.2.2 `uint64_t ProtocolPP::W128_T::u64[2]`

The documentation for this union was generated from the following file:

- `include/SFMT.h`

7.177 wasp Class Reference

```
#include "include/wasp.h"
```

7.177.1 Detailed Description

7.177.2 W.A.S.P

W.A.S.P is an interface to Protocol++ to allow it to connect to a testbench and generate XML files with Protocol++ data. W.A.S.P accepts two input types either PPP (constraint file for random generation of packets) or PROTPP (data file containing packets to be driven into a testbench).

7.177.3 PPP Format

The PPP format (i.e., `input.ppp`) contains only stream, protocol, and security information. W.A.S.P interprets this type of file as either generating an output PROTPP file if the output is provided, or as generating random descriptors for the testbench that will subsequently be driven into the bench after generation.

The PPP has three possible formats as follows. NOTE: For all of these formats, any values not provided by the user are randomized

- FORMAT1 - Format #1 allows full randomization of seed, streams, packets, datalen, protocol, and security association run

```
*  <?xml version='1.0' encoding='UTF-8'?>
*  <protocolpp ver='1.0' xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchemaLocation=
*    <general/>
*    <stream count='5' packets='3' datalen='1500..8192'>
*  </protocolpp>
*
```

- FORMAT2 - Format #2 allows randomization of streams, packets, datalen, and security association for the protocol(s) specified

```

*   <?xml version='1.0' encoding='UTF-8'?>
*   <protocolpp ver='1.0' xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchemaLocation=
*     <general seed='0x00112233' />
*     <stream count='3..5' packets='3..10' datalen='1..50'>
*       <protocol prot='IPV4:SRTP:WIFI' dir='encap'>
*       </protocol>
*     </stream>
*   </protocolpp>
*

```

- FORMAT3 - Format #3 allows randomization of streams, packets, and datalen for the protocol and security association specified

```

*   <?xml version='1.0' encoding='UTF-8'?>
*   <protocolpp ver='1.0' xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchemaLocation=
*     <general seed='0x00112233' />
*     <stream name='test1' count='1' packets='2025' datalen='390'>
*       <protocol name='ipsec'>
*         <security
*           name='ipsecsa'
*           ver='IPV4'
*           dir='ENCAP'
*           mode='tunnel'
*           cipher='aes_cbc'
*           auth='hmac_shal'
*           icvlen='16'
*           usext='false'
*           audit='false'
*           source='44332211'
*           dest='99887766'
*           spi='55331177'
*           seqnum='55331177'
*           hdrlen='80'
*           tfclen='22'
*           ckeylen='16'
*           cipherkey='AAAABBBBCCCCDDDEEEFFFFF00001111'
*           akeylen='16'
*           authkey='AAAABBBBCCCCDDDEEEFFFFF00001111'
*           ivlen='16'
*           iv='AAAABBBBCCCCDDDEEEFFFFF00001111'>
*         </security>
*       </protocol>
*     </stream>
*   </protocolpp>
*

```

7.177.4 PROTPP Format

The PROTPP formation (i.e., input.protpp) contains stream, protocol, security, input, output, and expected values. W.A.S.P interpretes this type of input as the user wants to drive the testbench with the input file.

The PRTOPP format is as follows :

```

*   <?xml version='1.0' encoding='UTF-8'?>
*   <protocolpp ver="1.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchemaLocation=
*     <general seed="1475990354" />
*     <stream name="test1"/>
*     <protocol prot="TLS" file="">
*       <security
*         name="tlsa"
*         dir='ENCAP'
*         ciphersuite="TLS_PSK_WITH_AES_128_CBC_SHA"
*         version="DTLS"
*         type="APPLICATION"
*         epoch="3251"
*         seqnum="15521080929407598078"
*         icvlen="20"
*         ivlen="16"
*         iv="AABBCCDFFFEEDDC0011223344556677"

```

```

*
*      ckeylen="16"
*      cipherkey="AAAABBBCCCCDDDEEEFFFF00001111"
*      akeylen="16"
*      authkey="AAAABBBCCCCDDDEEEFFFF00001111"
*      arlen="0"
*      arwin=""
*      randomiv="0"
*      ivex="1"
*      enthenmac="0"/>
*
```

</protocol>

```

<input name="nahdskdnac" stream="test1" prev="begin" write="vsmubmflmq">
    <data width="1" len="44">
        8FD8D2EC_4B637CB1_2B59B4DF_C50D5DA2
        E8D169BD_0762A0E9_CD6BA43B_C90EAF87
        E78ACA85_FE4B40EF_98AD9251
    </data>
</input>
<output name="vsmubmflmq" stream="test1" len="109"/>
<expect name="qmripfhnm0" stream="test1" read="vsmubmflmq" status="0">
    <data width="1" len="109">
        17FFE00_500CB3F4_DEFF6DFD_FEAABBCC
        DDFFEEDD_CC001122_33445566_77D4199E
        07E9B4C6_749D98D2_EA73A81A_1F39DE73
        557849F4_5DA8480A_70AD9E93_1B25AF6D
        5523F111_E821E61F_36D369C8_30A9A8CE
        BE027F04_7C825383_1A177A2E_8F279575
        B6D4EC0F_6D649AAA_038CA855_16
    </data>
</expect>
<input name="lrmgetesdc" stream="test1" prev="nahdskdnac" write="jmivnieulm">
    <data width="1" len="35">
        A42933AE_5CC6C4ED_49B7E367_B837A204
        0DA6C60B_B6040E27_921DA379_7FDE8843
        F8DF3E
    </data>
</input>
<output name="jmivnieulm" stream="test1" len="93"/>
<expect name="drqtlqemmf" stream="test1" read="jmivnieulm" status="0">
    <data width="1" len="93">
        17FFE00_400CB3F4_DEFF6DFD_FFAABBCC
        DDFFEEDD_CC001122_33445566_77F2297D
        60F8AD09_07578912_A4263A5B_E899EB93
        42629E24_22ED9B61_4C0C789D_96CCC43B
        06C571B2_937EEE32_92D1DBBF_8926B254
        0911F141_2E5377D7_6729B466_F3
    </data>
</expect>
<input name="xbadryuyhd" stream="test1" prev="lrmgetesdc" write="qjwsjajebf">
    <data width="1" len="5">
        491F4ACD_60
    </data>
</input>
<output name="qjwsjajebf" stream="test1" len="61"/>
<expect name="cafhwghrk" stream="test1" read="qjwsjajebf" status="0">
    <data width="1" len="61">
        17FFE00_200CB3F4_DEFF6DFE_00AABBCC
        DDFFEEDD_CC001122_33445566_772AD038
        5EB41522_EAF1BF81_0CC4D40B_E95758A5
        D309BFB1_4D1654C1_7E2DAA51_CC
    </data>
</expect>
</protocolpp>
*

```

7.177.5 Using W.A.S.P

To use W.A.S.P, you can use one of the three constructs to pass in the XML input. W.A.S.P will parse the input and either

- If the input is a PPP file and no output file is passed, will generate random descriptors and then run them

- If the input is a PPP file and output file is passed, will generate a output.protpp file
- In the input is PROTPP file, will run the parsed input

Usage :

```
wasp mywasp(input, output, seed);
```

- input - either a character string representing an XML file, a string representing a filename, or an XMLDocument
- output - a string representing a filename for the output
- seed - seed for the random number generator to reproduce results

For API Documentation:

See Also

[ProtocolPP::jdata](#)
[ProtocolPP::jpacket](#)
[ProtocolPP::jstream](#)
[ProtocolPP::jprotocolpp](#)
[tinyxml2::XMLPrinter](#)
[tinyxml2::XMLVisitor](#)

For Additional Documentation:

See Also

[jdata](#)
[jpacket](#)
[jstream](#)
[jprotocolpp](#)
[tinyxml2::XMLPrinter](#)
[tinyxml2::XMLVisitor](#)

Protocol++ written by : John Peter Greninger © John Peter Greninger 2015-2019 All Rights Reserved

All copyrights and trademarks are the property of their respective owners

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer
- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution
- Any and all modifications must be returned to John Peter Greninger at GitHub.com <https://github.com/jpgreninger/protocolpp> for evaluation. Inclusion of modifications in the source code shall be determined solely by John Peter Greninger. Failure to provide modifications shall render this license NULL and VOID and revoke any rights to use of Protocol++
- Commercial use requires a fee-based license obtainable at www.protocolpp.com
- Academic use requires written and notarized permission from John Peter and Sheila Rocha Greninger
- US Copyrights at <https://www.copyright.gov/>

– TXu002059872 (Version 1.0.0)

- TXu002066632 (Version 1.2.7)
- TXu002082674 (Version 1.4.0)
- TXu002097880 (Version 2.0.0)

The names of its contributors may not be used to endorse or promote products derived from this software without specific prior written permission

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE

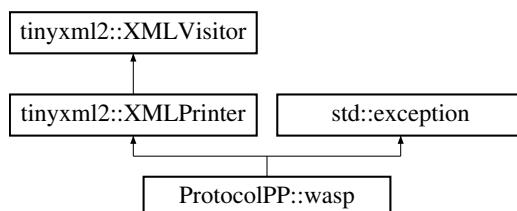
The documentation for this class was generated from the following file:

- [include/wasp.h](#)

7.178 ProtocolPP::wasp Class Reference

```
#include <wasp.h>
```

Inheritance diagram for ProtocolPP::wasp:



Public Member Functions

- [wasp](#) (const char *input, const char *out=0, unsigned long seed=static_cast< unsigned long >(time(NULL)))
- [wasp](#) (std::string &input, const char *out=0, unsigned long seed=static_cast< unsigned long >(time(NULL)))
- [wasp](#) (std::shared_ptr< [tinyxml2::XMLDocument](#) > &input, const char *out=0, unsigned long seed=static_cast< unsigned long >(time(NULL)))
- virtual [~wasp](#) ()
W.A.S.P standard deconstructor.
- void [get](#) (std::shared_ptr< [jdata](#) > &data)

Additional Inherited Members

7.178.1 Constructor & Destructor Documentation

7.178.1.1 ProtocolPP::wasp::wasp (const char * *input*, const char * *out* = 0, unsigned long *seed* = static_cast< unsigned long >(time(NULL)))

W.A.S.P is an interface to Protocol++ that allows the generation of random descriptors and the ability to drive them into a testbench

Parameters

<i>input</i>	- XML data represented in a character string
<i>out</i>	- Filename to write XML output to. If not provided data is written to memory
<i>seed</i>	- default seed for random data

7.178.1.2 `ProtocolPP::wasp::wasp (std::string & input, const char * out = 0, unsigned long seed = static_cast< unsigned long >(time(NULL)))`

W.A.S.P is an interface to Protocol++ that allows the generation of random descriptors and the ability to drive them into a testbench

Parameters

<i>input</i>	- Filename of an XML data file to load
<i>out</i>	- Filename to write XML output to. If not provided data is written to memory
<i>seed</i>	- default seed for random data

7.178.1.3 `ProtocolPP::wasp::wasp (std::shared_ptr< tinyxml2::XMLDocument > & input, const char * out = 0, unsigned long seed = static_cast< unsigned long >(time(NULL)))`

W.A.S.P is an interface to Protocol++ that allows the generation of random descriptors and the ability to drive them into a testbench

Parameters

<i>input</i>	- XML document to load
<i>out</i>	- Filename to write XML output to. If not provided data is written to memory
<i>seed</i>	- default seed for random data

See Also

[tinyxml2](#)

7.178.1.4 `virtual ProtocolPP::wasp::~wasp () [inline], [virtual]`

W.A.S.P standard deconstructor.

7.178.2 Member Function Documentation

7.178.2.1 `void ProtocolPP::wasp::get (std::shared_ptr< jdata > & data)`

Returns

- shared pointer to JDATA object

The documentation for this class was generated from the following file:

- [include/wasp.h](#)

7.179 tinyxml2::XMLAttribute Class Reference

```
#include <tinyxml2.h>
```

Public Member Functions

- const char * **Name** () const
The name of the attribute.
- const char * **Value** () const
The value of the attribute.
- int **GetLineNum** () const
Gets the line number the attribute is in, if the document was parsed from a file.
- const **XMLEAttribute** * **Next** () const
The next attribute in the list.
- int **IntValue** () const
- **ProtocolPP::jarray< uint8_t > ArrayValue** () const
Query as a byte array. Added for ProtocolPP()
- uint8_t **ByteValue** () const
Query as an uint8_t. Added for ProtocolPP()
- uint16_t **ShortValue** () const
Query as an uint16_t. Added for ProtocolPP()
- int64_t **Int64Value** () const
- unsigned **UnsignedValue** () const
Query as an unsigned integer. See [IntValue\(\)](#)
- uint64_t **UnsignedU64Value** () const
Query as an uint64_t. Added for ProtocolPP()
- bool **BoolValue** () const
Query as a boolean. See [IntValue\(\)](#)
- double **DoubleValue** () const
Query as a double. See [IntValue\(\)](#)
- float **FloatValue** () const
Query as a float. See [IntValue\(\)](#)
- **XMLError QueryIntValue** (int *value) const
- **XMLError QueryArrayValue** (**ProtocolPP::jarray< uint8_t >** *value) const
- **XMLError QueryByteValue** (uint8_t *value) const
- **XMLError QueryShortValue** (uint16_t *value) const
- **XMLError QueryUnsignedValue** (unsigned int *value) const
See [QueryIntValue](#).
- **XMLError QueryUnsignedU64Value** (uint64_t *value) const
- **XMLError QueryInt64Value** (int64_t *value) const
See [QueryIntValue](#).
- **XMLError QueryBoolValue** (bool *value) const
See [QueryIntValue](#).
- **XMLError QueryDoubleValue** (double *value) const
See [QueryIntValue](#).
- **XMLError QueryFloatValue** (float *value) const
See [QueryIntValue](#).
- void **SetAttribute** (const char *value)
Set the attribute to a string value.
- void **SetAttribute** (int value)
Set the attribute to value.
- void **SetAttribute** (**ProtocolPP::jarray< uint8_t >** value)
Set the attribute to value. Added for ProtocolPP()
- void **SetAttribute** (uint8_t value)
Set the attribute to value. Added for ProtocolPP()

- void [SetAttribute](#) (uint16_t value)
Set the attribute to value. Added for ProtocolPP()
- void [SetAttribute](#) (unsigned value)
Set the attribute to value.
- void [SetAttribute](#) (int64_t value)
Set the attribute to value.
- void [SetAttribute](#) (uint64_t value)
Set the attribute to value. Added for ProtocolPP()
- void [SetAttribute](#) (bool value)
Set the attribute to value.
- void [SetAttribute](#) (double value)
Set the attribute to value.
- void [SetAttribute](#) (float value)
Set the attribute to value.

Friends

- class [XMLElement](#)

7.179.1 Detailed Description

An attribute is a name-value pair. Elements have an arbitrary number of attributes, each with a unique name.

Note

The attributes are not XMLNodes. You may only query the [Next\(\)](#) attribute in a list.

7.179.2 Member Function Documentation

7.179.2.1 `ProtocolPP::jarray<uint8_t> tinyxml2::XmlAttribute::ArrayValue() const [inline]`

Query as a byte array. Added for ProtocolPP()

7.179.2.2 `bool tinyxml2::XmlAttribute::BoolValue() const [inline]`

Query as a boolean. See [IntValue\(\)](#)

7.179.2.3 `uint8_t tinyxml2::XmlAttribute::ByteValue() const [inline]`

Query as an uint8_t. Added for ProtocolPP()

7.179.2.4 `double tinyxml2::XmlAttribute::DoubleValue() const [inline]`

Query as a double. See [IntValue\(\)](#)

7.179.2.5 `float tinyxml2::XmlAttribute::FloatValue() const [inline]`

Query as a float. See [IntValue\(\)](#)

7.179.2.6 `int tinyxml2::XMLAttribute::GetLineNum() const [inline]`

Gets the line number the attribute is in, if the document was parsed from a file.

7.179.2.7 `int64_t tinyxml2::XMLAttribute::Int64Value() const [inline]`

7.179.2.8 `int tinyxml2::XMLAttribute::IntValue() const [inline]`

`IntValue` interprets the attribute as an integer, and returns the value. If the value isn't an integer, 0 will be returned. There is no error checking; use [QueryIntValue\(\)](#) if you need error checking.

7.179.2.9 `const char* tinyxml2::XMLAttribute::Name() const`

The name of the attribute.

7.179.2.10 `const XMLAttribute* tinyxml2::XMLAttribute::Next() const [inline]`

The next attribute in the list.

7.179.2.11 `XMLError tinyxml2::XMLAttribute::QueryArrayValue(ProtocolPP::jarray< uint8_t > * value) const`

`QueryArrayValue` interprets the attribute as a byte array, and returns the value in the provided array. The function will return `XML_NO_ERROR` on success, and `XML_WRONG_ATTRIBUTE_TYPE` if the conversion is not successful
Added for ProtocolPP()

7.179.2.12 `XMLError tinyxml2::XMLAttribute::QueryBoolValue(bool * value) const`

See [QueryIntValue](#).

7.179.2.13 `XMLError tinyxml2::XMLAttribute::QueryByteValue(uint8_t * value) const`

`QueryByteValue` interprets the attribute as a byte, and returns the value in the provided parameter. The function will return `XML_NO_ERROR` on success, and `XML_WRONG_ATTRIBUTE_TYPE` if the conversion is not successful
Added for ProtocolPP()

7.179.2.14 `XMLError tinyxml2::XMLAttribute::QueryDoubleValue(double * value) const`

See [QueryIntValue](#).

7.179.2.15 `XMLError tinyxml2::XMLAttribute::QueryFloatValue(float * value) const`

See [QueryIntValue](#).

7.179.2.16 `XMLError tinyxml2::XMLAttribute::QueryInt64Value(int64_t * value) const`

See [QueryIntValue](#).

7.179.2.17 XMLError tinyxml2::XmlAttribute::QueryIntValue (int * value) const

QueryIntValue interprets the attribute as an integer, and returns the value in the provided parameter. The function will return XML_SUCCESS on success, and XML_WRONG_ATTRIBUTE_TYPE if the conversion is not successful.

7.179.2.18 XMLError tinyxml2::XmlAttribute::QueryShortValue (uint16_t * value) const

QueryShortValue interprets the attribute as a uint16_t, and returns the value in the provided parameter. The function will return XML_NO_ERROR on success, and XML_WRONG_ATTRIBUTE_TYPE if the conversion is not successful
Added for ProtocolPP()

7.179.2.19 XMLError tinyxml2::XmlAttribute::QueryUnsignedU64Value (uint64_t * value) const

QueryUnsignedLongValue interprets the attribute as a uint64_t, and returns the value in the provided parameter. The function will return XML_NO_ERROR on success, and XML_WRONG_ATTRIBUTE_TYPE if the conversion is not successful
Added for ProtocolPP()

7.179.2.20 XMLError tinyxml2::XmlAttribute::QueryUnsignedValue (unsigned int * value) const

See QueryIntValue.

7.179.2.21 void tinyxml2::XmlAttribute::SetAttribute (const char * value)

Set the attribute to a string value.

7.179.2.22 void tinyxml2::XmlAttribute::SetAttribute (int value)

Set the attribute to value.

7.179.2.23 void tinyxml2::XmlAttribute::SetAttribute (ProtocolPP::jarray< uint8_t > value)

Set the attribute to value. Added for ProtocolPP()

7.179.2.24 void tinyxml2::XmlAttribute::SetAttribute (uint8_t value)

Set the attribute to value. Added for ProtocolPP()

7.179.2.25 void tinyxml2::XmlAttribute::SetAttribute (uint16_t value)

Set the attribute to value. Added for ProtocolPP()

7.179.2.26 void tinyxml2::XmlAttribute::SetAttribute (unsigned value)

Set the attribute to value.

7.179.2.27 void tinyxml2::XmlAttribute::SetAttribute (int64_t value)

Set the attribute to value.

7.179.2.28 void tinyxml2::XMLAttribute::SetAttribute (*uint64_t value*)

Set the attribute to value. Added for ProtocolPP()

7.179.2.29 void tinyxml2::XMLAttribute::SetAttribute (*bool value*)

Set the attribute to value.

7.179.2.30 void tinyxml2::XMLAttribute::SetAttribute (*double value*)

Set the attribute to value.

7.179.2.31 void tinyxml2::XMLAttribute::SetAttribute (*float value*)

Set the attribute to value.

7.179.2.32 uint16_t tinyxml2::XMLAttribute::ShortValue () const [inline]

Query as an uint16_t. Added for ProtocolPP()

7.179.2.33 uint64_t tinyxml2::XMLAttribute::UnsignedU64Value () const [inline]

Query as an uint64_t. Added for ProtocolPP()

7.179.2.34 unsigned tinyxml2::XMLAttribute::UnsignedValue () const [inline]

Query as an unsigned integer. See [IntValue\(\)](#)

7.179.2.35 const char* tinyxml2::XMLAttribute::Value () const

The value of the attribute.

7.179.3 Friends And Related Function Documentation

7.179.3.1 friend class XMLElement [friend]

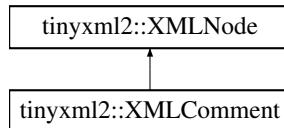
The documentation for this class was generated from the following file:

- [include/tinyxml2.h](#)

7.180 tinyxml2::XMLComment Class Reference

```
#include <tinyxml2.h>
```

Inheritance diagram for tinyxml2::XMLComment:



Public Member Functions

- virtual `XMLComment * ToComment ()`
Safely cast to a Comment, or null.
- virtual const `XMLComment * ToComment () const`
- virtual bool `Accept (XMLVisitor *visitor) const`
- virtual `XMLNode * ShallowClone (XMLDocument *document) const`
- virtual bool `ShallowEqual (const XMLNode *compare) const`

Protected Member Functions

- `XMLComment (XMLDocument *doc)`
- virtual `~XMLComment ()`
- char * `ParseDeep (char *p, StrPair *parentEndTag, int *curLineNumPtr)`

Friends

- class `XMLDocument`

Additional Inherited Members

7.180.1 Detailed Description

An XML Comment.

7.180.2 Constructor & Destructor Documentation

7.180.2.1 `tinyxml2::XMLComment::XMLComment (XMLDocument * doc)` [explicit], [protected]

7.180.2.2 `virtual tinyxml2::XMLComment::~XMLComment ()` [protected], [virtual]

7.180.3 Member Function Documentation

7.180.3.1 `virtual bool tinyxml2::XMLComment::Accept (XMLVisitor * visitor) const` [virtual]

Accept a hierarchical visit of the nodes in the TinyXML-2 DOM. Every node in the XML tree will be conditionally visited and the host will be called back via the `XMLVisitor` interface.

This is essentially a SAX interface for TinyXML-2. (Note however it doesn't re-parse the XML for the callbacks, so the performance of TinyXML-2 is unchanged by using this interface versus any other.)

The interface has been based on ideas from:

- <http://www.saxproject.org/>
- <http://c2.com/cgi/wiki?HierarchicalVisitorPattern>

Which are both good references for "visiting".

An example of using [Accept\(\)](#):

```
XMLPrinter printer;
tinyxmlDoc.Accept( &printer );
const char* xmlcstr = printer.CStr();
```

Implements [tinyxml2::XMLNode](#).

7.180.3.2 `char* tinyxml2::XMLComment::ParseDeep (char * p, StrPair * parentEndTag, int * curLineNumPtr)`
[protected], [virtual]

Reimplemented from [tinyxml2::XMLNode](#).

7.180.3.3 `virtual XMLNode* tinyxml2::XMLComment::ShallowClone (XMLDocument * document) const` [virtual]

Make a copy of this node, but not its children. You may pass in a Document pointer that will be the owner of the new Node. If the 'document' is null, then the node returned will be allocated from the current Document. (`this->GetDocument()`)

Note: if called on a [XMLDocument](#), this will return null.

Implements [tinyxml2::XMLNode](#).

7.180.3.4 `virtual bool tinyxml2::XMLComment::ShallowEqual (const XMLNode * compare) const` [virtual]

Test if 2 nodes are the same, but don't test children. The 2 nodes do not need to be in the same Document.

Note: if called on a [XMLDocument](#), this will return false.

Implements [tinyxml2::XMLNode](#).

7.180.3.5 `virtual XMLComment* tinyxml2::XMLComment::ToComment ()` [inline], [virtual]

Safely cast to a Comment, or null.

Reimplemented from [tinyxml2::XMLNode](#).

7.180.3.6 `virtual const XMLComment* tinyxml2::XMLComment::ToComment () const` [inline], [virtual]

Reimplemented from [tinyxml2::XMLNode](#).

7.180.4 Friends And Related Function Documentation

7.180.4.1 `friend class XMLDocument` [friend]

The documentation for this class was generated from the following file:

- [include/tinyxml2.h](#)

7.181 [tinyxml2::XMLConstHandle](#) Class Reference

```
#include <tinyxml2.h>
```

Public Member Functions

- `XMLConstHandle (const XMLNode *node)`
- `XMLConstHandle (const XMLNode &node)`
- `XMLConstHandle (const XMLConstHandle &ref)`
- `XMLConstHandle & operator= (const XMLConstHandle &ref)`
- `const XMLConstHandle FirstChild () const`
- `const XMLConstHandle FirstChildElement (const char *name=0) const`
- `const XMLConstHandle LastChild () const`
- `const XMLConstHandle LastChildElement (const char *name=0) const`
- `const XMLConstHandle PreviousSibling () const`
- `const XMLConstHandle PreviousSiblingElement (const char *name=0) const`
- `const XMLConstHandle NextSibling () const`
- `const XMLConstHandle NextSiblingElement (const char *name=0) const`
- `const XMLNode * ToNode () const`
- `const XMLElement * ToElement () const`
- `const XMLText * ToText () const`
- `const XMLUnknown * ToUnknown () const`
- `const XMLDeclaration * ToDeclaration () const`

7.181.1 Detailed Description

A variant of the [XMLHandle](#) class for working with const XMLNodes and Documents. It is the same in all regards, except for the 'const' qualifiers. See [XMLHandle](#) for API.

7.181.2 Constructor & Destructor Documentation

7.181.2.1 `tinyxml2::XMLConstHandle::XMLConstHandle (const XMLNode * node) [inline], [explicit]`

7.181.2.2 `tinyxml2::XMLConstHandle::XMLConstHandle (const XMLNode & node) [inline], [explicit]`

7.181.2.3 `tinyxml2::XMLConstHandle::XMLConstHandle (const XMLConstHandle & ref) [inline]`

7.181.3 Member Function Documentation

7.181.3.1 `const XMLConstHandle tinyxml2::XMLConstHandle::FirstChild () const [inline]`

7.181.3.2 `const XMLConstHandle tinyxml2::XMLConstHandle::FirstChildElement (const char * name = 0) const [inline]`

7.181.3.3 `const XMLConstHandle tinyxml2::XMLConstHandle::LastChild () const [inline]`

7.181.3.4 `const XMLConstHandle tinyxml2::XMLConstHandle::LastChildElement (const char * name = 0) const [inline]`

7.181.3.5 `const XMLConstHandle tinyxml2::XMLConstHandle::NextSibling () const [inline]`

7.181.3.6 `const XMLConstHandle tinyxml2::XMLConstHandle::NextSiblingElement (const char * name = 0) const [inline]`

7.181.3.7 `XMLConstHandle& tinyxml2::XMLConstHandle::operator= (const XMLConstHandle & ref) [inline]`

7.181.3.8 `const XMLConstHandle tinyxml2::XMLConstHandle::PreviousSibling () const [inline]`

- 7.181.3.9 `const XMLConstHandle tinyxml2::XMLConstHandle::PreviousSiblingElement (const char * name = 0) const [inline]`
- 7.181.3.10 `const XMLDeclaration* tinyxml2::XMLConstHandle::ToDeclaration () const [inline]`
- 7.181.3.11 `const XMLElement* tinyxml2::XMLConstHandle::ToElement () const [inline]`
- 7.181.3.12 `const XMLNode* tinyxml2::XMLConstHandle::ToNode () const [inline]`
- 7.181.3.13 `const XMLText* tinyxml2::XMLConstHandle::ToText () const [inline]`
- 7.181.3.14 `const XMLUnknown* tinyxml2::XMLConstHandle::ToUnknown () const [inline]`

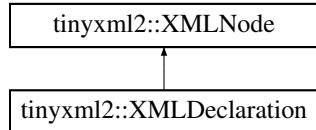
The documentation for this class was generated from the following file:

- [include/tinyxml2.h](#)

7.182 tinyxml2::XMLDeclaration Class Reference

```
#include <tinyxml2.h>
```

Inheritance diagram for tinyxml2::XMLDeclaration:



Public Member Functions

- `virtual XMLDeclaration * ToDeclaration ()`
Safely cast to a Declaration, or null.
- `virtual const XMLDeclaration * ToDeclaration () const`
- `virtual bool Accept (XMLVisitor *visitor) const`
- `virtual XMLNode * ShallowClone (XMLDocument *document) const`
- `virtual bool ShallowEqual (const XMLNode *compare) const`

Protected Member Functions

- `XMLDeclaration (XMLDocument *doc)`
- `virtual ~XMLDeclaration ()`
- `char * ParseDeep (char *p, StrPair *parentEndTag, int *curLineNumPtr)`

Friends

- `class XMLDocument`

Additional Inherited Members

7.182.1 Detailed Description

In correct XML the declaration is the first entry in the file.

```
<?xml version="1.0" standalone="yes"?>
```

TinyXML-2 will happily read or write files without a declaration, however.

The text of the declaration isn't interpreted. It is parsed and written as a string.

7.182.2 Constructor & Destructor Documentation

7.182.2.1 `tinyxml2::XMLDeclaration::XMLDeclaration (XMLDocument * doc) [explicit], [protected]`

7.182.2.2 `virtual tinyxml2::XMLDeclaration::~XMLDeclaration () [protected], [virtual]`

7.182.3 Member Function Documentation

7.182.3.1 `virtual bool tinyxml2::XMLDeclaration::Accept (XMLVisitor * visitor) const [virtual]`

Accept a hierarchical visit of the nodes in the TinyXML-2 DOM. Every node in the XML tree will be conditionally visited and the host will be called back via the [XMLVisitor](#) interface.

This is essentially a SAX interface for TinyXML-2. (Note however it doesn't re-parse the XML for the callbacks, so the performance of TinyXML-2 is unchanged by using this interface versus any other.)

The interface has been based on ideas from:

- <http://www.saxproject.org/>
- <http://c2.com/cgi/wiki?HierarchicalVisitorPattern>

Which are both good references for "visiting".

An example of using [Accept\(\)](#):

```
XMLPrinter printer;
tinyxmlDoc.Accept( &printer );
const char* xmlcstr = printer.CStr();
```

Implements [tinyxml2::XMLNode](#).

7.182.3.2 `char* tinyxml2::XMLDeclaration::ParseDeep (char * p, StrPair * parentEndTag, int * curLineNumPtr) [protected], [virtual]`

Reimplemented from [tinyxml2::XMLNode](#).

7.182.3.3 `virtual XMLNode* tinyxml2::XMLDeclaration::ShallowClone (XMLDocument * document) const [virtual]`

Make a copy of this node, but not its children. You may pass in a Document pointer that will be the owner of the new Node. If the 'document' is null, then the node returned will be allocated from the current Document. (`this->GetDocument()`)

Note: if called on a [XMLDocument](#), this will return null.

Implements [tinyxml2::XMLNode](#).

7.182.3.4 virtual bool tinyxml2::XMLDeclaration::ShallowEqual (const XMLNode * *compare*) const [virtual]

Test if 2 nodes are the same, but don't test children. The 2 nodes do not need to be in the same Document.

Note: if called on a [XMLDocument](#), this will return false.

Implements [tinyxml2::XMLNode](#).

7.182.3.5 virtual XMLDeclaration* tinyxml2::XMLDeclaration::ToDeclaration () [inline], [virtual]

Safely cast to a Declaration, or null.

Reimplemented from [tinyxml2::XMLNode](#).

7.182.3.6 virtual const XMLDeclaration* tinyxml2::XMLDeclaration::ToDeclaration () const [inline], [virtual]

Reimplemented from [tinyxml2::XMLNode](#).

7.182.4 Friends And Related Function Documentation

7.182.4.1 friend class [XMLDocument](#) [friend]

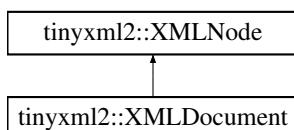
The documentation for this class was generated from the following file:

- [include/tinyxml2.h](#)

7.183 tinyxml2::XMLDocument Class Reference

```
#include <tinyxml2.h>
```

Inheritance diagram for [tinyxml2::XMLDocument](#):



Public Member Functions

- [XMLDocument](#) (bool processEntities=true, [Whitespace](#) whitespaceMode=[PRESERVE_WHITESPACE](#))
constructor
- [~XMLDocument](#) ()
- virtual [XMLDocument](#) * [ToDocument](#) ()
Safely cast to a Document, or null.
- virtual const [XMLDocument](#) * [ToDocument](#) () const
- [XMLError Parse](#) (const char *xml, size_t nBytes=(size_t)(-1))
- [XMLError LoadFile](#) (const char *filename)
- [XMLError LoadFile](#) (FILE *)
- [XMLError SaveFile](#) (const char *filename, bool compact=false)
- [XMLError SaveFile](#) (FILE *fp, bool compact=false)
- bool [ProcessEntities](#) () const
- [Whitespace WhitespaceMode](#) () const

- bool [HasBOM](#) () const
- void [SetBOM](#) (bool useBOM)
- [XMLElement](#) * [RootElement](#) ()
- const [XMLElement](#) * [RootElement](#) () const
- void [Print](#) ([XMLPrinter](#) *streamer=0) const
- virtual bool [Accept](#) ([XMLVisitor](#) *visitor) const
- [XMLElement](#) * [NewElement](#) (const char *name)
- [XMLComment](#) * [NewComment](#) (const char *comment)
- [XMLText](#) * [NewText](#) (const char *text)
- [XMLDeclaration](#) * [NewDeclaration](#) (const char *text=0)
- [XMLUnknown](#) * [NewUnknown](#) (const char *text)
- void [DeleteNode](#) ([XMLNode](#) *node)
- void [ClearError](#) ()
- bool [Error](#) () const

Return true if there was an error parsing the document.
- [XMLError](#) [ErrorID](#) () const

Return the errorID.

 - const char * [ErrorName](#) () const
 - const char * [ErrorStr](#) () const
 - void [PrintError](#) () const

A (trivial) utility function that prints the [ErrorStr](#)) to stdout.
 - int [ErrorLineNum](#) () const

Return the line where the error occurred, or zero if unknown.
 - void [Clear](#) ()

Clear the document, resetting it to the initial state.
- void [DeepCopy](#) ([XMLDocument](#) *target) const
- char * [Identify](#) (char *p, [XMLNode](#) **node)
- void [MarkInUse](#) ([XMLNode](#) *)
- virtual [XMLNode](#) * [ShallowClone](#) ([XMLDocument](#) *) const
- virtual bool [ShallowEqual](#) (const [XMLNode](#) *) const

Static Public Member Functions

- static const char * [ErrorIDToName](#) ([XMLError](#) errorID)

Friends

- class [XMLElement](#)
- class [XMLNode](#)
- class [XMLText](#)
- class [XMLComment](#)
- class [XMLDeclaration](#)
- class [XMLUnknown](#)

Additional Inherited Members

7.183.1 Detailed Description

A Document binds together all the functionality. It can be saved, loaded, and printed to the screen. All Nodes are connected and allocated to a Document. If the Document is deleted, all its Nodes are also deleted.

7.183.2 Constructor & Destructor Documentation

7.183.2.1 `tinyxml2::XMLDocument::XMLDocument(bool processEntities = true, Whitespace whitespaceMode = PRESERVE_WHITESPACE)`

constructor

7.183.2.2 `tinyxml2::XMLDocument::~XMLDocument()`

7.183.3 Member Function Documentation

7.183.3.1 `virtual bool tinyxml2::XMLDocument::Accept(XMLVisitor * visitor) const [virtual]`

Accept a hierarchical visit of the nodes in the TinyXML-2 DOM. Every node in the XML tree will be conditionally visited and the host will be called back via the [XMLVisitor](#) interface.

This is essentially a SAX interface for TinyXML-2. (Note however it doesn't re-parse the XML for the callbacks, so the performance of TinyXML-2 is unchanged by using this interface versus any other.)

The interface has been based on ideas from:

- <http://www.saxproject.org/>
- <http://c2.com/cgi/wiki?HierarchicalVisitorPattern>

Which are both good references for "visiting".

An example of using [Accept\(\)](#):

```
XMLPrinter printer;
tinyxmlDoc.Accept( &printer );
const char* xmlcstr = printer.CStr();
```

Implements [tinyxml2::XMLNode](#).

7.183.3.2 `void tinyxml2::XMLDocument::Clear()`

Clear the document, resetting it to the initial state.

7.183.3.3 `void tinyxml2::XMLDocument::ClearError() [inline]`

7.183.3.4 `void tinyxml2::XMLDocument::DeepCopy(XMLDocument * target) const`

Copies this document to a target document. The target will be completely cleared before the copy. If you want to copy a sub-tree, see [XMLNode::DeepClone\(\)](#).

NOTE: that the 'target' must be non-null.

7.183.3.5 `void tinyxml2::XMLDocument::DeleteNode(XMLNode * node)`

Delete a node associated with this document. It will be unlinked from the DOM.

7.183.3.6 `bool tinyxml2::XMLDocument::Error() const [inline]`

Return true if there was an error parsing the document.

7.183.3.7 **XMLError** `tinyxml2::XMLDocument::ErrorID() const` [inline]

Return the errorID.

7.183.3.8 **static const char*** `tinyxml2::XMLDocument::ErrorIDToName(XMLError errorID)` [static]

7.183.3.9 **int** `tinyxml2::XMLDocument::ErrorLineNum() const` [inline]

Return the line where the error occurred, or zero if unknown.

7.183.3.10 **const char*** `tinyxml2::XMLDocument::ErrorName() const`

7.183.3.11 **const char*** `tinyxml2::XMLDocument::ErrorStr() const`

Returns a "long form" error description. A hopefully helpful diagnostic with location, line number, and/or additional info.

7.183.3.12 **bool** `tinyxml2::XMLDocument::HasBOM() const` [inline]

Returns true if this document has a leading Byte Order Mark of UTF8.

7.183.3.13 **char*** `tinyxml2::XMLDocument::Identify(char * p, XMLNode ** node)`

7.183.3.14 **XMLError** `tinyxml2::XMLDocument::LoadFile(const char * filename)`

Load an XML file from disk. Returns XML_SUCCESS (0) on success, or an errorID.

7.183.3.15 **XMLError** `tinyxml2::XMLDocument::LoadFile(FILE *)`

Load an XML file from disk. You are responsible for providing and closing the FILE*.

NOTE: The file should be opened as binary ("rb") not text in order for TinyXML-2 to correctly do newline normalization.

Returns XML_SUCCESS (0) on success, or an errorID.

7.183.3.16 **void** `tinyxml2::XMLDocument::MarkInUse(XMLNode *)`

7.183.3.17 **XMLComment*** `tinyxml2::XMLDocument::NewComment(const char * comment)`

Create a new Comment associated with this Document. The memory for the Comment is managed by the Document.

7.183.3.18 **XMLDeclaration*** `tinyxml2::XMLDocument::NewDeclaration(const char * text = 0)`

Create a new Declaration associated with this Document. The memory for the object is managed by the Document. If the 'text' param is null, the standard declaration is used.:

```
<?xml version="1.0" encoding="UTF-8"?>
```

7.183.3.19 **XMLElement*** `tinyxml2::XMLDocument::NewElement(const char * name)`

Create a new Element associated with this Document. The memory for the Element is managed by the Document.

7.183.3.20 `XMLText* tinyxml2::XMLDocument::NewText (const char * text)`

Create a new Text associated with this Document. The memory for the Text is managed by the Document.

7.183.3.21 `XMLUnknown* tinyxml2::XMLDocument::NewUnknown (const char * text)`

Create a new Unknown associated with this Document. The memory for the object is managed by the Document.

7.183.3.22 `XMLError tinyxml2::XMLDocument::Parse (const char * xml, size_t nBytes = (size_t) (-1))`

Parse an XML file from a character string. Returns XML_SUCCESS (0) on success, or an errorID.

You may optionally pass in the 'nBytes', which is the number of bytes which will be parsed. If not specified, TinyXML-2 will assume 'xml' points to a null terminated string.

7.183.3.23 `void tinyxml2::XMLDocument::Print (XMLPrinter * streamer = 0) const`

Print the Document. If the Printer is not provided, it will print to stdout. If you provide Printer, this can print to a file:

```
XMLPrinter printer( fp );
doc.Print( &printer );
```

Or you can use a printer to print to memory:

```
XMLPrinter printer;
doc.Print( &printer );
// printer.CStr() has a const char* to the XML
```

7.183.3.24 `void tinyxml2::XMLDocument::PrintError () const`

A (trivial) utility function that prints the [ErrorStr\(\)](#) to stdout.

7.183.3.25 `bool tinyxml2::XMLDocument::ProcessEntities () const [inline]`

7.183.3.26 `XMLElement* tinyxml2::XMLDocument::RootElement () [inline]`

Return the root element of DOM. Equivalent to [FirstChildElement\(\)](#). To get the first node, use [FirstChild\(\)](#).

7.183.3.27 `const XMLElement* tinyxml2::XMLDocument::RootElement () const [inline]`

7.183.3.28 `XMLError tinyxml2::XMLDocument::SaveFile (const char * filename, bool compact = false)`

Save the XML file to disk. Returns XML_SUCCESS (0) on success, or an errorID.

7.183.3.29 `XMLError tinyxml2::XMLDocument::SaveFile (FILE * fp, bool compact = false)`

Save the XML file to disk. You are responsible for providing and closing the FILE*.

Returns XML_SUCCESS (0) on success, or an errorID.

7.183.3.30 `void tinyxml2::XMLDocument::SetBOM (bool useBOM) [inline]`

Sets whether to write the BOM when writing the file.

7.183.3.31 virtual **XMLNode*** tinyxml2::XMLDocument::ShallowClone (**XMLDocument * document**) const
 [inline], [virtual]

Make a copy of this node, but not its children. You may pass in a Document pointer that will be the owner of the new Node. If the 'document' is null, then the node returned will be allocated from the current Document. (this->[GetDocument\(\)](#))

Note: if called on a [XMLDocument](#), this will return null.

Implements [tinyxml2::XMLNode](#).

7.183.3.32 virtual bool tinyxml2::XMLDocument::ShallowEqual (**const XMLNode * compare**) const [inline],
 [virtual]

Test if 2 nodes are the same, but don't test children. The 2 nodes do not need to be in the same Document.

Note: if called on a [XMLDocument](#), this will return false.

Implements [tinyxml2::XMLNode](#).

7.183.3.33 virtual **XMLDocument*** tinyxml2::XMLDocument::ToDocument () [inline], [virtual]

Safely cast to a Document, or null.

Reimplemented from [tinyxml2::XMLNode](#).

7.183.3.34 virtual const **XMLDocument*** tinyxml2::XMLDocument::ToDocument () const [inline], [virtual]

Reimplemented from [tinyxml2::XMLNode](#).

7.183.3.35 **Whitespace** tinyxml2::XMLDocument::WhitespaceMode () const [inline]

7.183.4 Friends And Related Function Documentation

7.183.4.1 friend class **XMLComment** [friend]

7.183.4.2 friend class **XMLDeclaration** [friend]

7.183.4.3 friend class **XMLElement** [friend]

7.183.4.4 friend class **XMLNode** [friend]

7.183.4.5 friend class **XMLText** [friend]

7.183.4.6 friend class **XMLUnknown** [friend]

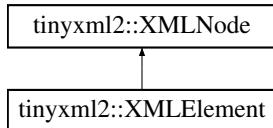
The documentation for this class was generated from the following file:

- include/tinyxml2.h

7.184 tinyxml2::XMLElement Class Reference

```
#include <tinyxml2.h>
```

Inheritance diagram for tinyxml2::XMLElement:



Public Types

- enum [ElementClosingType](#) { [OPEN](#), [CLOSED](#), [CLOSING](#) }

Public Member Functions

- const char * [Name](#) () const
Get the name of an element (which is the [Value\(\)](#) of the node.)
- void [SetName](#) (const char *str, bool staticMem=false)
Set the name of the element.
- virtual [XMLElement](#) * [ToElement](#) ()
Safely cast to an Element, or null.
- virtual const [XMLElement](#) * [ToElement](#) () const
- virtual bool [Accept](#) ([XMLVisitor](#) *visitor) const
- const char * [Attribute](#) (const char *name, const char *value=0) const
- int [IntAttribute](#) (const char *name, int defaultValue=0) const
- [ProtocolPP](#)::jarray< uint8_t > [ArrayAttribute](#) (const char *name, [ProtocolPP](#)::jarray< uint8_t > defaultValue=[ProtocolPP](#)::jarray< uint8_t >(0)) const
- uint8_t [ByteAttribute](#) (const char *name) const
- uint16_t [ShortAttribute](#) (const char *name) const
- unsigned [UnsignedAttribute](#) (const char *name, unsigned defaultValue=0) const
See [IntAttribute\(\)](#)
- int64_t [Int64Attribute](#) (const char *name, int64_t defaultValue=0) const
See [IntAttribute\(\)](#)
- bool [BoolAttribute](#) (const char *name, bool defaultValue=false) const
See [IntAttribute\(\)](#)
- uint64_t [UnsignedU64Attribute](#) (const char *name) const
- double [DoubleAttribute](#) (const char *name, double defaultValue=0) const
See [IntAttribute\(\)](#)
- float [FloatAttribute](#) (const char *name, float defaultValue=0) const
See [IntAttribute\(\)](#)
- [XMLElement](#) [QueryIntAttribute](#) (const char *name, int *value) const
- [XMLElement](#) [QueryArrayAttribute](#) (const char *name, [ProtocolPP](#)::jarray< uint8_t > *value) const
- [XMLElement](#) [QueryByteAttribute](#) (const char *name, uint8_t *value) const
- [XMLElement](#) [QueryShortAttribute](#) (const char *name, uint16_t *value) const
- [XMLElement](#) [QueryUnsignedAttribute](#) (const char *name, unsigned int *value) const
See [QueryIntAttribute\(\)](#)
- [XMLElement](#) [QueryInt64Attribute](#) (const char *name, int64_t *value) const
See [QueryIntAttribute\(\)](#)
- [XMLElement](#) [QueryUnsignedU64Attribute](#) (const char *name, uint64_t *value) const
- [XMLElement](#) [QueryBoolAttribute](#) (const char *name, bool *value) const
See [QueryIntAttribute\(\)](#)
- [XMLElement](#) [QueryDoubleAttribute](#) (const char *name, double *value) const
See [QueryIntAttribute\(\)](#)
- [XMLElement](#) [QueryFloatAttribute](#) (const char *name, float *value) const

- See [QueryIntAttribute\(\)](#)
- **XMLError QueryStringAttribute** (const char *name, const char **value) const
 - See [QueryIntAttribute\(\)](#)
- int **QueryAttribute** (const char *name, int *value) const
- int **QueryAttribute** (const char *name, [ProtocolPP::jarray< uint8_t >](#) *value) const
- int **QueryAttribute** (const char *name, uint8_t *value) const
- int **QueryAttribute** (const char *name, uint16_t *value) const
- int **QueryAttribute** (const char *name, unsigned int *value) const
- int **QueryAttribute** (const char *name, int64_t *value) const
- int **QueryAttribute** (const char *name, uint64_t *value) const
- int **QueryAttribute** (const char *name, bool *value) const
- int **QueryAttribute** (const char *name, double *value) const
- int **QueryAttribute** (const char *name, float *value) const
- void **SetAttribute** (const char *name, const char *value)
 - Sets the named attribute to value.*
- void **SetAttribute** (const char *name, int value)
 - Sets the named attribute to value.*
- void **SetAttribute** (const char *name, [ProtocolPP::jarray< uint8_t >](#) value)
 - Sets the named attribute to value (Added for Protocol++).*
- void **SetAttribute** (const char *name, uint8_t value)
 - Sets the named attribute to value (Added for Protocol++).*
- void **SetAttribute** (const char *name, uint16_t value)
 - Sets the named attribute to value (Added for Protocol++).*
- void **SetAttribute** (const char *name, int64_t value)
 - Sets the named attribute to value.*
- void **SetAttribute** (const char *name, uint64_t value)
 - Sets the named attribute to value.*
- void **SetAttribute** (const char *name, bool value)
 - Sets the named attribute to value.*
- void **SetAttribute** (const char *name, double value)
 - Sets the named attribute to value.*
- void **SetAttribute** (const char *name, float value)
 - Sets the named attribute to value.*
- void **DeleteAttribute** (const char *name)
- const **XMLAttribute * FirstAttribute** () const
 - Return the first attribute in the list.*
- const **XMLAttribute * FindAttribute** (const char *name) const
 - Query a specific attribute in the list.*
- const char * **GetText** () const
- void **SetText** (const char *inText)
- void **SetText** (int value)
 - Convenience method for setting text inside an element. See [SetText\(\)](#) for important limitations.*
- void **SetText** ([ProtocolPP::jarray< uint8_t >](#) value)
 - Convenience method for setting text inside an element. See [SetText\(\)](#) for important limitations.*
- void **SetText** (uint8_t value)
 - Convenience method for setting text inside an element. See [SetText\(\)](#) for important limitations.*
- void **SetText** (uint16_t value)
 - Convenience method for setting text inside an element. See [SetText\(\)](#) for important limitations.*
- void **SetText** (unsigned value)
 - Convenience method for setting text inside an element. See [SetText\(\)](#) for important limitations.*
- void **SetText** (uint64_t value)
 - Convenience method for setting text inside an element. See [SetText\(\)](#) for important limitations.*
- void **SetText** (const char *inText)
 - Convenience method for setting text inside an element. See [SetText\(\)](#) for important limitations.*

- void [SetText](#) (int64_t value)

Convenience method for setting text inside an element. See [SetText\(\)](#) for important limitations.
- void [SetText](#) (bool value)

Convenience method for setting text inside an element. See [SetText\(\)](#) for important limitations.
- void [SetText](#) (double value)

Convenience method for setting text inside an element. See [SetText\(\)](#) for important limitations.
- void [SetText](#) (float value)

Convenience method for setting text inside an element. See [SetText\(\)](#) for important limitations.
- [XMLError QueryIntText](#) (int *ival) const

See [QueryIntText\(\)](#). Added for ProtocolPP()
- [XMLError QueryArrayText](#) (ProtocolPP::jarray< uint8_t > *uval) const

See [QueryIntText\(\)](#). Added for ProtocolPP()
- [XMLError QueryByteText](#) (uint8_t *uval) const

See [QueryIntText\(\)](#). Added for ProtocolPP()
- [XMLError QueryShortText](#) (uint16_t *uval) const

See [QueryIntText\(\)](#). Added for ProtocolPP()
- [XMLError QueryUnsignedText](#) (unsigned *uval) const

See [QueryIntText\(\)](#)
- [XMLError QueryUnsignedU64Text](#) (uint64_t *uval) const

See [QueryIntText\(\)](#). Added for ProtocolPP()
- [XMLError QueryInt64Text](#) (int64_t *uval) const

See [QueryIntText\(\)](#)
- [XMLError QueryBoolText](#) (bool *bval) const

See [QueryIntText\(\)](#)
- [XMLError QueryDoubleText](#) (double *dval) const

See [QueryIntText\(\)](#)
- [XMLError QueryFloatText](#) (float *fval) const

See [QueryIntText\(\)](#)
- int [IntText](#) (int defaultValue=0) const
- unsigned [UnsignedText](#) (unsigned defaultValue=0) const

See [QueryIntText\(\)](#)
- int64_t [Int64Text](#) (int64_t defaultValue=0) const

See [QueryIntText\(\)](#)
- bool [BoolText](#) (bool defaultValue=false) const

See [QueryIntText\(\)](#)
- double [DoubleText](#) (double defaultValue=0) const

See [QueryIntText\(\)](#)
- float [FloatText](#) (float defaultValue=0) const

See [QueryIntText\(\)](#)
- [ElementClosingType ClosingType](#) () const
- virtual [XMLNode](#) * [ShallowClone](#) ([XMLDocument](#) *document) const
- virtual bool [ShallowEqual](#) (const [XMLNode](#) *compare) const

Protected Member Functions

- char * [ParseDeep](#) (char *p, [StrPair](#) *parentEndTag, int *curLineNumPtr)

Friends

- class [XMLDocument](#)

Additional Inherited Members

7.184.1 Detailed Description

The element is a container class. It has a value, the element name, and can contain other elements, text, comments, and unknowns. Elements also contain an arbitrary number of attributes.

7.184.2 Member Enumeration Documentation

7.184.2.1 enum tinyxml2::XMLElement::ElementClosingType

Enumerator

OPEN

CLOSED

CLOSING

7.184.3 Member Function Documentation

7.184.3.1 virtual bool tinyxml2::XMLElement::Accept (XMLVisitor * visitor) const [virtual]

Accept a hierarchical visit of the nodes in the TinyXML-2 DOM. Every node in the XML tree will be conditionally visited and the host will be called back via the [XMLVisitor](#) interface.

This is essentially a SAX interface for TinyXML-2. (Note however it doesn't re-parse the XML for the callbacks, so the performance of TinyXML-2 is unchanged by using this interface versus any other.)

The interface has been based on ideas from:

- <http://www.saxproject.org/>
- <http://c2.com/cgi/wiki?HierarchicalVisitorPattern>

Which are both good references for "visiting".

An example of using [Accept\(\)](#):

```
XMLPrinter printer;
tinyxmlDoc.Accept( &printer );
const char* xmlcstr = printer.CStr();
```

Implements [tinyxml2::XMLNode](#).

7.184.3.2 ProtocolPP::jarray<uint8_t> tinyxml2::XMLElement::ArrayAttribute (const char * name, ProtocolPP::jarray< uint8_t > defaultValue = ProtocolPP::jarray< uint8_t >(0)) const

Given an attribute name, [ArrayAttribute\(\)](#) returns the value of the attribute interpreted as a byte array. 0 will be returned if there is an error. For a method with error checking, see [QueryArrayAttribute\(\)](#). Added for ProtocolPP()

7.184.3.3 const char* tinyxml2::XMLElement::Attribute (const char * name, const char * value = 0) const

Given an attribute name, [Attribute\(\)](#) returns the value for the attribute of that name, or null if none exists. For example:

```
const char* value = ele->Attribute( "foo" );
```

The 'value' parameter is normally null. However, if specified, the attribute will only be returned if the 'name' and 'value' match. This allow you to write code:

```
if ( ele->Attribute( "foo", "bar" ) ) callFooIsBar();
```

rather than:

```
if ( ele->Attribute( "foo" ) ) {
    if ( strcmp( ele->Attribute( "foo" ), "bar" ) == 0 ) callFooIsBar();
}
```

7.184.3.4 `bool tinyxml2::XMLElement::BoolAttribute(const char * name, bool defaultValue = false) const`

See [IntAttribute\(\)](#)

7.184.3.5 `bool tinyxml2::XMLElement::BoolText(bool defaultValue = false) const`

See [QueryIntText\(\)](#)

7.184.3.6 `uint8_t tinyxml2::XMLElement::ByteAttribute(const char * name) const`

Given an attribute name, [ByteAttribute\(\)](#) returns the value of the attribute interpreted as a byte. 0 will be returned if there is an error. For a method with error checking, see [QueryByteAttribute\(\)](#). Added for ProtocolPP()

7.184.3.7 `ElementClosingType tinyxml2::XMLElement::ClosingType() const [inline]`

7.184.3.8 `void tinyxml2::XMLElement::DeleteAttribute(const char * name)`

Delete an attribute.

7.184.3.9 `double tinyxml2::XMLElement::DoubleAttribute(const char * name, double defaultValue = 0) const`

See [IntAttribute\(\)](#)

7.184.3.10 `double tinyxml2::XMLElement::DoubleText(double defaultValue = 0) const`

See [QueryIntText\(\)](#)

7.184.3.11 `const XMLAttribute* tinyxml2::XMLElement::FindAttribute(const char * name) const`

Query a specific attribute in the list.

7.184.3.12 `const XMLAttribute* tinyxml2::XMLElement::FirstAttribute() const [inline]`

Return the first attribute in the list.

7.184.3.13 `float tinyxml2::XMLElement::FloatAttribute(const char * name, float defaultValue = 0) const`

See [IntAttribute\(\)](#)

7.184.3.14 float tinyxml2::XMLElement::FloatText (float *defaultValue* = 0) const

See [QueryIntText\(\)](#)

7.184.3.15 const char* tinyxml2::XMLElement::GetText () const

Convenience function for easy access to the text inside an element. Although easy and concise, [GetText\(\)](#) is limited compared to getting the [XMLText](#) child and accessing it directly.

If the first child of 'this' is a [XMLText](#), the [GetText\(\)](#) returns the character string of the Text node, else null is returned. This is a convenient method for getting the text of simple contained text:

```
<foo>This is text</foo>
const char* str = fooElement->GetText();
```

'str' will be a pointer to "This is text".

Note that this function can be misleading. If the element foo was created from this XML:

```
<foo><b>This is text</b></foo>
```

then the value of str would be null. The first child node isn't a text node, it is another element. From this XML:

```
<foo>This is <b>text</b></foo>
```

[GetText\(\)](#) will return "This is ".

7.184.3.16 int64_t tinyxml2::XMLElement::Int64Attribute (const char * *name*, int64_t *defaultValue* = 0) const

See [IntAttribute\(\)](#)

7.184.3.17 int64_t tinyxml2::XMLElement::Int64Text (int64_t *defaultValue* = 0) const

See [QueryIntText\(\)](#)

7.184.3.18 int tinyxml2::XMLElement::IntAttribute (const char * *name*, int *defaultValue* = 0) const

Given an attribute name, [IntAttribute\(\)](#) returns the value of the attribute interpreted as an integer. The default value will be returned if the attribute isn't present, or if there is an error. (For a method with error checking, see [QueryIntAttribute\(\)](#)).

7.184.3.19 int tinyxml2::XMLElement::IntText (int *defaultValue* = 0) const

7.184.3.20 const char* tinyxml2::XMLElement::Name () const [inline]

Get the name of an element (which is the [Value\(\)](#) of the node.)

7.184.3.21 char* tinyxml2::XMLElement::ParseDeep (char * *p*, StrPair * *parentEndTag*, int * *curLineNumPtr*) [protected], [virtual]

Reimplemented from [tinyxml2::XMLNode](#).

7.184.3.22 XMLError tinyxml2::XMLElement::QueryArrayAttribute (`const char * name, ProtocolPP::jarray< uint8_t > * value`) const [inline]

Given an attribute name, `QueryArrayAttribute()` returns `XML_NO_ERROR`, `XML_WRONG_ATTRIBUTE_TYPE` if the conversion can't be performed, or `XML_NO_ATTRIBUTE` if the attribute doesn't exist. If successful, the result of the conversion will be written to '`value`'. If not successful, nothing will be written to '`value`'. This allows you to provide default value:

```
/// jarray<uint8_t> value(10, 0xAA);
/// QueryArrayAttribute( "key", &value ); // if "key" isn't found, value will still be 10 bytes of "AA"
///
```

Added for `ProtocolPP()`

7.184.3.23 XMLError tinyxml2::XMLElement::QueryArrayText (`ProtocolPP::jarray< uint8_t > * uval`) const

See [QueryIntText\(\)](#). Added for `ProtocolPP()`

7.184.3.24 int tinyxml2::XMLElement::QueryAttribute (`const char * name, int * value`) const [inline]

Given an attribute name, [QueryAttribute\(\)](#) returns `XML_SUCCESS`, `XML_WRONG_ATTRIBUTE_TYPE` if the conversion can't be performed, or `XML_NO_ATTRIBUTE` if the attribute doesn't exist. It is overloaded for the primitive types, and is a generally more convenient replacement of [QueryIntAttribute\(\)](#) and related functions.

If successful, the result of the conversion will be written to '`value`'. If not successful, nothing will be written to '`value`'. This allows you to provide default value:

```
int value = 10;
QueryAttribute( "foo", &value ); // if "foo" isn't found, value will still be 10
```

7.184.3.25 int tinyxml2::XMLElement::QueryAttribute (`const char * name, ProtocolPP::jarray< uint8_t > * value`) const [inline]

Given an attribute name, `QueryAttribute()` returns `XML_NO_ERROR`, `XML_WRONG_ATTRIBUTE_TYPE` if the conversion can't be performed, or `XML_NO_ATTRIBUTE` if the attribute doesn't exist. If successful, the result of the conversion will be written to '`value`'. If not successful, nothing will be written to '`value`'. This allows you to provide default value:

```
/// jarray<uint8_t> value(4, 0xCC);
/// QueryAttribute( "seqnum", &value ); // if "seqnum" isn't found, value will still be "00000000"
///
```

Added for `ProtocolPP()`

7.184.3.26 int tinyxml2::XMLElement::QueryAttribute (`const char * name, uint8_t * value`) const [inline]

Given an attribute name, `QueryAttribute()` returns `XML_NO_ERROR`, `XML_WRONG_ATTRIBUTE_TYPE` if the conversion can't be performed, or `XML_NO_ATTRIBUTE` if the attribute doesn't exist. If successful, the result of the conversion will be written to '`value`'. If not successful, nothing will be written to '`value`'. This allows you to provide default value:

```
/// uint8_t value = 0xFF;
/// QueryAttribute( "nh", &value ); // if "nh" isn't found, value will still be "FF"
///
```

Added for ProtocolPP()

7.184.3.27 int tinyxml2::XMLElement::QueryAttribute (const char * *name*, uint16_t * *value*) const [inline]

Given an attribute name, QueryAttribute() returns XML_NO_ERROR, XML_WRONG_ATTRIBUTE_TYPE if the conversion can't be performed, or XML_NO_ATTRIBUTE if the attribute doesn't exist. If successful, the result of the conversion will be written to 'value'. If not successful, nothing will be written to 'value'. This allows you to provide default value:

```
/// uint16_t value = 0xFFFF;
/// QueryAttribute( "seqnum", &value ); // if "seqnum" isn't found, value will still be "FFFF"
///
```

Added for ProtocolPP()

7.184.3.28 int tinyxml2::XMLElement::QueryAttribute (const char * *name*, unsigned int * *value*) const [inline]

7.184.3.29 int tinyxml2::XMLElement::QueryAttribute (const char * *name*, int64_t * *value*) const [inline]

7.184.3.30 int tinyxml2::XMLElement::QueryAttribute (const char * *name*, uint64_t * *value*) const [inline]

7.184.3.31 int tinyxml2::XMLElement::QueryAttribute (const char * *name*, bool * *value*) const [inline]

7.184.3.32 int tinyxml2::XMLElement::QueryAttribute (const char * *name*, double * *value*) const [inline]

7.184.3.33 int tinyxml2::XMLElement::QueryAttribute (const char * *name*, float * *value*) const [inline]

7.184.3.34 XMLError tinyxml2::XMLElement::QueryBoolAttribute (const char * *name*, bool * *value*) const [inline]

See [QueryIntAttribute\(\)](#)

7.184.3.35 XMLError tinyxml2::XMLElement::QueryBoolText (bool * *bval*) const

See [QueryIntText\(\)](#)

7.184.3.36 XMLError tinyxml2::XMLElement::QueryByteAttribute (const char * *name*, uint8_t * *value*) const [inline]

Given an attribute name, QueryByteAttribute() returns XML_NO_ERROR, XML_WRONG_ATTRIBUTE_TYPE if the conversion can't be performed, or XML_NO_ATTRIBUTE if the attribute doesn't exist. If successful, the result of the conversion will be written to 'value'. If not successful, nothing will be written to 'value'. This allows you to provide default value:

```
/// uint8_t value = 0xFF;
/// QueryByteAttribute( "nh", &value ); // if "nh" isn't found, value will still be "FF"
///
```

Added for ProtocolPP()

7.184.3.37 **XMLError** `tinyxml2::XMLElement::QueryByteText (uint8_t * uval) const`

See [QueryIntText\(\)](#). Added for ProtocolPP()

7.184.3.38 **XMLError** `tinyxml2::XMLElement::QueryDoubleAttribute (const char * name, double * value) const [inline]`

See [QueryIntAttribute\(\)](#)

7.184.3.39 **XMLError** `tinyxml2::XMLElement::QueryDoubleText (double * dval) const`

See [QueryIntText\(\)](#)

7.184.3.40 **XMLError** `tinyxml2::XMLElement::QueryFloatAttribute (const char * name, float * value) const [inline]`

See [QueryIntAttribute\(\)](#)

7.184.3.41 **XMLError** `tinyxml2::XMLElement::QueryFloatText (float * fval) const`

See [QueryIntText\(\)](#)

7.184.3.42 **XMLError** `tinyxml2::XMLElement::QueryInt64Attribute (const char * name, int64_t * value) const [inline]`

See [QueryIntAttribute\(\)](#)

7.184.3.43 **XMLError** `tinyxml2::XMLElement::QueryInt64Text (int64_t * uval) const`

See [QueryIntText\(\)](#)

7.184.3.44 **XMLError** `tinyxml2::XMLElement::QueryIntAttribute (const char * name, int * value) const [inline]`

Given an attribute name, [QueryIntAttribute\(\)](#) returns XML_SUCCESS, XML_WRONG_ATTRIBUTE_TYPE if the conversion can't be performed, or XML_NO_ATTRIBUTE if the attribute doesn't exist. If successful, the result of the conversion will be written to 'value'. If not successful, nothing will be written to 'value'. This allows you to provide default value:

```
int value = 10;
QueryIntAttribute( "foo", &value );      // if "foo" isn't found, value will still be 10
```

7.184.3.45 **XMLError** `tinyxml2::XMLElement::QueryIntText (int * ival) const`

Convenience method to query the value of a child text node. This is probably best shown by example. Given you have a document is this form:

```
<point>
  <x>1</x>
  <y>1.4</y>
</point>
```

The [QueryIntText\(\)](#) and similar functions provide a safe and easier way to get to the "value" of x and y.

```

int x = 0;
float y = 0;      // types of x and y are contrived for example
const XMLElement* xElement = pointElement->FirstChildElement( "x" );
const XMLElement* yElement = pointElement->FirstChildElement( "y" );
xElement->QueryIntText( &x );
yElement->QueryFloatText( &y );

```

Returns

XML_SUCCESS (0) on success, XML_CAN_NOT_CONVERT_TEXT if the text cannot be converted to the requested type, and XML_NO_TEXT_NODE if there is no child text to query.

7.184.3.46 XMLError tinyxml2::XMLElement::QueryShortAttribute (const char * *name*, uint16_t * *value*) const [inline]

Given an attribute name, QueryShortAttribute() returns XML_NO_ERROR, XML_WRONG_ATTRIBUTE_TYPE if the conversion can't be performed, or XML_NO_ATTRIBUTE if the attribute doesn't exist. If successful, the result of the conversion will be written to '*value*'. If not successful, nothing will be written to '*value*'. This allows you to provide default value:

```

/// uint16_t value = 0xFFFF;
/// QueryShortAttribute( "version", &value ); // if "version" isn't found, value will still be "FFFE"
///

```

Added for ProtocolPP()

7.184.3.47 XMLError tinyxml2::XMLElement::QueryShortText (uint16_t * *uval*) const

See [QueryIntText\(\)](#). Added for ProtocolPP()

7.184.3.48 XMLError tinyxml2::XMLElement::QueryStringAttribute (const char * *name*, const char ** *value*) const [inline]

See [QueryIntAttribute\(\)](#)

7.184.3.49 XMLError tinyxml2::XMLElement::QueryUnsignedAttribute (const char * *name*, unsigned int * *value*) const [inline]

See [QueryIntAttribute\(\)](#)

7.184.3.50 XMLError tinyxml2::XMLElement::QueryUnsignedText (unsigned * *uval*) const

See [QueryIntText\(\)](#)

7.184.3.51 XMLError tinyxml2::XMLElement::QueryUnsignedU64Attribute (const char * *name*, uint64_t * *value*) const [inline]

Given an attribute name, QueryUnsignedU64Attribute() returns XML_NO_ERROR, XML_WRONG_ATTRIBUTE_TYPE if the conversion can't be performed, or XML_NO_ATTRIBUTE if the attribute doesn't exist. If successful, the result of the conversion will be written to '*value*'. If not successful, nothing will be written to '*value*'. This allows you to provide default value:

```
/// uint64_t value = 0xFFFFFFFFFFFFFFFF;
/// QueryUnsignedU64Attribute( "seqnum", &value ); // if "seqnum" isn't found, value will still be "FFFFFFFF"
```

Added for ProtocolPP()

7.184.3.52 `XMLError tinyxml2::XMLElement::QueryUnsignedU64Text(uint64_t * uval) const`

See [QueryIntText\(\)](#). Added for ProtocolPP()

7.184.3.53 `void tinyxml2::XMLElement::SetAttribute(const char * name, const char * value) [inline]`

Sets the named attribute to value.

7.184.3.54 `void tinyxml2::XMLElement::SetAttribute(const char * name, int value) [inline]`

Sets the named attribute to value.

7.184.3.55 `void tinyxml2::XMLElement::SetAttribute(const char * name, ProtocolPP::jarray< uint8_t > value) [inline]`

Sets the named attribute to value (Added for Protocol++).

7.184.3.56 `void tinyxml2::XMLElement::SetAttribute(const char * name, uint8_t value) [inline]`

Sets the named attribute to value (Added for Protocol++).

7.184.3.57 `void tinyxml2::XMLElement::SetAttribute(const char * name, uint16_t value) [inline]`

Sets the named attribute to value (Added for Protocol++).

7.184.3.58 `void tinyxml2::XMLElement::SetAttribute(const char * name, int64_t value) [inline]`

Sets the named attribute to value.

7.184.3.59 `void tinyxml2::XMLElement::SetAttribute(const char * name, uint64_t value) [inline]`

Sets the named attribute to value.

7.184.3.60 `void tinyxml2::XMLElement::SetAttribute(const char * name, bool value) [inline]`

Sets the named attribute to value.

7.184.3.61 `void tinyxml2::XMLElement::SetAttribute(const char * name, double value) [inline]`

Sets the named attribute to value.

7.184.3.62 `void tinyxml2::XMLElement::SetAttribute(const char * name, float value) [inline]`

Sets the named attribute to value.

7.184.3.63 void tinyxml2::XMLElement::SetName (const char * *str*, bool *staticMem* = false) [inline]

Set the name of the element.

7.184.3.64 void tinyxml2::XMLElement::SetText (const char * *inText*)

Convenience function for easy access to the text inside an element. Although easy and concise, [SetText\(\)](#) is limited compared to creating an [XMLText](#) child and mutating it directly.

If the first child of 'this' is a [XMLText](#), [SetText\(\)](#) sets its value to the given string, otherwise it will create a first child that is an [XMLText](#).

This is a convenient method for setting the text of simple contained text:

```
<foo>This is text</foo>
    fooElement->SetText( "Hullabaloo! " );
<foo>Hullabaloo!</foo>
```

Note that this function can be misleading. If the element foo was created from this XML:

```
<foo><b>This is text</b></foo>
```

then it will not change "This is text", but rather prefix it with a text element:

```
<foo>Hullabaloo!<b>This is text</b></foo>
```

For this XML:

```
<foo />
```

[SetText\(\)](#) will generate

```
<foo>Hullabaloo!</foo>
```

7.184.3.65 void tinyxml2::XMLElement::SetText (int *value*)

Convenience method for setting text inside an element. See [SetText\(\)](#) for important limitations.

7.184.3.66 void tinyxml2::XMLElement::SetText (ProtocolPP::jarray< uint8_t > *value*)

Convenience method for setting text inside an element. See [SetText\(\)](#) for important limitations.

7.184.3.67 void tinyxml2::XMLElement::SetText (uint8_t *value*)

Convenience method for setting text inside an element. See [SetText\(\)](#) for important limitations.

7.184.3.68 void tinyxml2::XMLElement::SetText (uint16_t *value*)

Convenience method for setting text inside an element. See [SetText\(\)](#) for important limitations.

7.184.3.69 void tinyxml2::XMLElement::SetText (unsigned *value*)

Convenience method for setting text inside an element. See [SetText\(\)](#) for important limitations.

7.184.3.70 void tinyxml2::XMLElement::SetText (*uint64_t value*)

Convenience method for setting text inside an element. See [SetText\(\)](#) for important limitations.

7.184.3.71 void tinyxml2::XMLElement::SetText (*int64_t value*)

Convenience method for setting text inside an element. See [SetText\(\)](#) for important limitations.

7.184.3.72 void tinyxml2::XMLElement::SetText (*bool value*)

Convenience method for setting text inside an element. See [SetText\(\)](#) for important limitations.

7.184.3.73 void tinyxml2::XMLElement::SetText (*double value*)

Convenience method for setting text inside an element. See [SetText\(\)](#) for important limitations.

7.184.3.74 void tinyxml2::XMLElement::SetText (*float value*)

Convenience method for setting text inside an element. See [SetText\(\)](#) for important limitations.

7.184.3.75 virtual **XMLElement*** tinyxml2::XMLElement::ShallowClone (*XMLDocument * document*) const [virtual]

Make a copy of this node, but not its children. You may pass in a Document pointer that will be the owner of the new Node. If the 'document' is null, then the node returned will be allocated from the current Document. (*this->GetDocument()*)

Note: if called on a [XMLDocument](#), this will return null.

Implements [tinyxml2::XMLNode](#).

7.184.3.76 virtual bool tinyxml2::XMLElement::ShallowEqual (*const XMLNode * compare*) const [virtual]

Test if 2 nodes are the same, but don't test children. The 2 nodes do not need to be in the same Document.

Note: if called on a [XMLDocument](#), this will return false.

Implements [tinyxml2::XMLNode](#).

7.184.3.77 uint16_t tinyxml2::XMLElement::ShortAttribute (*const char * name*) const

Given an attribute name, [ShortAttribute\(\)](#) returns the value of the attribute interpreted as a *uint16_t*. 0 will be returned if there is an error. For a method with error checking, see [QueryShortAttribute\(\)](#). Added for ProtocolPP()

7.184.3.78 virtual **XMLElement*** tinyxml2::XMLElement::ToElement () [inline], [virtual]

Safely cast to an Element, or null.

Reimplemented from [tinyxml2::XMLNode](#).

7.184.3.79 virtual const **XMLElement*** tinyxml2::XMLElement::ToElement () const [inline], [virtual]

Reimplemented from [tinyxml2::XMLNode](#).

7.184.3.80 `unsigned tinyxml2::XMLElement::UnsignedAttribute (const char * name, unsigned defaultValue = 0) const`

See [IntAttribute\(\)](#)

7.184.3.81 `unsigned tinyxml2::XMLElement::UnsignedText (unsigned defaultValue = 0) const`

See [QueryIntText\(\)](#)

7.184.3.82 `uint64_t tinyxml2::XMLElement::UnsignedU64Attribute (const char * name) const`

Given an attribute name, [UnsignedU64Attribute\(\)](#) returns the value of the attribute interpreted as a `uint64_t`. 0 will be returned if there is an error. For a method with error checking, see [QueryUnsignedLongAttribute\(\)](#). Added for ProtocolPP()

7.184.4 Friends And Related Function Documentation

7.184.4.1 `friend class XMLDocument [friend]`

The documentation for this class was generated from the following file:

- `include/tinyxml2.h`

7.185 tinyxml2::XMLHandle Class Reference

```
#include <tinyxml2.h>
```

Public Member Functions

- [`XMLHandle \(XMLNode *node\)`](#)
Create a handle from any node (at any depth of the tree.) This can be a null pointer.
- [`XMLHandle \(XMLNode &node\)`](#)
Create a handle from a node.
- [`XMLHandle \(const XMLHandle &ref\)`](#)
Copy constructor.
- [`XMLHandle & operator= \(const XMLHandle &ref\)`](#)
Assignment.
- [`XMLHandle FirstChild \(\)`](#)
Get the first child of this handle.
- [`XMLHandle FirstChildElement \(const char *name=0\)`](#)
Get the first child element of this handle.
- [`XMLHandle LastChild \(\)`](#)
Get the last child of this handle.
- [`XMLHandle LastChildElement \(const char *name=0\)`](#)
Get the last child element of this handle.
- [`XMLHandle PreviousSibling \(\)`](#)
Get the previous sibling of this handle.
- [`XMLHandle PreviousSiblingElement \(const char *name=0\)`](#)
Get the previous sibling element of this handle.
- [`XMLHandle NextSibling \(\)`](#)
Get the next sibling of this handle.

- `XMLHandle NextSiblingElement (const char *name=0)`
Get the next sibling element of this handle.
- `XMLNode * ToNode ()`
Safe cast to `XMLNode`. This can return null.
- `XMLElement * ToElement ()`
Safe cast to `XMLElement`. This can return null.
- `XMLText * ToText ()`
Safe cast to `XMLText`. This can return null.
- `XMLUnknown * ToUnknown ()`
Safe cast to `XMLUnknown`. This can return null.
- `XMLDeclaration * ToDeclaration ()`
Safe cast to `XMLDeclaration`. This can return null.

7.185.1 Detailed Description

A `XMLHandle` is a class that wraps a node pointer with null checks; this is an incredibly useful thing. Note that `XMLHandle` is not part of the TinyXML-2 DOM structure. It is a separate utility class.

Take an example:

```
<Document>
  <Element attributeA = "valueA">
    <Child attributeB = "value1" />
    <Child attributeB = "value2" />
  </Element>
</Document>
```

Assuming you want the value of "attributeB" in the 2nd "Child" element, it's very easy to write a *lot* of code that looks like:

```
XMLElement* root = document.FirstChildElement( "Document" );
if ( root )
{
  XMLElement* element = root->FirstChildElement( "Element" );
  if ( element )
  {
    XMLElement* child = element->FirstChildElement( "Child" );
    if ( child )
    {
      XMLElement* child2 = child->NextSiblingElement( "Child" );
      if ( child2 )
      {
        // Finally do something useful.
```

And that doesn't even cover "else" cases. `XMLHandle` addresses the verbosity of such code. A `XMLHandle` checks for null pointers so it is perfectly safe and correct to use:

```
XMLHandle docHandle( &document );
XMLElement* child2 = docHandle.FirstChildElement( "Document" ).FirstChildElement( "Element" ).FirstChildElement();
if ( child2 )
{
  // do something useful
```

Which is MUCH more concise and useful.

It is also safe to copy handles - internally they are nothing more than node pointers.

```
XMLHandle handleCopy = handle;
```

See also `XMLConstHandle`, which is the same as `XMLHandle`, but operates on const objects.

7.185.2 Constructor & Destructor Documentation

7.185.2.1 `tinyxml2::XMLHandle::XMLHandle (XMLNode * node) [inline], [explicit]`

Create a handle from any node (at any depth of the tree.) This can be a null pointer.

7.185.2.2 `tinyxml2::XMLHandle::XMLHandle (XMLNode & node) [inline], [explicit]`

Create a handle from a node.

7.185.2.3 `tinyxml2::XMLHandle::XMLHandle (const XMLHandle & ref) [inline]`

Copy constructor.

7.185.3 Member Function Documentation

7.185.3.1 `XMLHandle tinyxml2::XMLHandle::FirstChild () [inline]`

Get the first child of this handle.

7.185.3.2 `XMLHandle tinyxml2::XMLHandle::FirstChildElement (const char * name = 0) [inline]`

Get the first child element of this handle.

7.185.3.3 `XMLHandle tinyxml2::XMLHandle::LastChild () [inline]`

Get the last child of this handle.

7.185.3.4 `XMLHandle tinyxml2::XMLHandle::LastChildElement (const char * name = 0) [inline]`

Get the last child element of this handle.

7.185.3.5 `XMLHandle tinyxml2::XMLHandle::NextSibling () [inline]`

Get the next sibling of this handle.

7.185.3.6 `XMLHandle tinyxml2::XMLHandle::NextSiblingElement (const char * name = 0) [inline]`

Get the next sibling element of this handle.

7.185.3.7 `XMLHandle& tinyxml2::XMLHandle::operator= (const XMLHandle & ref) [inline]`

Assignment.

7.185.3.8 `XMLHandle tinyxml2::XMLHandle::PreviousSibling () [inline]`

Get the previous sibling of this handle.

7.185.3.9 `XMLHandle tinyxml2::XMLHandle::PreviousSiblingElement (const char * name = 0) [inline]`

Get the previous sibling element of this handle.

7.185.3.10 `XMLDeclaration* tinyxml2::XMLHandle::ToDeclaration () [inline]`

Safe cast to [XMLDeclaration](#). This can return null.

7.185.3.11 `XMLElement* tinyxml2::XMLHandle::ToElement () [inline]`

Safe cast to [XMLElement](#). This can return null.

7.185.3.12 `XMLNode* tinyxml2::XMLHandle::ToNode () [inline]`

Safe cast to [XMLNode](#). This can return null.

7.185.3.13 `XMLText* tinyxml2::XMLHandle::ToText () [inline]`

Safe cast to [XMLText](#). This can return null.

7.185.3.14 `XMLUnknown* tinyxml2::XMLHandle::ToUnknown () [inline]`

Safe cast to [XMLUnknown](#). This can return null.

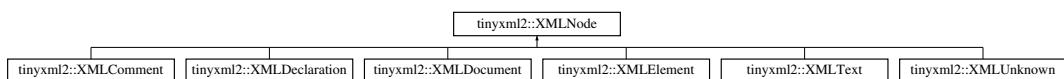
The documentation for this class was generated from the following file:

- [include/tinyxml2.h](#)

7.186 tinyxml2::XMLNode Class Reference

```
#include <tinyxml2.h>
```

Inheritance diagram for tinyxml2::XMLNode:



Public Member Functions

- `const XMLDocument * GetDocument () const`
Get the [XMLDocument](#) that owns this [XMLNode](#).
- `XMLDocument * GetDocument ()`
Get the [XMLDocument](#) that owns this [XMLNode](#).
- `virtual XMLElement * ToElement ()`
Safely cast to an Element, or null.
- `virtual XMLText * ToText ()`
Safely cast to Text, or null.
- `virtual XMLComment * ToComment ()`
Safely cast to a Comment, or null.

- virtual `XMLDocument * ToDocument ()`
Safely cast to a Document, or null.
- virtual `XMLDeclaration * ToDeclaration ()`
Safely cast to a Declaration, or null.
- virtual `XMLUnknown * ToUnknown ()`
Safely cast to an Unknown, or null.
- virtual const `XMLElement * ToElement () const`
- virtual const `XMLText * ToText () const`
- virtual const `XMLComment * ToComment () const`
- virtual const `XMLDocument * ToDocument () const`
- virtual const `XMLDeclaration * ToDeclaration () const`
- virtual const `XMLUnknown * ToUnknown () const`
- const char * `Value () const`
- void `SetValue (const char *val, bool staticMem=false)`
- int `GetLineNum () const`
Gets the line number the node is in, if the document was parsed from a file.
- const `XMLNode * Parent () const`
Get the parent of this node on the DOM.
- `XMLNode * Parent ()`
- bool `NoChildren () const`
Returns true if this node has no children.
- const `XMLNode * FirstChild () const`
Get the first child node, or null if none exists.
- `XMLNode * FirstChild ()`
- const `XMLElement * FirstChildElement (const char *name=0) const`
- `XMLElement * FirstChildElement (const char *name=0)`
- const `XMLNode * LastChild () const`
Get the last child node, or null if none exists.
- `XMLNode * LastChild ()`
- const `XMLElement * LastChildElement (const char *name=0) const`
- `XMLElement * LastChildElement (const char *name=0)`
- const `XMLNode * PreviousSibling () const`
Get the previous (left) sibling node of this node.
- `XMLNode * PreviousSibling ()`
- const `XMLElement * PreviousSiblingElement (const char *name=0) const`
Get the previous (left) sibling element of this node, with an optionally supplied name.
- `XMLElement * PreviousSiblingElement (const char *name=0)`
- const `XMLNode * NextSibling () const`
Get the next (right) sibling node of this node.
- `XMLNode * NextSibling ()`
- const `XMLElement * NextSiblingElement (const char *name=0) const`
Get the next (right) sibling element of this node, with an optionally supplied name.
- `XMLElement * NextSiblingElement (const char *name=0)`
- `XMLNode * InsertEndChild (XMLNode *addThis)`
- `XMLNode * LinkEndChild (XMLNode *addThis)`
- `XMLNode * InsertFirstChild (XMLNode *addThis)`
- `XMLNode * InsertAfterChild (XMLNode *afterThis, XMLNode *addThis)`
- void `DeleteChildren ()`
- void `DeleteChild (XMLNode *node)`
- virtual `XMLNode * ShallowClone (XMLDocument *document) const =0`
- `XMLNode * DeepClone (XMLDocument *target) const`
- virtual bool `ShallowEqual (const XMLNode *compare) const =0`
- virtual bool `Accept (XMLVisitor *visitor) const =0`
- void `SetUserData (void *userData)`
- void * `GetUserData () const`

Protected Member Functions

- `XMLNode (XMLDocument *)`
- `virtual ~XMLNode ()`
- `virtual char * ParseDeep (char *p, StrPair *parentEndTag, int *curLineNumPtr)`

Protected Attributes

- `XMLDocument * _document`
- `XMLNode * _parent`
- `StrPair _value`
- `int _parseLineNum`
- `XMLNode * _firstChild`
- `XMLNode * _lastChild`
- `XMLNode * _prev`
- `XMLNode * _next`
- `void * _userData`

Friends

- class `XMLDocument`
- class `XMLElement`

7.186.1 Detailed Description

`XMLNode` is a base class for every object that is in the XML Document Object Model (DOM), except `XMLAttributes`. Nodes have siblings, a parent, and children which can be navigated. A node is always in a `XMLDocument`. The type of a `XMLNode` can be queried, and it can be cast to its more defined type.

A `XMLDocument` allocates memory for all its Nodes. When the `XMLDocument` gets deleted, all its Nodes will also be deleted.

```
A Document can contain: Element (container or leaf)
                           Comment (leaf)
                           Unknown (leaf)
                           Declaration( leaf )
```

```
An Element can contain: Element (container or leaf)
                           Text      (leaf)
                           Attributes (not on tree)
                           Comment   (leaf)
                           Unknown   (leaf)
```

7.186.2 Constructor & Destructor Documentation

7.186.2.1 `tinyxml2::XMLNode::XMLNode (XMLDocument *) [explicit], [protected]`

7.186.2.2 `virtual tinyxml2::XMLNode::~XMLNode() [protected], [virtual]`

7.186.3 Member Function Documentation

7.186.3.1 `virtual bool tinyxml2::XMLNode::Accept (XMLVisitor * visitor) const [pure virtual]`

Accept a hierarchical visit of the nodes in the TinyXML-2 DOM. Every node in the XML tree will be conditionally visited and the host will be called back via the `XMLVisitor` interface.

This is essentially a SAX interface for TinyXML-2. (Note however it doesn't re-parse the XML for the callbacks, so the performance of TinyXML-2 is unchanged by using this interface versus any other.)

The interface has been based on ideas from:

- <http://www.saxproject.org/>
- <http://c2.com/cgi/wiki?HierarchicalVisitorPattern>

Which are both good references for "visiting".

An example of using [Accept\(\)](#):

```
XMLPrinter printer;
tinyxmlDoc.Accept( &printer );
const char* xmlcstr = printer.CStr();
```

Implemented in [tinyxml2::XMLDocument](#), [tinyxml2::XMLElement](#), [tinyxml2::XMLUnknown](#), [tinyxml2::XMLDeclaration](#), [tinyxml2::XMLComment](#), and [tinyxml2::XMLText](#).

7.186.3.2 XMLNode* tinyxml2::XMLNode::DeepClone ([XMLDocument](#) * *target*) const

Make a copy of this node and all its children.

If the 'target' is null, then the nodes will be allocated in the current document. If 'target' is specified, the memory will be allocated in the specified [XMLDocument](#).

NOTE: This is probably not the correct tool to copy a document, since XMLDocuments can have multiple top level XMLNodes. You probably want to use [XMLDocument::DeepCopy\(\)](#)

7.186.3.3 void tinyxml2::XMLNode::DeleteChild ([XMLNode](#) * *node*)

Delete a child of this node.

7.186.3.4 void tinyxml2::XMLNode::DeleteChildren ()

Delete all the children of this node.

7.186.3.5 const XMLNode* tinyxml2::XMLNode::FirstChild () const [inline]

Get the first child node, or null if none exists.

7.186.3.6 XMLNode* tinyxml2::XMLNode::FirstChild () [inline]

7.186.3.7 const XMLElement* tinyxml2::XMLNode::FirstChildElement (const char * *name* = 0) const

Get the first child element, or optionally the first child element with the specified name.

7.186.3.8 XMLElement* tinyxml2::XMLNode::FirstChildElement (const char * *name* = 0) [inline]

7.186.3.9 const XMLDocument* tinyxml2::XMLNode::GetDocument () const [inline]

Get the [XMLDocument](#) that owns this [XMLNode](#).

7.186.3.10 `XMLDocument* tinyxml2::XMLNode::GetDocument() [inline]`

Get the [XMLDocument](#) that owns this [XMLNode](#).

7.186.3.11 `int tinyxml2::XMLNode::GetLineNum() const [inline]`

Gets the line number the node is in, if the document was parsed from a file.

7.186.3.12 `void* tinyxml2::XMLNode::GetUserData() const [inline]`

Get user data set into the [XMLNode](#). TinyXML-2 in no way processes or interprets user data. It is initially 0.

7.186.3.13 `XMLNode* tinyxml2::XMLNode::InsertAfterChild(XMLNode * afterThis, XMLNode * addThis)`

Add a node after the specified child node. If the child node is already part of the document, it is moved from its old location to the new location. Returns the addThis argument or 0 if the afterThis node is not a child of this node, or if the node does not belong to the same document.

7.186.3.14 `XMLNode* tinyxml2::XMLNode::InsertEndChild(XMLNode * addThis)`

Add a child node as the last (right) child. If the child node is already part of the document, it is moved from its old location to the new location. Returns the addThis argument or 0 if the node does not belong to the same document.

7.186.3.15 `XMLNode* tinyxml2::XMLNode::InsertFirstChild(XMLNode * addThis)`

Add a child node as the first (left) child. If the child node is already part of the document, it is moved from its old location to the new location. Returns the addThis argument or 0 if the node does not belong to the same document.

7.186.3.16 `const XMLNode* tinyxml2::XMLNode::LastChild() const [inline]`

Get the last child node, or null if none exists.

7.186.3.17 `XMLNode* tinyxml2::XMLNode::LastChild() [inline]`

7.186.3.18 `const XMLElement* tinyxml2::XMLNode::LastChildElement(const char * name = 0) const`

Get the last child element or optionally the last child element with the specified name.

7.186.3.19 `XMLElement* tinyxml2::XMLNode::LastChildElement(const char * name = 0) [inline]`

7.186.3.20 `XMLNode* tinyxml2::XMLNode::LinkEndChild(XMLNode * addThis) [inline]`

7.186.3.21 `const XMLNode* tinyxml2::XMLNode::NextSibling() const [inline]`

Get the next (right) sibling node of this node.

7.186.3.22 `XMLNode* tinyxml2::XMLNode::NextSibling() [inline]`

7.186.3.23 `const XMLElement* tinyxml2::XMLNode::NextSiblingElement(const char * name = 0) const`

Get the next (right) sibling element of this node, with an optionally supplied name.

7.186.3.24 `XMLElement* tinyxml2::XMLNode::NextSiblingElement(const char * name = 0) [inline]`

7.186.3.25 `bool tinyxml2::XMLNode::NoChildren() const [inline]`

Returns true if this node has no children.

7.186.3.26 `const XMLNode* tinyxml2::XMLNode::Parent() const [inline]`

Get the parent of this node on the DOM.

7.186.3.27 `XMLNode* tinyxml2::XMLNode::Parent() [inline]`

7.186.3.28 `virtual char* tinyxml2::XMLNode::ParseDeep(char * p, StrPair * parentEndTag, int * curLineNumPtr) [protected], [virtual]`

Reimplemented in [tinyxml2::XMLElement](#), [tinyxml2::XMLUnknown](#), [tinyxml2::XMLDeclaration](#), [tinyxml2::XMLComment](#), and [tinyxml2::XMLText](#).

7.186.3.29 `const XMLNode* tinyxml2::XMLNode::PreviousSibling() const [inline]`

Get the previous (left) sibling node of this node.

7.186.3.30 `XMLNode* tinyxml2::XMLNode::PreviousSibling() [inline]`

7.186.3.31 `const XMLElement* tinyxml2::XMLNode::PreviousSiblingElement(const char * name = 0) const`

Get the previous (left) sibling element of this node, with an optionally supplied name.

7.186.3.32 `XMLElement* tinyxml2::XMLNode::PreviousSiblingElement(const char * name = 0) [inline]`

7.186.3.33 `void tinyxml2::XMLNode::SetUserData(void * userData) [inline]`

Set user data into the [XMLNode](#). TinyXML-2 in no way processes or interprets user data. It is initially 0.

7.186.3.34 `void tinyxml2::XMLNode::SetValue(const char * val, bool staticMem = false)`

Set the Value of an XML node.

See Also

[Value\(\)](#)

7.186.3.35 **virtual XMLNode*** `tinyxml2::XMLNode::ShallowClone (XMLDocument * document) const [pure virtual]`

Make a copy of this node, but not its children. You may pass in a Document pointer that will be the owner of the new Node. If the 'document' is null, then the node returned will be allocated from the current Document. (`this->GetDocument()`)

Note: if called on a `XMLDocument`, this will return null.

Implemented in `tinyxml2::XMLDocument`, `tinyxml2::XMLElement`, `tinyxml2::XMLUnknown`, `tinyxml2::XMLDeclaration`, `tinyxml2::XMLComment`, and `tinyxml2::XMLText`.

7.186.3.36 **virtual bool** `tinyxml2::XMLNode::ShallowEqual (const XMLNode * compare) const [pure virtual]`

Test if 2 nodes are the same, but don't test children. The 2 nodes do not need to be in the same Document.

Note: if called on a `XMLDocument`, this will return false.

Implemented in `tinyxml2::XMLDocument`, `tinyxml2::XMLElement`, `tinyxml2::XMLUnknown`, `tinyxml2::XMLDeclaration`, `tinyxml2::XMLComment`, and `tinyxml2::XMLText`.

7.186.3.37 **virtual XMLComment*** `tinyxml2::XMLNode::ToComment () [inline], [virtual]`

Safely cast to a Comment, or null.

Reimplemented in `tinyxml2::XMLComment`.

7.186.3.38 **virtual const XMLComment*** `tinyxml2::XMLNode::ToComment () const [inline], [virtual]`

Reimplemented in `tinyxml2::XMLComment`.

7.186.3.39 **virtual XMLDeclaration*** `tinyxml2::XMLNode::ToDeclaration () [inline], [virtual]`

Safely cast to a Declaration, or null.

Reimplemented in `tinyxml2::XMLDeclaration`.

7.186.3.40 **virtual const XMLDeclaration*** `tinyxml2::XMLNode::ToDeclaration () const [inline], [virtual]`

Reimplemented in `tinyxml2::XMLDeclaration`.

7.186.3.41 **virtual XMLDocument*** `tinyxml2::XMLNode::ToDocument () [inline], [virtual]`

Safely cast to a Document, or null.

Reimplemented in `tinyxml2::XMLDocument`.

7.186.3.42 **virtual const XMLDocument*** `tinyxml2::XMLNode::ToDocument () const [inline], [virtual]`

Reimplemented in `tinyxml2::XMLDocument`.

7.186.3.43 **virtual XMLElement*** `tinyxml2::XMLNode::ToElement () [inline], [virtual]`

Safely cast to an Element, or null.

Reimplemented in `tinyxml2::XMLElement`.

7.186.3.44 `virtual const XMLElement* tinyxml2::XMLNode::ToElement() const [inline], [virtual]`

Reimplemented in [tinyxml2::XMLElement](#).

7.186.3.45 `virtual XMLText* tinyxml2::XMLNode::ToText() [inline], [virtual]`

Safely cast to Text, or null.

Reimplemented in [tinyxml2::XMLText](#).

7.186.3.46 `virtual const XMLText* tinyxml2::XMLNode::ToText() const [inline], [virtual]`

Reimplemented in [tinyxml2::XMLText](#).

7.186.3.47 `virtual XMLUnknown* tinyxml2::XMLNode::ToUnknown() [inline], [virtual]`

Safely cast to an Unknown, or null.

Reimplemented in [tinyxml2::XMLUnknown](#).

7.186.3.48 `virtual const XMLUnknown* tinyxml2::XMLNode::ToUnknown() const [inline], [virtual]`

Reimplemented in [tinyxml2::XMLUnknown](#).

7.186.3.49 `const char* tinyxml2::XMLNode::Value() const`

The meaning of 'value' changes for the specific type.

Document: empty (NULL is returned, not an empty string)
Element: name of the element
Comment: the comment text
Unknown: the tag contents
Text: the text string

7.186.4 Friends And Related Function Documentation

7.186.4.1 `friend class XMLDocument [friend]`

7.186.4.2 `friend class XMLElement [friend]`

7.186.5 Member Data Documentation

7.186.5.1 `XMLDocument* tinyxml2::XMLNode::_document [protected]`

7.186.5.2 `XMLNode* tinyxml2::XMLNode::_firstChild [protected]`

7.186.5.3 `XMLNode* tinyxml2::XMLNode::_lastChild [protected]`

7.186.5.4 `XMLNode* tinyxml2::XMLNode::_next [protected]`

7.186.5.5 `XMLNode* tinyxml2::XMLNode::_parent [protected]`

7.186.5.6 `int tinyxml2::XMLNode::_parseLineNum [protected]`

7.186.5.7 `XMLNode* tinyxml2::XMLNode::_prev` [protected]

7.186.5.8 `void* tinyxml2::XMLNode::_userData` [protected]

7.186.5.9 `StrPair tinyxml2::XMLNode::_value` [mutable], [protected]

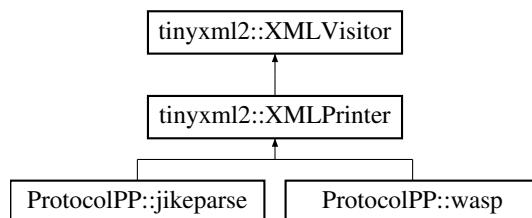
The documentation for this class was generated from the following file:

- [include/tinyxml2.h](#)

7.187 `tinyxml2::XMLPrinter` Class Reference

```
#include <tinyxml2.h>
```

Inheritance diagram for `tinyxml2::XMLPrinter`:



Public Member Functions

- `XMLPrinter` (const char *file, bool compact=false, int depth=2)
- `XMLPrinter` (FILE *file=0, bool compact=false, int depth=2)
- virtual `~XMLPrinter` ()
- void `PushHeader` (bool writeBOM, bool writeDeclaration)
- void `OpenElement` (const char *name, bool compactMode=false)
- void `PushAttribute` (const char *name, const char *value)

If streaming, add an attribute to an open element.
- void `PushAttribute` (const char *name, int value)
- void `PushAttribute` (const char *name, `ProtocolPP::jarray< uint8_t >` value)

If streaming, add an attribute to an open element. Added for ProtocolPP()
- void `PushAttribute` (const char *name, uint8_t value)

If streaming, add an attribute to an open element. Added for ProtocolPP()
- void `PushAttribute` (const char *name, uint16_t value)

If streaming, add an attribute to an open element. Added for ProtocolPP()
- void `PushAttribute` (const char *name, uint64_t value)

If streaming, add an attribute to an open element. Added for ProtocolPP()
- void `PushAttribute` (const char *name, bool value)
- void `PushAttribute` (const char *name, int64_t value)
- void `PushAttribute` (const char *name, double value)
- virtual void `CloseElement` (bool compactMode=false)

If streaming, close the Element.
- void `PushText` (const char *text, bool cdata=false)

Add a text node.
- void `PushText` (int value)

Add a text node from an integer.

- void [PushText](#) (`ProtocolIPP::jarray< uint8_t > value`)

Add a text node from an byte array. Added for ProtocolIPP()
- void [PushText](#) (`uint8_t value`)

Add a text node from an uint8_t. Added for ProtocolIPP()
- void [PushText](#) (`uint16_t value`)

Add a text node from an uint16_t. Added for ProtocolIPP()
- void [PushText](#) (`unsigned value`)

Add a text node from an unsigned.
- void [PushText](#) (`uint64_t value`)

Add a text node from an uint64_t. Added for ProtocolIPP()
- void [PushText](#) (`int64_t value`)

Add a text node from an unsigned.
- void [PushText](#) (`bool value`)

Add a text node from a bool.
- void [PushText](#) (`float value`)

Add a text node from a float.
- void [PushText](#) (`double value`)

Add a text node from a double.
- void [PushComment](#) (`const char *comment`)

Add a comment.
- void [PushDeclaration](#) (`const char *value`)
- void [PushUnknown](#) (`const char *value`)
- virtual bool [VisitEnter](#) (`const XMLDocument &`)

Visit a document.
- virtual bool [VisitExit](#) (`const XMLDocument &`)

Visit a document.
- virtual bool [VisitEnter](#) (`const XMLElement &element, const XMLAttribute *attribute`)

Visit an element.
- virtual bool [VisitExit](#) (`const XMLElement &element`)

Visit an element.
- virtual bool [Visit](#) (`const XMLText &text`)

Visit a text node.
- virtual bool [Visit](#) (`const XMLComment &comment`)

Visit a comment node.
- virtual bool [Visit](#) (`const XMLDeclaration &declaration`)

Visit a declaration.
- virtual bool [Visit](#) (`const XMLUnknown &unknown`)

Visit an unknown node.
- const char * [CStr](#) () const
- int [CStrSize](#) () const
- void [ClearBuffer](#) ()

Protected Member Functions

- virtual bool [CompactMode](#) (`const XMLElement &`)
- virtual void [PrintSpace](#) (`int depth`)
- void [Print](#) (`const char *format,...`)
- void [Write](#) (`const char *data, size_t size`)
- void [Write](#) (`const char *data`)
- void [Putc](#) (`char ch`)
- void [SealElementIfJustOpened](#) ()

Protected Attributes

- bool `_elementJustOpened`
- `DynArray< const char *, 10 > _stack`

7.187.1 Detailed Description

Printing functionality. The [XMLPrinter](#) gives you more options than the [XMLDocument::Print\(\)](#) method.

It can:

1. Print to memory.
2. Print to a file you provide.
3. Print XML without a [XMLDocument](#).

Print to Memory

```
XMLPrinter printer;
doc.Print( &printer );
SomeFunction( printer.CStr() );
```

Print to a File

You provide the file pointer.

```
XMLPrinter printer( fp );
doc.Print( &printer );
```

Print without a [XMLDocument](#)

When loading, an XML parser is very useful. However, sometimes when saving, it just gets in the way. The code is often set up for streaming, and constructing the DOM is just overhead.

The Printer supports the streaming case. The following code prints out a trivially simple XML file without ever creating an XML document.

```
XMLPrinter printer( fp );
printer.OpenElement( "foo" );
printer.PushAttribute( "foo", "bar" );
printer.CloseElement();
```

7.187.2 Constructor & Destructor Documentation

7.187.2.1 `tinyxml2::XMLPrinter::XMLPrinter(const char * file, bool compact = false, int depth = 2)`

Construct the printer. If the file is specified, this will open the file and print to it. Else it will print to memory, and the result is available in [CStr\(\)](#). If 'compact' is set to true, then output is created with only required whitespace and newlines Added for ProtocolPP()

7.187.2.2 `tinyxml2::XMLPrinter::XMLPrinter(FILE * file = 0, bool compact = false, int depth = 2)`

Construct the printer. If the FILE* is specified, this will print to the FILE. Else it will print to memory, and the result is available in [CStr\(\)](#). If 'compact' is set to true, then output is created with only required whitespace and newlines.

7.187.2.3 virtual tinyxml2::XMLPrinter::~XMLPrinter() [inline], [virtual]

7.187.3 Member Function Documentation

7.187.3.1 void tinyxml2::XMLPrinter::ClearBuffer() [inline]

If in print to memory mode, reset the buffer to the beginning.

7.187.3.2 virtual void tinyxml2::XMLPrinter::CloseElement(bool *compactMode* = false) [virtual]

If streaming, close the Element.

7.187.3.3 virtual bool tinyxml2::XMLPrinter::CompactMode(const XMLElement &) [inline], [protected], [virtual]

7.187.3.4 const char* tinyxml2::XMLPrinter::CStr() const [inline]

If in print to memory mode, return a pointer to the XML file in memory.

7.187.3.5 int tinyxml2::XMLPrinter::CStrSize() const [inline]

If in print to memory mode, return the size of the XML file in memory. (Note the size returned includes the terminating null.)

7.187.3.6 void tinyxml2::XMLPrinter::OpenElement(const char * *name*, bool *compactMode* = false)

If streaming, start writing an element. The element must be closed with [CloseElement\(\)](#)

7.187.3.7 void tinyxml2::XMLPrinter::Print(const char * *format*, ...) [protected]

7.187.3.8 virtual void tinyxml2::XMLPrinter::PrintSpace(int *depth*) [protected], [virtual]

Prints out the space before an element. You may override to change the space and tabs used. A [PrintSpace\(\)](#) override should call [Print\(\)](#).

7.187.3.9 void tinyxml2::XMLPrinter::PushAttribute(const char * *name*, const char * *value*)

If streaming, add an attribute to an open element.

7.187.3.10 void tinyxml2::XMLPrinter::PushAttribute(const char * *name*, int *value*)

7.187.3.11 void tinyxml2::XMLPrinter::PushAttribute(const char * *name*, ProtocolPP::jarray< uint8_t > *value*)

If streaming, add an attribute to an open element. Added for ProtocolPP()

7.187.3.12 void tinyxml2::XMLPrinter::PushAttribute(const char * *name*, uint8_t *value*)

If streaming, add an attribute to an open element. Added for ProtocolPP()

7.187.3.13 void tinyxml2::XMLPrinter::PushAttribute (*const char * name, uint16_t value*)

If streaming, add an attribute to an open element. Added for ProtocolPP()

7.187.3.14 void tinyxml2::XMLPrinter::PushAttribute (*const char * name, unsigned value*)

7.187.3.15 void tinyxml2::XMLPrinter::PushAttribute (*const char * name, uint64_t value*)

If streaming, add an attribute to an open element. Added for ProtocolPP()

7.187.3.16 void tinyxml2::XMLPrinter::PushAttribute (*const char * name, bool value*)

7.187.3.17 void tinyxml2::XMLPrinter::PushAttribute (*const char * name, int64_t value*)

7.187.3.18 void tinyxml2::XMLPrinter::PushAttribute (*const char * name, double value*)

7.187.3.19 void tinyxml2::XMLPrinter::PushComment (*const char * comment*)

Add a comment.

7.187.3.20 void tinyxml2::XMLPrinter::PushDeclaration (*const char * value*)

7.187.3.21 void tinyxml2::XMLPrinter::PushHeader (*bool writeBOM, bool writeDeclaration*)

If streaming, write the BOM and declaration.

7.187.3.22 void tinyxml2::XMLPrinter::PushText (*const char * text, bool cdata = false*)

Add a text node.

7.187.3.23 void tinyxml2::XMLPrinter::PushText (*int value*)

Add a text node from an integer.

7.187.3.24 void tinyxml2::XMLPrinter::PushText (*ProtocolPP::jarray< uint8_t > value*)

Add a text node from an byte array. Added for ProtocolPP()

7.187.3.25 void tinyxml2::XMLPrinter::PushText (*uint8_t value*)

Add a text node from an uint8_t. Added for ProtocolPP()

7.187.3.26 void tinyxml2::XMLPrinter::PushText (*uint16_t value*)

Add a text node from an uint16_t. Added for ProtocolPP()

7.187.3.27 void tinyxml2::XMLPrinter::PushText (*unsigned value*)

Add a text node from an unsigned.

7.187.3.28 void tinyxml2::XMLPrinter::PushText (*uint64_t value*)

Add a text node from an *uint64_t*. Added for ProtocolPP()

7.187.3.29 void tinyxml2::XMLPrinter::PushText (*int64_t value*)

Add a text node from an *unsigned*.

7.187.3.30 void tinyxml2::XMLPrinter::PushText (*bool value*)

Add a text node from a *bool*.

7.187.3.31 void tinyxml2::XMLPrinter::PushText (*float value*)

Add a text node from a *float*.

7.187.3.32 void tinyxml2::XMLPrinter::PushText (*double value*)

Add a text node from a *double*.

7.187.3.33 void tinyxml2::XMLPrinter::PushUnknown (*const char * value*)

7.187.3.34 void tinyxml2::XMLPrinter::Putc (*char ch*) [protected]

7.187.3.35 void tinyxml2::XMLPrinter::SealElementIfJustOpened () [protected]

7.187.3.36 virtual bool tinyxml2::XMLPrinter::Visit (*const XMLText &*) [virtual]

Visit a text node.

Reimplemented from [tinyxml2::XMLVisitor](#).

7.187.3.37 virtual bool tinyxml2::XMLPrinter::Visit (*const XMLComment &*) [virtual]

Visit a comment node.

Reimplemented from [tinyxml2::XMLVisitor](#).

7.187.3.38 virtual bool tinyxml2::XMLPrinter::Visit (*const XMLDeclaration &*) [virtual]

Visit a declaration.

Reimplemented from [tinyxml2::XMLVisitor](#).

7.187.3.39 virtual bool tinyxml2::XMLPrinter::Visit (*const XMLUnknown &*) [virtual]

Visit an unknown node.

Reimplemented from [tinyxml2::XMLVisitor](#).

7.187.3.40 virtual bool tinyxml2::XMLPrinter::VisitEnter (const XMLDocument &) [virtual]

Visit a document.

Reimplemented from [tinyxml2::XMLVisitor](#).

7.187.3.41 virtual bool tinyxml2::XMLPrinter::VisitEnter (const XMLElement & , const XMLAttribute *) [virtual]

Visit an element.

Reimplemented from [tinyxml2::XMLVisitor](#).

7.187.3.42 virtual bool tinyxml2::XMLPrinter::VisitExit (const XMLDocument &) [inline], [virtual]

Visit a document.

Reimplemented from [tinyxml2::XMLVisitor](#).

7.187.3.43 virtual bool tinyxml2::XMLPrinter::VisitExit (const XMLElement &) [virtual]

Visit an element.

Reimplemented from [tinyxml2::XMLVisitor](#).

7.187.3.44 void tinyxml2::XMLPrinter::Write (const char * data, size_t size) [protected]

7.187.3.45 void tinyxml2::XMLPrinter::Write (const char * data) [inline], [protected]

7.187.4 Member Data Documentation

7.187.4.1 bool tinyxml2::XMLPrinter::_elementJustOpened [protected]

7.187.4.2 DynArray<const char*, 10> tinyxml2::XMLPrinter::_stack [protected]

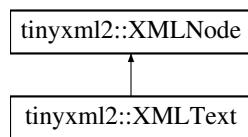
The documentation for this class was generated from the following file:

- [include/tinyxml2.h](#)

7.188 tinyxml2::XMLText Class Reference

```
#include <tinyxml2.h>
```

Inheritance diagram for tinyxml2::XMLText:



Public Member Functions

- virtual bool [Accept \(XMLVisitor *visitor\) const](#)
- virtual [XMLText * ToText \(\)](#)

- Safely cast to Text, or null.*
- virtual const `XMLText * ToText () const`
 - void `SetCData (bool isCData)`
Declare whether this should be CDATA or standard text.
 - bool `CData () const`
Returns true if this is a CDATA text element.
 - virtual `XMLNode * ShallowClone (XMLDocument *document) const`
 - virtual bool `ShallowEqual (const XMLNode *compare) const`

Protected Member Functions

- `XMLText (XMLDocument *doc)`
- virtual `~XMLText ()`
- char * `ParseDeep (char *p, StrPair *parentEndTag, int *curLineNumPtr)`

Friends

- class `XMLDocument`

Additional Inherited Members

7.188.1 Detailed Description

XML text.

Note that a text node can have child element nodes, for example:

```
<root>This is <b>bold</b></root>
```

A text node can have 2 ways to output the next. "normal" output and CDATA. It will default to the mode it was parsed from the XML file and you generally want to leave it alone, but you can change the output mode with `SetCData()` and query it with `CData()`.

7.188.2 Constructor & Destructor Documentation

7.188.2.1 `tinyxml2::XMLText::XMLText (XMLDocument * doc)` [inline], [explicit], [protected]

7.188.2.2 virtual `tinyxml2::XMLText::~XMLText ()` [inline], [protected], [virtual]

7.188.3 Member Function Documentation

7.188.3.1 virtual bool `tinyxml2::XMLText::Accept (XMLVisitor * visitor) const` [virtual]

Accept a hierarchical visit of the nodes in the TinyXML-2 DOM. Every node in the XML tree will be conditionally visited and the host will be called back via the `XMLVisitor` interface.

This is essentially a SAX interface for TinyXML-2. (Note however it doesn't re-parse the XML for the callbacks, so the performance of TinyXML-2 is unchanged by using this interface versus any other.)

The interface has been based on ideas from:

- <http://www.saxproject.org/>
- <http://c2.com/cgi/wiki?HierarchicalVisitorPattern>

Which are both good references for "visiting".

An example of using [Accept\(\)](#):

```
XMLPrinter printer;
tinyxmlDoc.Accept( &printer );
const char* xmlcstr = printer.CStr();
```

Implements [tinyxml2::XMLNode](#).

7.188.3.2 bool tinyxml2::XMLText::CData() const [inline]

Returns true if this is a CDATA text element.

7.188.3.3 char* tinyxml2::XMLText::ParseDeep(char * p, StrPair * parentEndTag, int * curLineNumPtr) [protected], [virtual]

Reimplemented from [tinyxml2::XMLNode](#).

7.188.3.4 void tinyxml2::XMLText::SetCData(bool isCData) [inline]

Declare whether this should be CDATA or standard text.

7.188.3.5 virtual XMLNode* tinyxml2::XMLText::ShallowClone(XMLDocument * document) const [virtual]

Make a copy of this node, but not its children. You may pass in a Document pointer that will be the owner of the new Node. If the 'document' is null, then the node returned will be allocated from the current Document. ([this->GetDocument\(\)](#))

Note: if called on a [XMLDocument](#), this will return null.

Implements [tinyxml2::XMLNode](#).

7.188.3.6 virtual bool tinyxml2::XMLText::ShallowEqual(const XMLNode * compare) const [virtual]

Test if 2 nodes are the same, but don't test children. The 2 nodes do not need to be in the same Document.

Note: if called on a [XMLDocument](#), this will return false.

Implements [tinyxml2::XMLNode](#).

7.188.3.7 virtual XMLText* tinyxml2::XMLText::ToText() [inline], [virtual]

Safely cast to Text, or null.

Reimplemented from [tinyxml2::XMLNode](#).

7.188.3.8 virtual const XMLText* tinyxml2::XMLText::ToText() const [inline], [virtual]

Reimplemented from [tinyxml2::XMLNode](#).

7.188.4 Friends And Related Function Documentation

7.188.4.1 friend class XMLDocument [friend]

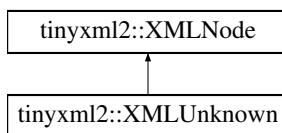
The documentation for this class was generated from the following file:

- include/tinyxml2.h

7.189 tinyxml2::XMLUnknown Class Reference

```
#include <tinyxml2.h>
```

Inheritance diagram for tinyxml2::XMLUnknown:



Public Member Functions

- virtual [XMLUnknown * ToUnknown \(\)](#)
Safely cast to an Unknown, or null.
- virtual const [XMLUnknown * ToUnknown \(\) const](#)
- virtual bool [Accept \(XMLVisitor *visitor\) const](#)
- virtual [XMLNode * ShallowClone \(XMLDocument *document\) const](#)
- virtual bool [ShallowEqual \(const XMLNode *compare\) const](#)

Protected Member Functions

- [XMLUnknown \(XMLDocument *doc\)](#)
- virtual [~XMLUnknown \(\)](#)
- char * [ParseDeep \(char *p, StrPair *parentEndTag, int *curLineNumPtr\)](#)

Friends

- class [XMLDocument](#)

Additional Inherited Members

7.189.1 Detailed Description

Any tag that TinyXML-2 doesn't recognize is saved as an unknown. It is a tag of text, but should not be modified. It will be written back to the XML, unchanged, when the file is saved.

DTD tags get thrown into XMLUnknowns.

7.189.2 Constructor & Destructor Documentation

7.189.2.1 [tinyxml2::XMLUnknown::XMLUnknown \(XMLDocument * doc \) \[explicit\], \[protected\]](#)

7.189.2.2 [virtual tinyxml2::XMLUnknown::~XMLUnknown \(\) \[protected\], \[virtual\]](#)

7.189.3 Member Function Documentation

7.189.3.1 virtual bool tinyxml2::XMLUnknown::Accept(XMLVisitor * visitor) const [virtual]

Accept a hierarchical visit of the nodes in the TinyXML-2 DOM. Every node in the XML tree will be conditionally visited and the host will be called back via the [XMLVisitor](#) interface.

This is essentially a SAX interface for TinyXML-2. (Note however it doesn't re-parse the XML for the callbacks, so the performance of TinyXML-2 is unchanged by using this interface versus any other.)

The interface has been based on ideas from:

- <http://www.saxproject.org/>
- <http://c2.com/cgi/wiki?HierarchicalVisitorPattern>

Which are both good references for "visiting".

An example of using [Accept\(\)](#):

```
XMLPrinter printer;
tinyxmlDoc.Accept( &printer );
const char* xmlcstr = printer.CStr();
```

Implements [tinyxml2::XMLNode](#).

7.189.3.2 char* tinyxml2::XMLUnknown::ParseDeep(char * p, StrPair * parentEndTag, int * curLineNumPtr) [protected], [virtual]

Reimplemented from [tinyxml2::XMLNode](#).

7.189.3.3 virtual XMLNode* tinyxml2::XMLUnknown::ShallowClone(XMLDocument * document) const [virtual]

Make a copy of this node, but not its children. You may pass in a Document pointer that will be the owner of the new Node. If the 'document' is null, then the node returned will be allocated from the current Document. (this->[GetDocument\(\)](#))

Note: if called on a [XMLDocument](#), this will return null.

Implements [tinyxml2::XMLNode](#).

7.189.3.4 virtual bool tinyxml2::XMLUnknown::ShallowEqual(const XMLNode * compare) const [virtual]

Test if 2 nodes are the same, but don't test children. The 2 nodes do not need to be in the same Document.

Note: if called on a [XMLDocument](#), this will return false.

Implements [tinyxml2::XMLNode](#).

7.189.3.5 virtual XMLUnknown* tinyxml2::XMLUnknown::ToUnknown() [inline], [virtual]

Safely cast to an Unknown, or null.

Reimplemented from [tinyxml2::XMLNode](#).

7.189.3.6 virtual const XMLUnknown* tinyxml2::XMLUnknown::ToUnknown() const [inline], [virtual]

Reimplemented from [tinyxml2::XMLNode](#).

7.189.4 Friends And Related Function Documentation

7.189.4.1 friend class XMLDocument [friend]

The documentation for this class was generated from the following file:

- [include/tinyxml2.h](#)

7.190 tinyxml2::XMLUtil Class Reference

```
#include <tinyxml2.h>
```

Static Public Member Functions

- static const char * [SkipWhiteSpace](#) (const char *p, int *curLineNumPtr)
- static char * [SkipWhiteSpace](#) (char *p, int *curLineNumPtr)
- static bool [IsWhiteSpace](#) (char p)
- static bool [IsNameStartChar](#) (unsigned char ch)
- static bool [IsNameChar](#) (unsigned char ch)
- static bool [StringEqual](#) (const char *p, const char *q, int nChar=INT_MAX)
- static bool [IsUTF8Continuation](#) (char p)
- static const char * [ReadBOM](#) (const char *p, bool *hasBOM)
- static const char * [GetCharacterRef](#) (const char *p, char *value, int *length)
- static void [ConvertUTF32ToUTF8](#) (unsigned long input, char *output, int *length)
- static void [ToStr](#) (int v, char *buffer, int bufferSize)
- static void [ToStr](#) ([ProtocolPP::jarray](#)< uint8_t > v, char *buffer, int bufferSize)
- static void [ToStr](#) (uint8_t v, char *buffer, int bufferSize)
- static void [ToStr](#) (uint16_t v, char *buffer, int bufferSize)
- static void [ToStr](#) (unsigned v, char *buffer, int bufferSize)
- static void [ToStr](#) (uint64_t v, char *buffer, int bufferSize)
- static void [ToStr](#) (bool v, char *buffer, int bufferSize)
- static void [ToStr](#) (float v, char *buffer, int bufferSize)
- static void [ToStr](#) (double v, char *buffer, int bufferSize)
- static void [ToStr](#) (int64_t v, char *buffer, int bufferSize)
- static bool [ToInt](#) (const char *str, int *value)
- static bool [ToArray](#) (const char *str, [ProtocolPP::jarray](#)< uint8_t > *value)
- static bool [ToByte](#) (const char *str, uint8_t *value)
- static bool [ToShort](#) (const char *str, uint16_t *value)
- static bool [ToUnsigned](#) (const char *str, unsigned *value)
- static bool [ToUnsignedU64](#) (const char *str, uint64_t *value)
- static bool [.ToBoolean](#) (const char *str, bool *value)
- static bool [ToFloat](#) (const char *str, float *value)
- static bool [.ToDouble](#) (const char *str, double *value)
- static bool [ToInt64](#) (const char *str, int64_t *value)
- static void [SetBoolSerialization](#) (const char *writeTrue, const char *writeFalse)

7.190.1 Member Function Documentation

- 7.190.1.1 `static void tinyxml2::XMLUtil::ConvertUTF32ToUTF8 (unsigned long input, char * output, int * length) [static]`
- 7.190.1.2 `static const char* tinyxml2::XMLUtil::GetCharacterRef (const char * p, char * value, int * length) [static]`
- 7.190.1.3 `static bool tinyxml2::XMLUtil::IsNameChar (unsigned char ch) [inline], [static]`
- 7.190.1.4 `static bool tinyxml2::XMLUtil::IsNameStartChar (unsigned char ch) [inline], [static]`
- 7.190.1.5 `static bool tinyxml2::XMLUtil::IsUTF8Continuation (char p) [inline], [static]`
- 7.190.1.6 `static bool tinyxml2::XMLUtil::IsWhiteSpace (char p) [inline], [static]`
- 7.190.1.7 `static const char* tinyxml2::XMLUtil::ReadBOM (const char * p, bool * hasBOM) [static]`
- 7.190.1.8 `static void tinyxml2::XMLUtil::SetBoolSerialization (const char * writeTrue, const char * writeFalse) [static]`
- 7.190.1.9 `static const char* tinyxml2::XMLUtil::SkipWhiteSpace (const char * p, int * curLineNumPtr) [inline], [static]`
- 7.190.1.10 `static char* tinyxml2::XMLUtil::SkipWhiteSpace (char * p, int * curLineNumPtr) [inline], [static]`
- 7.190.1.11 `static bool tinyxml2::XMLUtil::StringEqual (const char * p, const char * q, int nChar = INT_MAX) [inline], [static]`
- 7.190.1.12 `static bool tinyxml2::XMLUtil::ToArray (const char * str, ProtocolPP::jarray< uint8_t > * value) [static]`
- 7.190.1.13 `static bool tinyxml2::XMLUtil::ToBool (const char * str, bool * value) [static]`
- 7.190.1.14 `static bool tinyxml2::XMLUtil::ToByte (const char * str, uint8_t * value) [static]`
- 7.190.1.15 `static bool tinyxml2::XMLUtil::.ToDouble (const char * str, double * value) [static]`
- 7.190.1.16 `static bool tinyxml2::XMLUtil::ToFloat (const char * str, float * value) [static]`
- 7.190.1.17 `static bool tinyxml2::XMLUtil::ToInt (const char * str, int * value) [static]`
- 7.190.1.18 `static bool tinyxml2::XMLUtil::ToInt64 (const char * str, int64_t * value) [static]`
- 7.190.1.19 `static bool tinyxml2::XMLUtil::ToShort (const char * str, uint16_t * value) [static]`
- 7.190.1.20 `static void tinyxml2::XMLUtil::ToStr (int v, char * buffer, int bufferSize) [static]`
- 7.190.1.21 `static void tinyxml2::XMLUtil::ToStr (ProtocolPP::jarray< uint8_t > v, char * buffer, int bufferSize) [static]`
- 7.190.1.22 `static void tinyxml2::XMLUtil::ToStr (uint8_t v, char * buffer, int bufferSize) [static]`
- 7.190.1.23 `static void tinyxml2::XMLUtil::ToStr (uint16_t v, char * buffer, int bufferSize) [static]`
- 7.190.1.24 `static void tinyxml2::XMLUtil::ToStr (unsigned v, char * buffer, int bufferSize) [static]`
- 7.190.1.25 `static void tinyxml2::XMLUtil::ToStr (uint64_t v, char * buffer, int bufferSize) [static]`

- 7.190.1.26 static void tinyxml2::XMLUtil::ToStr (bool *v*, char * *buffer*, int *bufferSize*) [static]
- 7.190.1.27 static void tinyxml2::XMLUtil::ToStr (float *v*, char * *buffer*, int *bufferSize*) [static]
- 7.190.1.28 static void tinyxml2::XMLUtil::ToStr (double *v*, char * *buffer*, int *bufferSize*) [static]
- 7.190.1.29 static void tinyxml2::XMLUtil::ToStr (int64_t *v*, char * *buffer*, int *bufferSize*) [static]
- 7.190.1.30 static bool tinyxml2::XMLUtil::ToUnsigned (const char * *str*, unsigned * *value*) [static]
- 7.190.1.31 static bool tinyxml2::XMLUtil::ToUnsignedU64 (const char * *str*, uint64_t * *value*) [static]

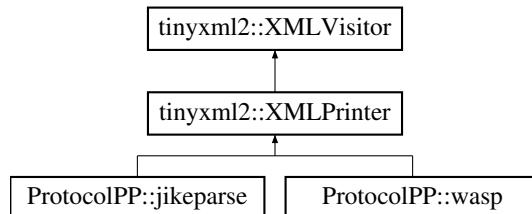
The documentation for this class was generated from the following file:

- include/tinyxml2.h

7.191 tinyxml2::XMLVisitor Class Reference

```
#include <tinyxml2.h>
```

Inheritance diagram for tinyxml2::XMLVisitor:



Public Member Functions

- virtual ~XMLVisitor ()
- virtual bool VisitEnter (const XMLDocument &)

Visit a document.
- virtual bool VisitExit (const XMLDocument &)

Visit a document.
- virtual bool VisitEnter (const XMLElement &, const XMLAttribute *)

Visit an element.
- virtual bool VisitExit (const XMLElement &)

Visit an element.
- virtual bool Visit (const XMLDeclaration &)

Visit a declaration.
- virtual bool Visit (const XMLText &)

Visit a text node.
- virtual bool Visit (const XMLComment &)

Visit a comment node.
- virtual bool Visit (const XMLUnknown &)

Visit an unknown node.

7.191.1 Detailed Description

Implements the interface to the "Visitor pattern" (see the Accept() method.) If you call the Accept() method, it requires being passed a [XMLVisitor](#) class to handle callbacks. For nodes that contain other nodes (Document, Element) you will get called with a VisitEnter/VisitExit pair. Nodes that are always leafs are simply called with [Visit\(\)](#).

If you return 'true' from a Visit method, recursive parsing will continue. If you return false, **no children of this node or its siblings** will be visited.

All flavors of Visit methods have a default implementation that returns 'true' (continue visiting). You need to only override methods that are interesting to you.

Generally Accept() is called on the [XMLDocument](#), although all nodes support visiting.

You should never change the document from a callback.

See Also

[XMLNode::Accept\(\)](#)

7.191.2 Constructor & Destructor Documentation

7.191.2.1 `virtual tinyxml2::XMLVisitor::~XMLVisitor() [inline], [virtual]`

7.191.3 Member Function Documentation

7.191.3.1 `virtual bool tinyxml2::XMLVisitor::Visit(const XMLDeclaration &) [inline], [virtual]`

Visit a declaration.

Reimplemented in [tinyxml2::XMLPrinter](#).

7.191.3.2 `virtual bool tinyxml2::XMLVisitor::Visit(const XMLText &) [inline], [virtual]`

Visit a text node.

Reimplemented in [tinyxml2::XMLPrinter](#).

7.191.3.3 `virtual bool tinyxml2::XMLVisitor::Visit(const XMLComment &) [inline], [virtual]`

Visit a comment node.

Reimplemented in [tinyxml2::XMLPrinter](#).

7.191.3.4 `virtual bool tinyxml2::XMLVisitor::Visit(const XMLUnknown &) [inline], [virtual]`

Visit an unknown node.

Reimplemented in [tinyxml2::XMLPrinter](#).

7.191.3.5 `virtual bool tinyxml2::XMLVisitor::VisitEnter(const XMLDocument &) [inline], [virtual]`

Visit a document.

Reimplemented in [tinyxml2::XMLPrinter](#).

7.191.3.6 `virtual bool tinyxml2::XMLVisitor::VisitEnter(const XMLElement &, const XMLAttribute *) [inline], [virtual]`

Visit an element.

Reimplemented in [tinyxml2::XMLPrinter](#).

7.191.3.7 `virtual bool tinyxml2::XMLVisitor::VisitExit(const XMLDocument &) [inline], [virtual]`

Visit a document.

Reimplemented in [tinyxml2::XMLPrinter](#).

7.191.3.8 `virtual bool tinyxml2::XMLVisitor::VisitExit(const XMLElement &) [inline], [virtual]`

Visit an element.

Reimplemented in [tinyxml2::XMLPrinter](#).

The documentation for this class was generated from the following file:

- [include/tinyxml2.h](#)

field type	field name	Example
direction_t	DIRECTION	set_field<ProtocolPP::direction_t>(ProtocolPP::field_t::DIRECTION, ProtocolPP::ENCAP)
iana_t	VERSION	set_field<ProtocolPP::iana_t>(-ProtocolPP::field_t::VERSION, ProtocolPP::ICMPV6)
icmppmsg_t	MESSAGE	set_field<ProtocolPP::icmppmsg_t>(ProtocolPP::field_t::MESSAGE, ProtocolPP::TIMEEXCEED)
icmpcode_t	CODE	set_field<ProtocolPP::icmpcode_t>(ProtocolPP::field_t::CODE, ProtocolPP::TTL_EXPIRE)
uint8_t	DSECN	set_field<uint8_t>(ProtocolPP::field_t::DSECN, 0xFF)
	TTLHOP	set_field<uint8_t>(ProtocolPP::field_t::TTLHOP, 0xAA)
	FLAGS	set_field<uint8_t>(ProtocolPP::field_t::FLAGS, 0xCC)
uint16_t	FRAGOFFSET	set_field<uint16_t>(ProtocolPP::field_t::FRAGOFFSET, 0x001E)
	ID	set_field<uint16_t>(ProtocolPP::field_t::ID, 0x01CC)
	LABEL	set_field<uint32_t>(ProtocolPP::field_t::LABEL, 0xA1CC)
jarray<uint8_t>	SOURCE	set_field<jarray<uint8_t>>(ProtocolPP::field_t::SOURCE, jarray<uint8_t>("0xAABBCCDD"))
	DESTINATION	set_field<jarray<uint8_t>>(ProtocolPP::field_t::DESTINATION, jarray<uint8_t>("0xEE11AABB"))

Table 7.9: ICMP SA Set Fields

Exchange Type	Value
RESERVED	0-33
IKE_SA_INIT	34
IKE_AUTH	35
CREATE_CHILD_SA	36
INFORMATIONAL	37
RESERVED TO IANA	38-239
Reserved for private use	240-255

Table 7.10: IKE Exchange Type

Next Payload Type	Notation	Value
No Next Payload	PYLD_NONE	0
Security Association	PYLD_SA	33
Key Exchange	PYLD_KE	34
Identification - Initiator	PYLD_IDI	35
Identification - Responder	PYLD_IDR	36
Certificate	PYLD_CERT	37
Certificate Request	PYLD_CERTREQ	38
Authentication	PYLD_AUTH	39
Nonce	PYLD_NINR	40
Notify	PYLD_N	41
Delete	PYLD_D	42
Vendor ID	PYLD_V	43
Traffic Selector Initiator	PYLD_TSI	44
Traffic Selector Responder	PYLD_TSR	45
Encrypted and Authenticated	PYLD_SK	46
Configuration	PYLD_CP	47
Extensible Authentication	PYLD_EAP	48
Generic Secure Password Method	PYLD_GSPM	49
Group Identification	PYLD_IDG	50
Group Security Association	PYLD_GSA	51
Key Download	PYLD_KD	52
Encrypted and Authenticated Fragment	PYLD_SKF	53
Puzzle Solution	PYLD_PS	54

Table 7.11: IKEv2 Payloads

Protocol	Identifier	Protocol ID
IKE	PROTO_IKE	1
AH	PROTO_AH	2
ESP	PROTO_ESP	3
FC_ESP_HDR	PROTO_FC_ESP_HDR	4
FC_CT_AUTH	PROTO_FC_CT_AUTH	5

Table 7.12: IKEv2 Protocol IDs

Description	Identifier	Transform Type	Used In
Encryption Algorithm (ENCR)	IKE_ENCR	1	IKE and ESP
Pseudorandom Function (PRF)	IKE_PRF	2	IKE
Integrity Algorithm (INTEG)	IKE_INTEG	3	IKE, AH, optional in ESP
Diffie-Hellman Group (D-H)	IKE_DH	4	IKE, optional in AH and ESP
Extended Sequence Numbers (ESN)	IKE_ESN	5	AH and ESP

Table 7.13: IKEv2 Transform Types

Description	Identifier	Diffie-Hellman Group Number
Diffie-Hellman MODP Group 768-bit	DH_MODP_768	1
Diffie-Hellman MODP Group 1024-bit	DH_MODP_1024	2
Diffie-Hellman MODP Group 1536-bit	DH_MODP_1536	5
Diffie-Hellman MODP Group 2048-bit	DH_MODP_2048	14
Diffie-Hellman MODP Group 3072-bit	DH_MODP_3072	15
Diffie-Hellman MODP Group 4096-bit	DH_MODP_4096	16
Diffie-Hellman MODP Group 6144-bit	DH_MODP_6144	17
Diffie-Hellman MODP Group 8192-bit	DH_MODP_8192	18
Diffie-Hellman Elliptic Curve with 256-bit generator	DH_ECP_256	19
Diffie-Hellman Elliptic Curve with 384-bit generator	DH_ECP_384	20
Diffie-Hellman Elliptic Curve with 521-bit generator	DH_ECP_521	21
Diffie-Hellman MODP Group 1024-bit with 160-bit Subgroup	DH_MODP_1024_160	22
Diffie-Hellman MODP Group 2048-bit with 224-bit Subgroup	DH_MODP_2048_224	23
Diffie-Hellman MODP Group 2048-bit with 256-bit Subgroup	DH_MODP_2048_256	24
Diffie-Hellman Elliptic Curve with 192-bit random curve	DH_ECP_192_RANDOM	25
Diffie-Hellman Elliptic Curve with 224-bit random curve	DH_ECP_224_RANDOM	26
Diffie-Hellman BrainPool Curve with 224-bit prime	DH_BRAINPOOL_P224R1	27
Diffie-Hellman BrainPool Curve with 256-bit prime	DH_BRAINPOOL_P256R1	28
Diffie-Hellman BrainPool Curve with 384-bit prime	DH_BRAINPOOL_P384R1	29
Diffie-Hellman BrainPool Curve with 512-bit prime	DH_BRAINPOOL_P512R1	30
Diffie-Hellman Elliptic Curve 25519	DH_CURVE_25519	31
Diffie-Hellman Elliptic Curve 448	DH_CURVE_448	32

Table 7.14: IKEv2 Diffie-Hellman Group Number

Description	Identifier	ID Type
IPv4 Address ID type	ID_IPV4_ADDR	1
FQDN ID type	ID_FQDN	2
RFC822 Address ID	ID_RFC822_ADDR	3
IPv6 Address ID type	ID_IPV6_ADDR	5
ASN1 DN ID	ID_DER ASN1 DN	9
ASN1 GN ID	ID_DER ASN1 GN	10
Key ID	ID_KEY_ID	11
FC Name ID	ID_FC_NAME	12
NULL ID	ID_NULL_ID	13

Table 7.15: IKEv2 Identification Type

Description	Identifier	CERT Encoding
Certificate encoded as X509_PKCS_7	CERT_X509_PKCS_7	1
Certificate encoded as PGP	CERT_PGP	2
Certificate encoded as DNS	CERT_DNS	3
Certificate encoded as X509_SIGNATURE	CERT_X509_SIGNATURE	4
Certificate encoded as Kerberos	CERT_KERBEROS	6
Certificate encoded as CRL	CERT_CRL	7
Certificate encoded as ARL	CERT_ARL	8
Certificate encoded as SPKI	CERT_SPKI	9
Certificate encoded as RAW_RSA	CERT_RAW_RSA	10
Certificate encoded as X509 Attribute	CERT_X509_ATTRIBUTE	11
Certificate encoded as HASH URL	CERT_HASH_URL	12
Certificate encoded as HASH URL Bundle	CERT_HASH_URL_BUNDLE	13
Certificate encoded as OSCP Content	CERT_OSCP_CONTENT	14
Certificate encoded as RAW Public Key	CERT_RAW_PUBLIC_KEY	15

Table 7.16: IKEv2 Certificate Encoding

Description	Identifier	AUTH Method
RSA Authentication	AUTH_RSA	1
Pre-Shared Key Authentication	AUTH_PSK	2
Digital Secure Signature Authentication	AUTH_DSS	3
Elliptic Curve with SHA-256 on the P-256 Curve Authentication	AUTH_ECDSA_P256	9
Elliptic Curve with SHA-384 on the P-384 Curve Authentication	AUTH_ECDSA_P384	10
Elliptic Curve with SHA-512 on the P-512 Curve Authentication	AUTH_ECDSA_P512	11
Generic Secure Password Authentication	AUTH_GENERIC_SECURE_PASSWORD	12
No PRF Authentication	AUTH_NULL	13
Digital Signature Authentication	AUTH_DIGITAL_SIGNATURE	14

Table 7.17: IKEv2 Authentication Method

Description	Identifier	CFG Type
Request the configuration	CFG_REQUEST	1
Reply to a configuration request	CFG_REPLY	2
Set the configuration	CFG_SET	3
Acknowledge the configuration request	CFG_ASK	4

Table 7.18: IKEv2 Configuration Types

Description	Identifier	CFG Attribute
Configure an internal IPv4 Address	CFG_INTERNAL_IP4_ADDRESS	1
Configure an internal IPv4 Netmask	CFG_INTERNAL_IP4_NETMASK	2
Configure an internal IPv4 Domain Name Service	CFG_INTERNAL_IP4_DNS	3
Configure an internal IPv4 NBNS setting	CFG_INTERNAL_IP4_NBNS	4
Configure an internal IPv4 DHCP	CFG_INTERNAL_IP4_DHCP	6
Configure application version	CFG_APPLICATION_VERSION	7
Configure an internal IPv6 Address	CFG_INTERNAL_IP6_ADDRESS	8
Configure an internal IPv6 Domain Name Service	CFG_INTERNAL_IP6_DNS	10
Configure an internal IPv6 DHCP	CFG_INTERNAL_IP6_DHCP	12
Configure an internal IPv4 Subnet	CFG_INTERNAL_IP4_SUBNET	13
Configure supported attributes	CFG_SUPPORTED_ATTRIBUTE-S	14
Configure an internal IPv6 Subnet	CFG_INTERNAL_IP6_SUBNET	15
Configure a MIP6 Home Prefix	CFG_MIP6_HOME_PREFIX	16
Configure an internal IPv6 Link	CFG_INTERNAL_IP6_LINK	17
Configure an internal IPv6 Prefix	CFG_INTERNAL_IP6_PREFIX	18
Configure a Home Agent Address	CFG_HOME_AGENT_ADDR	19
Configure a P CSCG IPv4 Address	CFG_P_CSCG_IP4_ADDR	20
Configure a P CSCF IPv6 Address	CFG_P_CSCF_IP6_ADDR	21
Configure a FTT KAT	CFG_FTT_KAT	22
Configure an External Source IPv4 NAT Information	CFG_EXT_SRC_IP4_NAT	23
Configure a Timeout Period for Liveness Check Number	CFG_TIMEOUT_LIVENESS	24
Configure an Internal DNS Domain	CFG_INTERNAL_DNS_DOMAIN	25
Configure an Internal DNS Secure TA	CFG_INTERNAL_DNS_SEC_TA	26

Table 7.19: IKEv2 Configuration Attribute Types

Exchange Type	Value
RESERVED	0-33
IKE_SA_INIT	34
IKE_AUTH	35
CREATE_CHILD_SA	36
INFORMATIONAL	37
RESERVED TO IANA	38-239
Reserved for private use	240-255

Table 7.20: IKE Exchange Type

Next Payload Type	Notation	Value
No Next Payload	PYLD_NONE	0
Security Association	PYLD_SA	33
Key Exchange	PYLD_KE	34
Identification - Initiator	PYLD_IDI	35
Identification - Responder	PYLD_IDR	36
Certificate	PYLD_CERT	37
Certificate Request	PYLD_CERTREQ	38
Authentication	PYLD_AUTH	39
Nonce	PYLD_NINR	40
Notify	PYLD_N	41
Delete	PYLD_D	42
Vendor ID	PYLD_V	43
Traffic Selector Initiator	PYLD_TSI	44
Traffic Selector Responder	PYLD_TSR	45
Encrypted and Authenticated	PYLD_SK	46
Configuration	PYLD_CP	47
Extensible Authentication	PYLD_EAP	48
Generic Secure Password Method	PYLD_GSPM	49
Group Identification	PYLD_IDG	50
Group Security Association	PYLD_GSA	51
Key Download	PYLD_KD	52
Encrypted and Authenticated Fragment	PYLD_SKF	53
Puzzle Solution	PYLD_PS	54

Table 7.21: IKEv2 Payloads

Protocol	Identifier	Protocol ID
IKE	PROTO_IKE	1
AH	PROTO_AH	2
ESP	PROTO_ESP	3
FC_ESP_HDR	PROTO_FC_ESP_HDR	4
FC_CT_AUTH	PROTO_FC_CT_AUTH	5

Table 7.22: IKEv2 Protocol IDs

Description	Identifier	Transform Type	Used In
Encryption Algorithm (ENCR)	IKE_ENCR	1	IKE and ESP
Pseudorandom Function (PRF)	IKE_PRF	2	IKE
Integrity Algorithm (INTEG)	IKE_INTEG	3	IKE, AH, optional in ESP
Diffie-Hellman Group (D-H)	IKE_DH	4	IKE, optional in AH and ESP
Extended Sequence Numbers (ESN)	IKE_ESN	5	AH and ESP

Table 7.23: IKEv2 Transform Types

Description	Identifier	Diffie-Hellman Group Number
Diffie-Hellman MODP Group 768-bit	DH_MODP_768	1
Diffie-Hellman MODP Group 1024-bit	DH_MODP_1024	2
Diffie-Hellman MODP Group 1536-bit	DH_MODP_1536	5
Diffie-Hellman MODP Group 2048-bit	DH_MODP_2048	14
Diffie-Hellman MODP Group 3072-bit	DH_MODP_3072	15
Diffie-Hellman MODP Group 4096-bit	DH_MODP_4096	16
Diffie-Hellman MODP Group 6144-bit	DH_MODP_6144	17
Diffie-Hellman MODP Group 8192-bit	DH_MODP_8192	18
Diffie-Hellman Elliptic Curve with 256-bit generator	DH_ECP_256	19
Diffie-Hellman Elliptic Curve with 384-bit generator	DH_ECP_384	20
Diffie-Hellman Elliptic Curve with 521-bit generator	DH_ECP_521	21
Diffie-Hellman MODP Group 1024-bit with 160-bit Subgroup	DH_MODP_1024_160	22
Diffie-Hellman MODP Group 2048-bit with 224-bit Subgroup	DH_MODP_2048_224	23
Diffie-Hellman MODP Group 2048-bit with 256-bit Subgroup	DH_MODP_2048_256	24
Diffie-Hellman Elliptic Curve with 192-bit random curve	DH_ECP_192_RANDOM	25
Diffie-Hellman Elliptic Curve with 224-bit random curve	DH_ECP_224_RANDOM	26
Diffie-Hellman BrainPool Curve with 224-bit prime	DH_BRAINPOOL_P224R1	27
Diffie-Hellman BrainPool Curve with 256-bit prime	DH_BRAINPOOL_P256R1	28
Diffie-Hellman BrainPool Curve with 384-bit prime	DH_BRAINPOOL_P384R1	29
Diffie-Hellman BrainPool Curve with 512-bit prime	DH_BRAINPOOL_P512R1	30
Diffie-Hellman Elliptic Curve 25519	DH_CURVE_25519	31
Diffie-Hellman Elliptic Curve 448	DH_CURVE_448	32

Table 7.24: IKEv2 Diffie-Hellman Group Number

Description	Identifier	ID Type
IPv4 Address ID type	ID_IPV4_ADDR	1
FQDN ID type	ID_FQDN	2
RFC822 Address ID	ID_RFC822_ADDR	3
IPv6 Address ID type	ID_IPV6_ADDR	5
ASN1 DN ID	ID_DER ASN1 DN	9
ASN1 GN ID	ID_DER ASN1 GN	10
Key ID	ID_KEY_ID	11
FC Name ID	ID_FC_NAME	12
NULL ID	ID_NULL_ID	13

Table 7.25: IKEv2 Identification Type

Description	Identifier	CERT Encoding
Certificate encoded as X509_PKCS_7	CERT_X509_PKCS_7	1
Certificate encoded as PGP	CERT_PGP	2
Certificate encoded as DNS	CERT_DNS	3
Certificate encoded as X509_SIGNATURE	CERT_X509_SIGNATURE	4
Certificate encoded as Kerberos	CERT_KERBEROS	6
Certificate encoded as CRL	CERT_CRL	7
Certificate encoded as ARL	CERT_ARL	8
Certificate encoded as SPKI	CERT_SPKI	9
Certificate encoded as RAW_RSA	CERT_RAW_RSA	10
Certificate encoded as X509 Attribute	CERT_X509_ATTRIBUTE	11
Certificate encoded as HASH URL	CERT_HASH_URL	12
Certificate encoded as HASH URL Bundle	CERT_HASH_URL_BUNDLE	13
Certificate encoded as OSCP Content	CERT_OSCP_CONTENT	14
Certificate encoded as RAW Public Key	CERT_RAW_PUBLIC_KEY	15

Table 7.26: IKEv2 Certificate Encoding

Chapter 8

File Documentation

8.1 drivers/include/jdirectdrive.h File Reference

```
#include <time.h>
#include <netdb.h>
#include <unistd.h>
#include <sys/wait.h>
#include <exception>
#include "StdCapture.h"
#include "jdata.h"
#include "jpacket.h"
#include "jstream.h"
#include "jlogger.h"
#include "jrинг.h"
#include "jmmu.h"
#include "jprotocolpp.h"
#include <stdio.h>
#include <string.h>
#include <malloc.h>
#include <errno.h>
#include <sys/socket.h>
#include <sys/types.h>
#include <sys/ioctl.h>
#include <net/if.h>
#include <netinet/in.h>
#include <netinet/ip.h>
#include <netinet/if_ether.h>
#include <netinet/udp.h>
#include <linux/if_packet.h>
#include <arpa/inet.h>
```

Classes

- class [DriverPP::jdirectdrive](#)

Namespaces

- [DriverPP](#)

Variables

- class [DriverPP::jdirectdrive](#) [DriverPP::jsgt](#)

8.2 drivers/include/jdrive.h File Reference

```
#include <time.h>
#include <netdb.h>
#include <unistd.h>
#include <sys/wait.h>
#include <exception>
#include <keyutils.h>
#include "jdata.h"
#include "jpacket.h"
#include "jstream.h"
#include "jlogger.h"
#include "jrинг.h"
#include "jmmu.h"
#include "jprotocolpp.h"
#include <stdio.h>
#include <string.h>
#include <malloc.h>
#include <errno.h>
#include <sys/socket.h>
#include <sys/types.h>
#include <sys/ioctl.h>
#include <net/if.h>
#include <netinet/in.h>
#include <netinet/ip.h>
#include <netinet/if_ether.h>
#include <netinet/udp.h>
#include <linux/if_packet.h>
#include <arpa/inet.h>
```

Classes

- class [DriverPP::jdrive](#)

Namespaces

- [DriverPP](#)

8.3 drivers/include/jdriver.h File Reference

```
#include <time.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>
#include <unistd.h>
#include <sys/wait.h>
#include <exception>
#include "interfacepp.h"
```

Classes

- class [DriverPP::jdriver< Q >](#)

Namespaces

- [DriverPP](#)

8.4 drivers/include/jringdrive.h File Reference

```
#include <time.h>
#include <netdb.h>
#include <unistd.h>
#include <sys/wait.h>
#include <exception>
#include <keyutils.h>
#include "StdCapture.h"
#include "jdata.h"
#include "jpacket.h"
#include "jstream.h"
#include "jlogger.h"
#include "jring.h"
#include "jmmu.h"
#include "jprotocolpp.h"
#include <stdio.h>
#include <string.h>
#include <malloc.h>
#include <errno.h>
#include <sys/socket.h>
#include <sys/types.h>
#include <sys/ioctl.h>
#include <net/if.h>
#include <netinet/in.h>
#include <netinet/ip.h>
#include <netinet/if_ether.h>
#include <netinet/udp.h>
#include <linux/if_packet.h>
#include <arpa/inet.h>
```

Classes

- class [DriverPP::jringdrive](#)

Namespaces

- [DriverPP](#)

8.5 drivers/include/StdCapture.h File Reference

```
#include <unistd.h>
#include <fcntl.h>
#include <stdio.h>
#include <mutex>
```

Classes

- class [StdCapture](#)

8.6 include/aead_chacha_poly1305.h File Reference

```
#include <vector>
#include <iostream>
#include "aes.h"
#include "poly1305.h"
#include "chacha.h"
#include "chacha20.h"
```

Classes

- class [ProtocolPP::aead_chacha_poly1305](#)

Namespaces

- [ProtocolPP](#)

8.7 include/chacha20.h File Reference

```
#include <assert.h>
#include <stddef.h>
#include <stdint.h>
#include <cstring>
#include <iostream>
```

Classes

- class [ProtocolPP::chacha20](#)

Namespaces

- [ProtocolPP](#)

8.8 include/ciphers.h File Reference

```
#include "jmodes.h"
```

Classes

- class [ProtocolPP::ciphers](#)

Namespaces

- [ProtocolPP](#)

8.9 include/config.h File Reference

Library configuration file.

Macros

- `#define IS_LITTLE_ENDIAN`
- `#define NO_OS_DEPENDENCE`
- `#define USE_MS_CRYPTOAPI`
- `#define PROTOCOLPP_NO_UNALIGNED_DATA_ACCESS`
- `#define PROTOCOLPP_VERSION 565`
- `#define PROTOCOLPP_DATA_DIR ""`
- `#define GZIP_OS_CODE 0`
- `#define PREFER_BERKELEY_STYLE_SOCKETS`
- `#define PROTOCOLPP_RIJNDAEL_NAME "AES"`
- `#define PROTOCOLPP_INIT_PRIORITY 250`
- `#define PROTOCOLPP_USER_PRIORITY (PROTOCOLPP_INIT_PRIORITY + 101)`
- `#define WORKAROUND_MS_BUG_Q258000`
- `#define NAMESPACE_BEGIN(x) namespace x {`
- `#define NAMESPACE_END }`
- `#define DOCUMENTED_TYPEDEF(x, y) typedef x y;`
- `#define ANONYMOUS_NAMESPACE_BEGIN namespace {`
- `#define ANONYMOUS_NAMESPACE_END }`
- `#define USING_NAMESPACE(x) using namespace x;`
- `#define DOCUMENTED_NAMESPACE_BEGIN(x) namespace x {`
- `#define DOCUMENTED_NAMESPACE_END }`
- `#define TYPE_OF_SOCKLEN_T ::socklen_t`
- `#define W64LIT(x) x##ULL`
- `#define PROTOCOLPP_NATIVE_DWORD_AVAILABLE 1`
- `#define PROTOCOLPP_BOOL_SLOW_WORD64 1`
- `#define PROTOCOLPP_L1_CACHE_LINE_SIZE 32`
- `#define PROTOCOLPP_ALIGN_DATA(x)`
- `#define PROTOCOLPP_SECTION_ALIGN16`
- `#define PROTOCOLPP_SECTION_INIT`
- `#define PROTOCOLPP_FASTCALL`
- `#define CPP_TYPENAME typename`
- `#define PROTOCOLPP_VC6_INT64`
- `#define PROTOCOLPP_NO_VTABLE`
- `#define PROTOCOLPP_UNCAUGHT_EXCEPTION_AVAILABLE`

- `#define PROTOCOLPP_BOOL_SSE2_INTRINSICS_AVAILABLE 0`
- `#define PROTOCOLPP_BOOL_SSE4_INTRINSICS_AVAILABLE 0`
- `#define PROTOCOLPP_BOOL_AESNI_INTRINSICS_AVAILABLE 0`
- `#define PROTOCOLPP_BOOL_AVX_AVAILABLE 0`
- `#define PROTOCOLPP_BOOL_ALIGN16 0`
- `#define PROTOCOLPP_NO_ALIGNED_ALLOC`
- `#define PROTOCOLPP_NOINLINE_DOTDOTDOT ...`
- `#define PROTOCOLPP_NOINLINE`
- `#define PROTOCOLPP_CONSTANT(x) static const int x;`
- `#define PROTOCOLPP_BOOL_X32 0`
- `#define PROTOCOLPP_BOOL_X86 0`
- `#define PROTOCOLPP_BOOL_X64 0`
- `#define PROTOCOLPP_BOOL_ARM32 0`
- `#define PROTOCOLPP_BOOL_ARM64 0`
- `#define PROTOCOLPP_DLL`
- `#define PROTOCOLPP_API`
- `#define PROTOCOLPP_EXTERN_DLL_TEMPLATE_CLASS extern template class PROTOCOLPP_DLL`
- `#define PROTOCOLPP_DLL_TEMPLATE_CLASS PROTOCOLPP_EXTERN_DLL_TEMPLATE_CLASS`
- `#define PROTOCOLPP_EXTERN_STATIC_TEMPLATE_CLASS extern template class PROTOCOLPP_STATIC_TEMPLATE_CLASS`
- `#define PROTOCOLPP_STATIC_TEMPLATE_CLASS PROTOCOLPP_EXTERN_STATIC_TEMPLATE_CLASS`
- `#define PROTOCOLPP_UNUSED(x) ((void)(x))`
- `#define PROTOCOLPP_DEPRECATED(msg)`
- `#define PROTOCOLPP_THROW`
- `#define PROTOCOLPP_NO_THROW`
- `#define PROTOCOLPP_CONSTEXPR`

Typedefs

- `typedef unsigned char byte`
- `typedef unsigned short word16`
- `typedef unsigned int word32`
- `typedef unsigned long long word64`
- `typedef word64 lword`
- `typedef word16 hword`
- `typedef word32 word`
- `typedef word64 dword`

Variables

- `const lword LWORD_MAX = W64LIT(0xffffffffffffffffffff)`
- `const unsigned int WORD_SIZE = sizeof(word)`
- `const unsigned int WORD_BITS = WORD_SIZE * 8`

8.9.1 Detailed Description

Library configuration file.

8.9.2 Macro Definition Documentation

```
8.9.2.1 #define ANONYMOUS_NAMESPACE_BEGIN namespace {  
  
8.9.2.2 #define ANONYMOUS_NAMESPACE_END }  
  
8.9.2.3 #define CPP_TYPENAME typename  
  
8.9.2.4 #define DOCUMENTED_NAMESPACE_BEGIN( x ) namespace x {  
  
8.9.2.5 #define DOCUMENTED_NAMESPACE_END }  
  
8.9.2.6 #define DOCUMENTED_TYPEDEF( x, y ) typedef x y;  
  
8.9.2.7 #define GZIP_OS_CODE 0  
  
8.9.2.8 #define IS_LITTLE_ENDIAN  
  
8.9.2.9 #define NAMESPACE_BEGIN( x ) namespace x {  
  
8.9.2.10 #define NAMESPACE_END }  
  
8.9.2.11 #define NO_OS_DEPENDENCE  
  
8.9.2.12 #define PREFER_BERKELEY_STYLE_SOCKETS  
  
8.9.2.13 #define PROTOCOLPP_ALIGN_DATA( x )  
  
8.9.2.14 #define PROTOCOLPP_API  
  
8.9.2.15 #define PROTOCOLPP_BOOL_AESNI_INTRINSICS_AVAILABLE 0  
  
8.9.2.16 #define PROTOCOLPP_BOOL_ALIGN16 0  
  
8.9.2.17 #define PROTOCOLPP_BOOL_ARM32 0  
  
8.9.2.18 #define PROTOCOLPP_BOOL_ARM64 0  
  
8.9.2.19 #define PROTOCOLPP_BOOL_AVX_AVAILABLE 0  
  
8.9.2.20 #define PROTOCOLPP_BOOL_SLOW_WORD64 1  
  
8.9.2.21 #define PROTOCOLPP_BOOL_SSE2_INTRINSICS_AVAILABLE 0  
  
8.9.2.22 #define PROTOCOLPP_BOOL_SSE4_INTRINSICS_AVAILABLE 0  
  
8.9.2.23 #define PROTOCOLPP_BOOL_X32 0  
  
8.9.2.24 #define PROTOCOLPP_BOOL_X64 0  
  
8.9.2.25 #define PROTOCOLPP_BOOL_X86 0  
  
8.9.2.26 #define PROTOCOLPP_CONSTANT( x ) static const int x;  
  
8.9.2.27 #define PROTOCOLPP_CONSTEXPR
```

```
8.9.2.28 #define PROTOCOLPP_DATA_DIR ""

8.9.2.29 #define PROTOCOLPP_DEPRECATED( msg )

8.9.2.30 #define PROTOCOLPP_DLL

8.9.2.31 #define PROTOCOLPP_DLL_TEMPLATE_CLASS PROTOCOLPP_EXTERN_DLL_TEMPLATE_CLASS

8.9.2.32 #define PROTOCOLPP_EXTERN_DLL_TEMPLATE_CLASS extern template class PROTOCOLPP_DLL

8.9.2.33 #define PROTOCOLPP_EXTERN_STATIC_TEMPLATE_CLASS extern template class

8.9.2.34 #define PROTOCOLPP_FASTCALL

8.9.2.35 #define PROTOCOLPP_INIT_PRIORITY 250

8.9.2.36 #define PROTOCOLPP_L1_CACHE_LINE_SIZE 32

8.9.2.37 #define PROTOCOLPP_NATIVE_DWORD_AVAILABLE 1

8.9.2.38 #define PROTOCOLPP_NO_ALIGNED_ALLOC

8.9.2.39 #define PROTOCOLPP_NO_THROW

8.9.2.40 #define PROTOCOLPP_NO_UNALIGNED_DATA_ACCESS

8.9.2.41 #define PROTOCOLPP_NO_VTABLE

8.9.2.42 #define PROTOCOLPP_NOINLINE

8.9.2.43 #define PROTOCOLPP_NOINLINE_DOTDOTDOT ...

8.9.2.44 #define PROTOCOLPP_RIJNDAEL_NAME "AES"

8.9.2.45 #define PROTOCOLPP_SECTION_ALIGN16

8.9.2.46 #define PROTOCOLPP_SECTION_INIT

8.9.2.47 #define PROTOCOLPP_STATIC_TEMPLATE_CLASS PROTOCOLPP_EXTERN_STATIC_TEMPLATE_CLASS

8.9.2.48 #define PROTOCOLPP_THROW

8.9.2.49 #define PROTOCOLPP_UNCAUGHT_EXCEPTION_AVAILABLE

8.9.2.50 #define PROTOCOLPP_UNUSED( x ) ((void)(x))

8.9.2.51 #define PROTOCOLPP_USER_PRIORITY (PROTOCOLPP_INIT_PRIORITY + 101)

8.9.2.52 #define PROTOCOLPP_VC6_INT64

8.9.2.53 #define PROTOCOLPP_VERSION 565

8.9.2.54 #define TYPE_OF_SOCKLEN_T ::socklen_t

8.9.2.55 #define USE_MS_CRYPTOAPI
```

8.9.2.56 #define USING_NAMESPACE(*x*) using namespace *x*;

8.9.2.57 #define W64LIT(*x*) *x*##ULL

8.9.2.58 #define WORKAROUND_MS_BUG_Q258000

8.9.3 Typedef Documentation

8.9.3.1 typedef unsigned char byte

8.9.3.2 typedef word64 dword

8.9.3.3 typedef word16 hword

8.9.3.4 typedef word64 lword

8.9.3.5 typedef word32 word

8.9.3.6 typedef unsigned short word16

8.9.3.7 typedef unsigned int word32

8.9.3.8 typedef unsigned long long word64

8.9.4 Variable Documentation

8.9.4.1 const lword LWORD_MAX = W64LIT(0xffffffffffffffffffff)

8.9.4.2 const unsigned int WORD_BITS = WORD_SIZE * 8

8.9.4.3 const unsigned int WORD_SIZE = sizeof(word)

8.10 include/EnumString.h File Reference

```
#include <string>
#include <map>
#include <cassert>
#include <time.h>
```

Classes

- class [ProtocolPP::EnumStringBase< DerivedType, EnumType >](#)
- struct [ProtocolPP::EnumString< EnumType >](#)

Namespaces

- [ProtocolPP](#)

Macros

- #define Begin_Enum_String(*EnumerationName*)
- #define Enum_String(*EnumeratorName*) RegisterEnumerator(*EnumeratorName*, #*EnumeratorName*);
- #define End_Enum_String }

8.10.1 Macro Definition Documentation

8.10.1.1 `#define Begin_Enum_String(EnumerationName)`

Value:

```
template <> struct EnumString<EnumerationName> :
    public EnumStringBase< EnumString<EnumerationName>, EnumerationName >
{
    static void RegisterEnumerators()
```

8.10.1.2 `#define End_Enum_String }`

8.10.1.3 `#define Enum_String(EnumeratorName) RegisterEnumerator(EnumeratorName, #EnumeratorName);`

8.11 include/jarray.h File Reference

```
#include <vector>
#include <iostream>
#include <sstream>
#include <string>
#include <regex>
#include <iomanip>
#include "jenum.h"
```

Classes

- class [ProtocolPP::jarray< T >](#)

Namespaces

- [ProtocolPP](#)

8.12 include/jdata.h File Reference

```
#include <memory>
#include "jstream.h"
#include "jpacket.h"
#include "jprotocol.h"
```

Classes

- class [ProtocolPP::jdata](#)

Namespaces

- [ProtocolPP](#)

8.13 include/jdsa.h File Reference

```
#include <sys/types.h>
#include <unistd.h>
#include <string.h>
#include <crypt.h>
#include <memory>
#include "jlogger.h"
#include "jenum.h"
#include "jarray.h"
#include "dsa.h"
#include "nbtheory.h"
#include "osrng.h"
#include "dh.h"
#include "sha.h"
#include "secblock.h"
#include "integer.h"
```

Classes

- class [ProtocolPP::jdsa](#)

Namespaces

- [ProtocolPP](#)

8.14 include/jecdsa.h File Reference

```
#include <sys/types.h>
#include <unistd.h>
#include <string.h>
#include <crypt.h>
#include <memory>
#include "jlogger.h"
#include "jenum.h"
#include "jarray.h"
#include "nbtheory.h"
#include "osrng.h"
#include "aes.h"
#include "md5.h"
#include "sha.h"
#include "dh.h"
#include "eccrypto.h"
#include "ecp.h"
#include "ec2n.h"
#include "rsa.h"
#include "dsa.h"
#include "pssr.h"
#include "secblock.h"
#include "oids.h"
#include "asn.h"
#include "integer.h"
```

Classes

- class [ProtocolPP::jecdsa](#)

Namespaces

- [ProtocolPP](#)

Macros

- `#define CRYPTOPP_ENABLE_NAMESPACE_WEAK 1`

8.14.1 Macro Definition Documentation

8.14.1.1 #define CRYPTOPP_ENABLE_NAMESPACE_WEAK 1

8.15 include/jenum.h File Reference

```
#include "EnumString.h"
```

Namespaces

- [ProtocolPP](#)

Macros

- `#define HEX(a, b) "0x" << std::setw(b) << std::setfill('0') << std::hex << static_cast<unsigned long long>(a) << std::dec`
- `#define HEXLOG(a, b) "0x", std::setw(b), std::setfill('0'), std::hex, static_cast<unsigned long long>(a), std::dec`

Enumerations

- enum [ProtocolPP::field_t](#) {
 [ProtocolPP::DIRECTION](#), [ProtocolPP::VERSION](#), [ProtocolPP::MODE](#), [ProtocolPP::SPI](#),
 [ProtocolPP::SEQNUM](#), [ProtocolPP::EXTSEQNUM](#), [ProtocolPP::ARLEN](#), [ProtocolPP::ARWIN](#),
 [ProtocolPP::CIPHER](#), [ProtocolPP::CKEYLEN](#), [ProtocolPP::CIPHERKEY](#), [ProtocolPP::AUTH](#),
 [ProtocolPP::AKEYLEN](#), [ProtocolPP::AUTHKEY](#), [ProtocolPP::IVLEN](#), [ProtocolPP::IV](#),
 [ProtocolPP::SALTLEN](#), [ProtocolPP::SALT](#), [ProtocolPP::BYTECNT](#), [ProtocolPP::LIFETIME](#),
 [ProtocolPP::SEQNUMOVRFLW](#), [ProtocolPP::STATEFULFRAG](#), [ProtocolPP::BYPASSDF](#), [ProtocolPP::BYPASSDSCP](#),
 [ProtocolPP::NAT](#), [ProtocolPP::NCHK](#), [ProtocolPP::DSECN](#), [ProtocolPP::TTLHOP](#),
 [ProtocolPP::FLAGS](#), [ProtocolPP::NATSRC](#), [ProtocolPP::NATDST](#), [ProtocolPP::ID](#),
 [ProtocolPP::LABEL](#), [ProtocolPP::FRAGOFFSET](#), [ProtocolPP::MOREFRAG](#), [ProtocolPP::FRAGID](#),
 [ProtocolPP::MTU](#), [ProtocolPP::SOURCE](#), [ProtocolPP::DESTINATION](#), [ProtocolPP::EXTHDR](#),
 [ProtocolPP::NH](#), [ProtocolPP::ICVLEN](#), [ProtocolPP::HDRLEN](#), [ProtocolPP::TFCLEN](#),
 [ProtocolPP::USEXT](#), [ProtocolPP::RANDIV](#), [ProtocolPP::NODEJUMBO](#), [ProtocolPP::AUDIT](#),
 [ProtocolPP::AUDITLOG](#), [ProtocolPP::CHECKSUM](#), [ProtocolPP::LENGTH](#), [ProtocolPP::TYPE](#),
 [ProtocolPP::CODE](#), [ProtocolPP::POINTER](#), [ProtocolPP::IDENTIFIER](#), [ProtocolPP::GATEWAY](#),
 [ProtocolPP::ORIGTIMESTAMP](#), [ProtocolPP::RXTIMESTAMP](#), [ProtocolPP::TXTIMESTAMP](#), [ProtocolPP::M-](#)
 }

```

ESSAGE,
ProtocolPP::ICMPCODE, ProtocolPP::DATACTRL, ProtocolPP::PDUTYPE, ProtocolPP::POLLBIT,
ProtocolPP::EXTENSION, ProtocolPP::RSN, ProtocolPP::SNLEN, ProtocolPP::HDREXT,
ProtocolPP::LENGTHIND, ProtocolPP::HFNI, ProtocolPP::SUFU, ProtocolPP::FMS,
ProtocolPP::BITMAPLEN, ProtocolPP::BITMAP, ProtocolPP::PGKINDEX, ProtocolPP::PTKINDENT,
ProtocolPP::SDUTYPE, ProtocolPP::KDID, ProtocolPP::NMP, ProtocolPP::HRW,
ProtocolPP::BEARER, ProtocolPP::FRESH, ProtocolPP::ETHERTYPE, ProtocolPP::SL,
ProtocolPP::TCIAN, ProtocolPP::PN, ProtocolPP::XPN, ProtocolPP::SCI,
ProtocolPP::SSCI, ProtocolPP::SAKEY, ProtocolPP::ENRECEIVE, ProtocolPP::ENTRANSMIT,
ProtocolPP::PROTECTFRAMES, ProtocolPP::INUSE, ProtocolPP::CREATETIME, ProtocolPP::STARTTIME,
ProtocolPP::STOPTIME, ProtocolPP::PADDING, ProtocolPP::CC, ProtocolPP::MARKER,
ProtocolPP::PT, ProtocolPP::ROC, ProtocolPP::TIMESTAMP, ProtocolPP::SSRC,
ProtocolPP::CSRC, ProtocolPP::BLKSIZE, ProtocolPP::MKI, ProtocolPP::MKILEN,
ProtocolPP::MKIDATA, ProtocolPP::ACKNUM, ProtocolPP::OFFSET, ProtocolPP::RCVWINDOW,
ProtocolPP::URGENT, ProtocolPP::STATE, ProtocolPP::OPTIONS, ProtocolPP::SEGLEN,
ProtocolPP::PRECEDENCE, ProtocolPP::SNDUNA, ProtocolPP::SNDNXT, ProtocolPP::SNDWND,
ProtocolPP::SNDUP, ProtocolPP:: SNDW1, ProtocolPP::SNDW2, ProtocolPP::ISS,
ProtocolPP::RCVNXT, ProtocolPP::RCVWND, ProtocolPP::RCVUP, ProtocolPP::IRS,
ProtocolPP::TCPTIMEOUT, ProtocolPP::EPOCH, ProtocolPP::CIPHERSUITE, ProtocolPP::ENCTHENMAC,
ProtocolPP::IVEX, ProtocolPP::FRAMECTL, ProtocolPP::CTLEXT, ProtocolPP::PROTVER,
ProtocolPP::SUBTYPE, ProtocolPP::TDS, ProtocolPP::FDS, ProtocolPP::MFRAG,
ProtocolPP::RETRY, ProtocolPP::PWRMGMT, ProtocolPP::MDATA, ProtocolPP::WEP,
ProtocolPP::ORDER, ProtocolPP::KDFALG, ProtocolPP::ADDR1, ProtocolPP::ADDR2,
ProtocolPP::ADDR3, ProtocolPP::SEQCTL, ProtocolPP::ADDR4, ProtocolPP::QOSCTL,
ProtocolPP::HTCTL, ProtocolPP::EXTIV, ProtocolPP::KEYID, ProtocolPP::SPPCAP,
ProtocolPP::FCS, ProtocolPP::HT, ProtocolPP::EC, ProtocolPP::ESF,
ProtocolPP::CI, ProtocolPP::EKS, ProtocolPP::CID, ProtocolPP::FID,
ProtocolPP::EH, ProtocolPP::HCS, ProtocolPP::IKECNXT, ProtocolPP::EXCHG,
ProtocolPP::NXTPYLD, ProtocolPP::MAJOR_VERSION, ProtocolPP::MINOR_VERSION, ProtocolPP::MSGIDINIT,
ProtocolPP::MSGIDRESP, ProtocolPP::SPli, ProtocolPP::SPlr, ProtocolPP::Ni,
ProtocolPP::Nr, ProtocolPP::SKd, ProtocolPP::SKei, ProtocolPP::SKer,
ProtocolPP::SKsi, ProtocolPP::SKsr, ProtocolPP::INTEG, ProtocolPP::SKai,
ProtocolPP::SKar, ProtocolPP::PRF, ProtocolPP::PRFLEN, ProtocolPP::SKpi,
ProtocolPP::SKpr, ProtocolPP::DH, ProtocolPP::IKEAUTH, ProtocolPP::BITSIZE,
ProtocolPP::PRVKEY, ProtocolPP::PUBKEY, ProtocolPP::PRIME, ProtocolPP::SUBPRIME,
ProtocolPP::GENERATOR, ProtocolPP::GX, ProtocolPP::GY, ProtocolPP::CURVE }

• enum ProtocolPP::endian_t { ProtocolPP::BIG, ProtocolPP::LITTLE }

• enum ProtocolPP::platform_t { ProtocolPP::WASPPLAT, ProtocolPP::SECPLAT }

• enum ProtocolPP::pad_t {
    ProtocolPP::RANDOM, ProtocolPP::INCREMENT, ProtocolPP::ZERO, ProtocolPP::SIZE,
    ProtocolPP::UNKNWN }

• enum ProtocolPP::protocol_t {
    ProtocolPP::NO_ERROR, ProtocolPP::PROTOCOL, ProtocolPP::MACSEC, ProtocolPP::WIFI,
    ProtocolPP::WIGIG, ProtocolPP::WIMAX, ProtocolPP::LTE, ProtocolPP::RLC,
    ProtocolPP::TLS, ProtocolPP::SRTP, ProtocolPP::UDPP, ProtocolPP::TCPP,
    ProtocolPP::ICMPP, ProtocolPP::IP, ProtocolPP::IPSEC, ProtocolPP::XMLPARSER,
    ProtocolPP::WASP, ProtocolPP::RINGDRIVER, ProtocolPP::DIRECTDRIVER, ProtocolPP::DRIVER,
    ProtocolPP::LOOPBACK, ProtocolPP::ETHERNET, ProtocolPP::IKEV2, ProtocolPP::NOPROTO }

• enum ProtocolPP::direction_t {
    ProtocolPP::ENCAP, ProtocolPP::DECAP, ProtocolPP::ENC, ProtocolPP::DEC,
    ProtocolPP::DOWNLINK, ProtocolPP::UPLINK, ProtocolPP::NODIR }

• enum ProtocolPP::cipher_t {
    ProtocolPP::NULL_CIPHER, ProtocolPP::DES_CBC, ProtocolPP::DES40_CBC, ProtocolPP::TDES_CBC,
    ProtocolPP::AES_CBC, ProtocolPP::AES_CTR, ProtocolPP::AES_CCM, ProtocolPP::AES_GCM,
    ProtocolPP::CHACHA20, ProtocolPP::CHACHA20_POLY1305, ProtocolPP::CAMELLIA_CBC, ProtocolPP::
}

```

```

::CAMELLIA_CTR,
ProtocolPP::CAMELLIA_CCM, ProtocolPP::CAMELLIA_GCM, ProtocolPP::ARIA_CBC, ProtocolPP::ARIA_-
CTR,
ProtocolPP::ARIA_GCM, ProtocolPP::SEED_CBC, ProtocolPP::SEED_CTR, ProtocolPP::SEED_GCM,
ProtocolPP::SM4_CBC, ProtocolPP::SM4_CTR, ProtocolPP::SM4_GCM, ProtocolPP::SM4_CCM,
ProtocolPP::ARC4, ProtocolPP::SNOWE, ProtocolPP::ZUCE, ProtocolPP::NONE_ENC }

encryption ciphers

• enum ProtocolPP::auth_t {
ProtocolPP::NULL_AUTH, ProtocolPP::MD5, ProtocolPP::SHA1, ProtocolPP::SHA224,
ProtocolPP::SHA256, ProtocolPP::SHA384, ProtocolPP::SHA512, ProtocolPP::SHA3_224,
ProtocolPP::SHA3_256, ProtocolPP::SHA3_384, ProtocolPP::SHA3_512, ProtocolPP::AES_XCBC_MAC,
ProtocolPP::POLY1305, ProtocolPP::AES_CMAC, ProtocolPP::AES_GMAC, ProtocolPP::CRC32,
ProtocolPP::SM3, ProtocolPP::SNOWA, ProtocolPP::ZUCA, ProtocolPP::NONE_AUTH }

authentication types

• enum ProtocolPP::iana_t {
ProtocolPP::HOPOPT, ProtocolPP::ICMP, ProtocolPP::IGMP, ProtocolPP::GGP,
ProtocolPP::IPV4, ProtocolPP::ST, ProtocolPP::TCP, ProtocolPP::CBT,
ProtocolPP::EGP, ProtocolPP::IGP, ProtocolPP::BBN_RCC_MON, ProtocolPP::NVP_II,
ProtocolPP::PUP, ProtocolPP::ARGUS, ProtocolPP::EMCON, ProtocolPP::XNET,
ProtocolPP::CHAOS, ProtocolPP::UDP, ProtocolPP::MUX, ProtocolPP::DCN_MEAS,
ProtocolPP::HMP, ProtocolPP::PRM, ProtocolPP::XNS_IDP, ProtocolPP::TRUNK_1,
ProtocolPP::TRUNK_2, ProtocolPP::LEAF_1, ProtocolPP::LEAF_2, ProtocolPP::RDP,
ProtocolPP::IRTP, ProtocolPP::ISO_TP4, ProtocolPP::NETBLT, ProtocolPP::MFE_NSP,
ProtocolPP::MERIT_INP, ProtocolPP::DCCP, ProtocolPP::THREE_PC, ProtocolPP::IDPR,
ProtocolPP::XTP, ProtocolPP::DDP, ProtocolPP::IDPR_CMTP, ProtocolPP::TP_PLUS,
ProtocolPP::IL, ProtocolPP::IPV6, ProtocolPP::SDRP, ProtocolPP::IPV6_ROUTE,
ProtocolPP::IPV6_FRAG, ProtocolPP::IDRP, ProtocolPP::RSVP, ProtocolPP::GRE,
ProtocolPP::DSR, ProtocolPP::BNA, ProtocolPP::ESP, ProtocolPP::AH,
ProtocolPP::I_NLSP, ProtocolPP::SWIPE, ProtocolPP::NARP, ProtocolPP::MOBILE,
ProtocolPP::TLSP, ProtocolPP::SKIP, ProtocolPP::ICMPV6, ProtocolPP::IPV6_NONXT,
ProtocolPP::IPV6_OPTS, ProtocolPP::HOST_INT_PROT, ProtocolPP::CFTP, ProtocolPP::LOCAL_NET,
ProtocolPP::SAT_EXPAK, ProtocolPP::KRYPTOLAN, ProtocolPP::RVD, ProtocolPP::IPPC,
ProtocolPP::DFS, ProtocolPP::SAT_MON, ProtocolPP::VISA, ProtocolPP::IPCV,
ProtocolPP::CPNX, ProtocolPP::CPHB, ProtocolPP::WSN, ProtocolPP::PVP,
ProtocolPP::BR_SAT_MON, ProtocolPP::SUN_ND, ProtocolPP::WB_MON, ProtocolPP::WB_EXPAK,
ProtocolPP::ISO_IP, ProtocolPP::VMTP, ProtocolPP::SECURE_VMTP, ProtocolPP::VINES,
ProtocolPP::TTP, ProtocolPP::IPTM, ProtocolPP::NSFNET_IGP, ProtocolPP::DGP,
ProtocolPP::TCF, ProtocolPP::EIGRP, ProtocolPP::OSPFIGP, ProtocolPP::SPRITE_RPC,
ProtocolPP::LARP, ProtocolPP::MTP, ProtocolPP::AX_25, ProtocolPP::MICP,
ProtocolPP::SCC_SP, ProtocolPP::ETHERIP, ProtocolPP::IENCAP, ProtocolPP::PRIV_ENCRYPT,
ProtocolPP::GMTP, ProtocolPP::IFMP, ProtocolPP::PNNI, ProtocolPP::PIM,
ProtocolPP::ARIS, ProtocolPP::SCPS, ProtocolPP::QNX, ProtocolPP::A_N,
ProtocolPP::IPCOMP, ProtocolPP::SNP, ProtocolPP::COMPAQ_PEER, ProtocolPP::IPX_IN_IP,
ProtocolPP::VRRP, ProtocolPP::PGM, ProtocolPP::ZERO_HOP_PROT, ProtocolPP::L2TP,
ProtocolPP::DDX, ProtocolPP::IATP, ProtocolPP::STP, ProtocolPP::SRP,
ProtocolPP::UTI, ProtocolPP::SMP, ProtocolPP::SM, ProtocolPP::PTP,
ProtocolPP::ISIS_OVER_IPV4, ProtocolPP::FIRE, ProtocolPP::CRTP, ProtocolPP::CRUDP,
ProtocolPP::SSCOPMCE, ProtocolPP::IPLT, ProtocolPP::SPS, ProtocolPP::PIPE,
ProtocolPP::SCTP, ProtocolPP::FC, ProtocolPP::RSVP_E2E_IGNORE, ProtocolPP::MOBILITY_HEADER,
ProtocolPP::UDPLITE, ProtocolPP::MPLS_IN_IP, ProtocolPP::MANET, ProtocolPP::HIP,
ProtocolPP::SHIM6, ProtocolPP::WESP, ProtocolPP::ROHC, ProtocolPP::JUMBOGRAM =194 }

• enum ProtocolPP::err_t {
ProtocolPP::ERR_NONE, ProtocolPP::ERR_LATE, ProtocolPP::ERR_REPLY, ProtocolPP::ERR_ROLLO-
VER,
ProtocolPP::ERR_ROLLUNDER, ProtocolPP::ERR_PROGRAM, ProtocolPP::ERR_ICV, ProtocolPP::ERR-
_CRC,
ProtocolPP::ERR_READ_ENDOFILE, ProtocolPP::ERR_READ_FAILFILE, ProtocolPP::ERR_READ_BAD-

```

```

FILE, ProtocolPP::ERR_WRITE_FAILFILE,
ProtocolPP::ERR_WRITE_BADFILE, ProtocolPP::ERR_CHECKSUM, ProtocolPP::ERR_LISTEN, Protocol-
PP::ERR_ACKNUM,
ProtocolPP::ERR_CLOSED, ProtocolPP::ERR_BITS, ProtocolPP::ERR_TTL, ProtocolPP::ERR_JUMBOG-
RAM_FORMAT,
ProtocolPP::WARN_DUMMY, ProtocolPP::WARN_IPV6_ROUTE, ProtocolPP::WARN_ZERO_DATA,
ProtocolPP::ERR_ICMP_HDR_EXT_LEN,
ProtocolPP::ERR_ICMP_SEGMENTS_LEFT, ProtocolPP::ERR_MULTICAST_EXT_HDR, ProtocolPP::ER-
R_UNKNOWN_ROUTE_TYPE, ProtocolPP::ERR_CIPHER_KEY_SIZE,
ProtocolPP::ERR_AUTH_KEY_SIZE, ProtocolPP::ERR_IV_SIZE, ProtocolPP::ERR_SALT_SIZE, Protocol-
PP::ERR_ICV_SIZE,
ProtocolPP::ERR_UNKNOWN_NXTHDR, ProtocolPP::ERR_FORMAT_ERROR, ProtocolPP::WARN_INPU-
T_QUEUE_FULL, ProtocolPP::WARN_OUTPUT_QUEUE_EMPTY,
ProtocolPP::ERR_SA_NOT_FOUND, ProtocolPP::ERR_KEY_NOKEY, ProtocolPP::ERR_KEY_EXPIRED,
ProtocolPP::ERR_KEY_REVOKED,
ProtocolPP::ERR_KEY_ACCESS, ProtocolPP::ERR_KEY_NOT_SUPP, ProtocolPP::ERR_KEY_QUOTA,
ProtocolPP::ERR_KEY_INVALID,
ProtocolPP::ERR_KEY_REJECTED, ProtocolPP::ERR_KEY_NOMEM, ProtocolPP::ERR_KEY_INTR,
ProtocolPP::ERR_UNKNOWN }

• enum ProtocolPP::replay_t {
    ProtocolPP::NORMAL, ProtocolPP::SHIFT, ProtocolPP::WINDOW, ProtocolPP::REPLAY,
    ProtocolPP::LATE, ProtocolPP::ROLLOVER, ProtocolPP::ROLLUNDER }

• enum ProtocolPP::ipmode_t { ProtocolPP::TRANSPORT, ProtocolPP::TUNNEL }

• enum ProtocolPP::icmppmsg_t {
    ProtocolPP::ECHORPLYMSG, ProtocolPP::RSVD1, ProtocolPP::RSVD2, ProtocolPP::DESTUNRCH,
    ProtocolPP::SRCQNCH, ProtocolPP::RDIRMSG, ProtocolPP::ALTHOST, ProtocolPP::RSVD,
    ProtocolPP::ECHORQST, ProtocolPP::RTRADVERT, ProtocolPP::RTRSOLCIT, ProtocolPP::TIMEXCEED,
    ProtocolPP::BADIPHDR, ProtocolPP::TIMESTMP, ProtocolPP::TIMERPLY, ProtocolPP::INFORQST,
    ProtocolPP::INFORPLY, ProtocolPP::ADMSKRQST, ProtocolPP::ADMSKRPLY, ProtocolPP::TRACERTE,
    ProtocolPP::NODEST, ProtocolPP::PKTBIG, ProtocolPP::TIMEOUT, ProtocolPP::PARAM,
    ProtocolPP::PRVTE1, ProtocolPP::PRVTE2, ProtocolPP::RSVDE, ProtocolPP::ECHORQSTV6,
    ProtocolPP::ECHORPLYV6, ProtocolPP::PRVTI1, ProtocolPP::PRVTI2, ProtocolPP::RSVDI }

• enum ProtocolPP::icmpcode_t {
    ProtocolPP::ECHORPLY, ProtocolPP::NONETWRK, ProtocolPP::NOHOST, ProtocolPP::NOPROT,
    ProtocolPP::NOPORT, ProtocolPP::FRAGRQD, ProtocolPP::RTEFAIL, ProtocolPP::DSTUNKWN,
    ProtocolPP::DSTHSTUNKWN, ProtocolPP::SRCHSTISOLT, ProtocolPP::NETWKPROHIB, ProtocolPP::H-
    OSTPROHIB,
    ProtocolPP::NETWKNOTOS, ProtocolPP::HOSTNOTOS, ProtocolPP::COMMPROHIB, ProtocolPP::HOST-
    VILATE,
    ProtocolPP::CUTOFF, ProtocolPP::REDIRNETWK, ProtocolPP::REDIRHOST, ProtocolPP::REDIRTSN,
    ProtocolPP::REDIRTOSH, ProtocolPP::TTLEXPIRE, ProtocolPP::FRAGEXPRESS, ProtocolPP::PTRINDERR,
    ProtocolPP::OPTERR, ProtocolPP::BADLENGTH, ProtocolPP::NOROUTE, ProtocolPP::COMMPROHIBV6,
    ProtocolPP::BYNDSRCADDR, ProtocolPP::ADDRUNREACH, ProtocolPP::PORTUNREACH, ProtocolPP::-
    SRCADDRPLCY,
    ProtocolPP::REJECTDST, ProtocolPP::HOPLMTEXCD, ProtocolPP::FRAGTIME, ProtocolPP::ERRHDRFI-
    ELD,
    ProtocolPP::NXTHDRERR, ProtocolPP::IPV6OPTERR, ProtocolPP::SEQNUMRST }

• enum ProtocolPP::tlsver_t {
    ProtocolPP::SSL30 = 0x0300, ProtocolPP::TLS10 = 0x0301, ProtocolPP::TLS11 = 0x0302, ProtocolPP::TL-
    S12 = 0x0303,
    ProtocolPP::DTLS = 0xFFFF }

• enum ProtocolPP::tlstype_t {
    ProtocolPP::CHG_CIPHER_SPEC = 20, ProtocolPP::ALERT = 21, ProtocolPP::HANDSHAKE = 22,
    ProtocolPP::APPLICATION = 23,
    ProtocolPP::HEARTBEAT = 24 }

• enum ProtocolPP::tlsvl_t { ProtocolPP::TLSVL0, ProtocolPP::WARNING, ProtocolPP::FATAL }

• enum ProtocolPP::tls_handshake_t {
    ProtocolPP::HELLO_REQUEST = 0, ProtocolPP::CLIENT_HELLO = 1, ProtocolPP::SERVER_HELLO = 2,
    ProtocolPP::CERTIFICATE = 3, ProtocolPP::CERTIFICATE_VERIFY = 4, ProtocolPP::KEY_EXCHANGE = 5,
    ProtocolPP::CERTIFICATE_REQUEST = 6, ProtocolPP::CERTIFICATE_VERIFY = 7, ProtocolPP::KEY_EX-
    CHANGE = 8, ProtocolPP::SIGNATURE = 9, ProtocolPP::SIGNATURE_ALGORITHM = 10, ProtocolPP::ENC-
    RYPTION = 11, ProtocolPP::ENCRYPTION_ALGORITHM = 12, ProtocolPP::COMPRESS = 13, ProtocolPP::COMP-
    RESS_ALGORITHM = 14, ProtocolPP::EXTENSIONS = 15, ProtocolPP::EXTENSION = 16, ProtocolPP::H-
    ANDSHAKE = 17, ProtocolPP::SESSION_TICKET = 18, ProtocolPP::SESSION_TICKET_ALGORITHM = 19 }

```

```

ProtocolPP::NEW_SESSION_TICKET =4,
ProtocolPP::CERTIFICATE =11, ProtocolPP::SERVER_KEY_EXCHANGE =12, ProtocolPP::CERTIFICATE_REQUEST =13, ProtocolPP::SERVER_HELLO_DONE =14,
ProtocolPP::CERTIFICATE_VERIFY =15, ProtocolPP::CLIENT_KEY_EXCHANGE =16, ProtocolPP::FINISHED =20 }

• enum ProtocolPP::tls_error_t {
ProtocolPP::CLOSE = 0, ProtocolPP::MESSAGE_FATAL =10, ProtocolPP::BAD_RECORD_MAC =20,
ProtocolPP::DECRYPT_FATAL =21,
ProtocolPP::RECORD_OVERFLOW =22, ProtocolPP::DECOMPRESS_FAIL =30, ProtocolPP::HANDSHAKE_FAIL =40, ProtocolPP::NO_CERTIFICATE =41,
ProtocolPP::BAD_CERTIFICATE =42, ProtocolPP::UNSUPPORTED_CERTIFICATE =43, ProtocolPP::CERTIFICATE_REVOKED =44, ProtocolPP::CERTIFICATE_EXPIRED =45,
ProtocolPP::CERTIFICATE_UNKNOWN =46, ProtocolPP::ILLEGAL_PARAMETER =47, ProtocolPP::UNKNOWN_CA =48, ProtocolPP::ACCESS_DENIED =49,
ProtocolPP::DECODE_ERROR =50, ProtocolPP::DECRYPT_ERROR =51, ProtocolPP::EXPORT_RESTRICTION =60, ProtocolPP::PROTOCOL_VERSION =70,
ProtocolPP::INSUFFICIENT_SECURITY =71, ProtocolPP::INTERNAL_ERROR =80, ProtocolPP::USER_CANCELLED =90, ProtocolPP::NO_RENEGOTIATION =100,
ProtocolPP::UNSUPPORTED_EXTENSION =110, ProtocolPP::CERTIFICATE_UNOBTAINABLE =111,
ProtocolPP::UNRECOGNIZED_NAME =112, ProtocolPP::BAD_CERTIFICATE_STATUS_RESPONSE =113,
ProtocolPP::BAD_CERTIFICATE_HASH_VALUE =114, ProtocolPP::UNKNOWN_PSK_IDENTITY =115,
ProtocolPP::NO_APPLICATION_PROTOCOL =120 }

• enum ProtocolPP::srtpcipher_t {
ProtocolPP::NULL_SRTP, ProtocolPP::AES_CTR_SHA1, ProtocolPP::AEAD_AES_128_GCM, ProtocolPP::AEAD_AES_256_GCM,
ProtocolPP::AEAD_AES_128_GCM_8, ProtocolPP::AEAD_AES_256_GCM_8, ProtocolPP::AEAD_AES_128_GCM_12, ProtocolPP::AEAD_AES_256_GCM_12,
ProtocolPP::AEAD_AES_128_CCM, ProtocolPP::AEAD_AES_256_CCM, ProtocolPP::AEAD_AES_128_CCM_8, ProtocolPP::AEAD_AES_256_CCM_8,
ProtocolPP::AEAD_AES_128_CCM_12, ProtocolPP::AEAD_AES_256_CCM_12, ProtocolPP::ARIA_128_CTR_HMAC_SHA1_80, ProtocolPP::ARIA_128_CTR_HMAC_SHA1_32,
ProtocolPP::ARIA_192_CTR_HMAC_SHA1_80, ProtocolPP::ARIA_192_CTR_HMAC_SHA1_32, ProtocolPP::ARIA_256_CTR_HMAC_SHA1_80, ProtocolPP::ARIA_256_CTR_HMAC_SHA1_32,
ProtocolPP::AEAD_ARIA_128_GCM, ProtocolPP::AEAD_ARIA_256_GCM }

• enum ProtocolPP::tls_ciphersuite_t {
ProtocolPP::TLS_NULL_WITH_NULL_NULL =0x0000, ProtocolPP::TLS_RSA_WITH_NULL_MD5 =0x0001,
ProtocolPP::TLS_RSA_WITH_NULL_SHA =0x0002, ProtocolPP::TLS_RSA_EXPORT_WITH_RC4_40_MD5 =0x0003,
ProtocolPP::TLS_RSA_WITH_RC4_128_MD5 =0x0004, ProtocolPP::TLS_RSA_WITH_RC4_128_SHA =0x0005, ProtocolPP::TLS_RSA_EXPORT_WITH_DES40_CBC_SHA =0x0008, ProtocolPP::TLS_RSA_WITH_DES_CBC_SHA =0x0009,
ProtocolPP::TLS_RSA_WITH_3DES_EDE_CBC_SHA =0x000A, ProtocolPP::TLS_DH_DSS_EXPORT_WITH_DES40_CBC_SHA =0x000B, ProtocolPP::TLS_DH_DSS_WITH_DES_CBC_SHA =0x000C, ProtocolPP::TLS_DH_DSS_WITH_3DES_EDE_CBC_SHA =0x000D,
ProtocolPP::TLS_DH_RSA_EXPORT_WITH_DES40_CBC_SHA =0x000E, ProtocolPP::TLS_DH_RSA_WITH_DES_CBC_SHA =0x000F, ProtocolPP::TLS_DH_RSA_WITH_3DES_EDE_CBC_SHA =0x0010,
ProtocolPP::TLS_DHE_DSS_EXPORT_WITH_DES40_CBC_SHA =0x0011, ProtocolPP::TLS_DHE_DSS_WITH_DES_CBC_SHA =0x0012, ProtocolPP::TLS_DHE_DSS_WITH_3DES_EDE_CBC_SHA =0x0013, ProtocolPP::TLS_DHE_RSA_EXPORT_WITH_DES40_CBC_SHA =0x0014,
ProtocolPP::TLS_DHE_RSA_WITH_DES_CBC_SHA =0x0015, ProtocolPP::TLS_DHE_RSA_WITH_3DES_EDE_CBC_SHA =0x0016, ProtocolPP::TLS_DH_ANON_EXPORT_WITH_RC4_40_MD5 =0x0017, ProtocolPP::TLS_DH_ANON_WITH_RC4_128_MD5 =0x0018,
ProtocolPP::TLS_DH_ANON_EXPORT_WITH_DES40_CBC_SHA =0x0019, ProtocolPP::TLS_DH_ANON_WITH_DES_CBC_SHA =0x001A, ProtocolPP::TLS_DH_ANON_WITH_3DES_EDE_CBC_SHA =0x001B, ProtocolPP::TLS_KRB5_WITH_DES_CBC_SHA =0x001E, ProtocolPP::TLS_KRB5_WITH_3DES_EDE_CBC_SHA =0x001F,
ProtocolPP::TLS_KRB5_WITH_RC4_128_SHA =0x0020, ProtocolPP::TLS_KRB5_WITH_DES_CBC_MD5
}

```

=0x0022, ProtocolPP::TLS_KRB5_WITH_3DES_EDE_CBC_MD5 =0x0023, ProtocolPP::TLS_KRB5_WITH_RC4_128_MD5 =0x0024,
ProtocolPP::TLS_KRB5_EXPORT_WITH_DES_CBC_40_SHA =0x0026, ProtocolPP::TLS_KRB5_EXPORT_WITH_RC4_40_SHA =0x0028, ProtocolPP::TLS_KRB5_EXPORT_WITH_DES_CBC_40_MD5 =0x0029,
ProtocolPP::TLS_KRB5_EXPORT_WITH_RC4_40_MD5 =0x002B,
ProtocolPP::TLS_PSK_WITH_NULL_SHA =0x002C, ProtocolPP::TLS_DHE_PSK_WITH_NULL_SHA =0x002D, ProtocolPP::TLS_RSA_PSK_WITH_NULL_SHA =0x002E, ProtocolPP::TLS_RSA_WITH_AES_128_CBC_SHA =0x002F,
ProtocolPP::TLS_DH_DSS_WITH_AES_128_CBC_SHA =0x0030, ProtocolPP::TLS_DH_RSA_WITH_AES_128_CBC_SHA =0x0031, ProtocolPP::TLS_DHE_DSS_WITH_AES_128_CBC_SHA =0x0032, ProtocolPP::TLS_DHE_RSA_WITH_AES_128_CBC_SHA =0x0033,
ProtocolPP::TLS_DH_ANON_WITH_AES_128_CBC_SHA =0x0034, ProtocolPP::TLS_RSA_WITH_AES_256_CBC_SHA =0x0035, ProtocolPP::TLS_DH_DSS_WITH_AES_256_CBC_SHA =0x0036, ProtocolPP::TLS_DH_RSA_WITH_AES_256_CBC_SHA =0x0037,
ProtocolPP::TLS_DHE_DSS_WITH_AES_256_CBC_SHA =0x0038, ProtocolPP::TLS_DHE_RSA_WITH_AES_256_CBC_SHA =0x0039, ProtocolPP::TLS_DH_ANON_WITH_AES_256_CBC_SHA =0x003A, ProtocolPP::TLS_RSA_WITH_NULL_SHA256 =0x003B,
ProtocolPP::TLS_RSA_WITH_AES_128_CBC_SHA256 =0x003C, ProtocolPP::TLS_RSA_WITH_AES_256_CBC_SHA256 =0x003D, ProtocolPP::TLS_DH_DSS_WITH_AES_128_CBC_SHA256 =0x003E, ProtocolPP::TLS_DH_RSA_WITH_AES_128_CBC_SHA256 =0x003F,
ProtocolPP::TLS_DHE_DSS_WITH_AES_128_CBC_SHA256 =0x0040, ProtocolPP::TLS_RSA_WITH_CAMELLIA_128_CBC_SHA =0x0041, ProtocolPP::TLS_DH_DSS_WITH_CAMELLIA_128_CBC_SHA =0x0042, ProtocolPP::TLS_DH_RSA_WITH_CAMELLIA_128_CBC_SHA =0x0043,
ProtocolPP::TLS_DHE_DSS_WITH_CAMELLIA_128_CBC_SHA =0x0044, ProtocolPP::TLS_DHE_RSA_WITH_CAMELLIA_128_CBC_SHA =0x0045, ProtocolPP::TLS_DH_ANON_WITH_CAMELLIA_128_CBC_SHA =0x0046, ProtocolPP::TLS_DHE_RSA_WITH_AES_128_CBC_SHA256 =0x0067,
ProtocolPP::TLS_DH_DSS_WITH_AES_256_CBC_SHA256 =0x0068, ProtocolPP::TLS_DH_RSA_WITH_AES_256_CBC_SHA256 =0x0069, ProtocolPP::TLS_DHE_DSS_WITH_AES_256_CBC_SHA256 =0x006A, ProtocolPP::TLS_DHE_RSA_WITH_AES_256_CBC_SHA256 =0x006B,
ProtocolPP::TLS_DH_ANON_WITH_AES_128_CBC_SHA256 =0x006C, ProtocolPP::TLS_DH_ANON_WITH_AES_256_CBC_SHA256 =0x006D, ProtocolPP::TLS_RSA_WITH_CAMELLIA_256_CBC_SHA =0x0084, ProtocolPP::TLS_DH_DSS_WITH_CAMELLIA_256_CBC_SHA =0x0085,
ProtocolPP::TLS_DH_RSA_WITH_CAMELLIA_256_CBC_SHA =0x0086, ProtocolPP::TLS_DHE_DSS_WITH_CAMELLIA_256_CBC_SHA =0x0087, ProtocolPP::TLS_DHE_RSA_WITH_CAMELLIA_256_CBC_SHA =0x0088, ProtocolPP::TLS_DH_ANON_WITH_CAMELLIA_256_CBC_SHA =0x0089,
ProtocolPP::TLS_PSK_WITH_RC4_128_SHA =0x008A, ProtocolPP::TLS_PSK_WITH_3DES_EDE_CBC_SHA =0x008B, ProtocolPP::TLS_PSK_WITH_AES_128_CBC_SHA =0x008C, ProtocolPP::TLS_PSK_WITH_AES_256_CBC_SHA =0x008D,
ProtocolPP::TLS_DHE_PSK_WITH_RC4_128_SHA =0x008E, ProtocolPP::TLS_DHE_PSK_WITH_3DES_EDE_CBC_SHA =0x008F, ProtocolPP::TLS_DHE_PSK_WITH_AES_128_CBC_SHA =0x0090, ProtocolPP::TLS_DHE_PSK_WITH_AES_256_CBC_SHA =0x0091,
ProtocolPP::TLS_RSA_PSK_WITH_RC4_128_SHA =0x0092, ProtocolPP::TLS_RSA_PSK_WITH_3DES_EDE_CBC_SHA =0x0093, ProtocolPP::TLS_RSA_PSK_WITH_AES_128_CBC_SHA =0x0094, ProtocolPP::TLS_RSA_PSK_WITH_AES_256_CBC_SHA =0x0095,
ProtocolPP::TLS_RSA_WITH_SEED_CBC_SHA =0x0096, ProtocolPP::TLS_DH_DSS_WITH_SEED_CBC_SHA =0x0097, ProtocolPP::TLS_DH_RSA_WITH_SEED_CBC_SHA =0x0098, ProtocolPP::TLS_DHE_DSS_WITH_SEED_CBC_SHA =0x0099,
ProtocolPP::TLS_DHE_RSA_WITH_SEED_CBC_SHA =0x009A, ProtocolPP::TLS_DH_ANON_WITH_SEED_CBC_SHA =0x009B, ProtocolPP::TLS_RSA_WITH_AES_128_GCM_SHA256 =0x009C, ProtocolPP::TLS_RSA_WITH_AES_256_GCM_SHA384 =0x009D,
ProtocolPP::TLS_DHE_RSA_WITH_AES_128_GCM_SHA256 =0x009E, ProtocolPP::TLS_DHE_RSA_WITH_AES_256_GCM_SHA384 =0x009F, ProtocolPP::TLS_DH_RSA_WITH_AES_128_GCM_SHA256 =0x00A0, ProtocolPP::TLS_DH_RSA_WITH_AES_256_GCM_SHA384 =0x00A1,
ProtocolPP::TLS_DHE_DSS_WITH_AES_128_GCM_SHA256 =0x00A2, ProtocolPP::TLS_DHE_DSS_WITH_AES_256_GCM_SHA384 =0x00A3, ProtocolPP::TLS_DHE_DSS_WITH_AES_128_GCM_SHA256 =0x00A4, ProtocolPP::TLS_DHE_DSS_WITH_AES_256_GCM_SHA384 =0x00A5,
ProtocolPP::TLS_DH_ANON_WITH_AES_128_GCM_SHA256 =0x00A6, ProtocolPP::TLS_DH_ANON_WITH_AES_256_GCM_SHA384 =0x00A7, ProtocolPP::TLS_PSK_WITH_AES_128_GCM_SHA256 =0x00A8,

ProtocolIPP::TLS_PSK_WITH_AES_256_GCM_SHA384 =0x00A9,
 ProtocolIPP::TLS_DHE_PSK_WITH_AES_128_GCM_SHA256 =0x00AA, ProtocolIPP::TLS_DHE_PSK_WITH_AES_256_GCM_SHA384 =0x00AB, ProtocolIPP::TLS_RSA_PSK_WITH_AES_128_GCM_SHA256 =0x00AC, ProtocolIPP::TLS_RSA_PSK_WITH_AES_256_GCM_SHA384 =0x00AD,
 ProtocolIPP::TLS_PSK_WITH_AES_128_CBC_SHA256 =0x00AE, ProtocolIPP::TLS_PSK_WITH_AES_256_CBC_SHA384 =0x00AF, ProtocolIPP::TLS_PSK_WITH_NULL_SHA256 =0x00B0, ProtocolIPP::TLS_PSK_WITH_NULL_SHA384 =0x00B1,
 ProtocolIPP::TLS_DHE_PSK_WITH_AES_128_CBC_SHA256 =0x00B2, ProtocolIPP::TLS_DHE_PSK_WITH_NULL_SHA256 =0x00B4, ProtocolIPP::TLS_DHE_PSK_WITH_NULL_SHA384 =0x00B5,
 ProtocolIPP::TLS_RSA_PSK_WITH_AES_128_CBC_SHA256 =0x00B6, ProtocolIPP::TLS_RSA_PSK_WITH_AES_256_CBC_SHA384 =0x00B7, ProtocolIPP::TLS_RSA_PSK_WITH_NULL_SHA256 =0x00B8, ProtocolIPP::TLS_RSA_PSK_WITH_NULL_SHA384 =0x00B9,
 ProtocolIPP::TLS_RSA_WITH_CAMELLIA_128_CBC_SHA256 =0x00BA, ProtocolIPP::TLS_DH_DSS_WITH_CAMELLIA_128_CBC_SHA256 =0x00BB, ProtocolIPP::TLS_DH_RSA_WITH_CAMELLIA_128_CBC_SHA256 =0x00BC, ProtocolIPP::TLS_DHE_DSS_WITH_CAMELLIA_128_CBC_SHA256 =0x00BD,
 ProtocolIPP::TLS_DHE_RSA_WITH_CAMELLIA_128_CBC_SHA256 =0x00BE, ProtocolIPP::TLS_DH_ANON_WITH_CAMELLIA_128_CBC_SHA256 =0x00BF, ProtocolIPP::TLS_RSA_WITH_CAMELLIA_256_CBC_SHA256 =0x00C0, ProtocolIPP::TLS_DH_DSS_WITH_CAMELLIA_256_CBC_SHA256 =0x00C1,
 ProtocolIPP::TLS_DH_RSA_WITH_CAMELLIA_256_CBC_SHA256 =0x00C2, ProtocolIPP::TLS_DHE_DSS_WITH_CAMELLIA_256_CBC_SHA256 =0x00C3, ProtocolIPP::TLS_DHE_RSA_WITH_CAMELLIA_256_CBC_SHA256 =0x00C4, ProtocolIPP::TLS_DH_ANON_WITH_CAMELLIA_256_CBC_SHA256 =0x00C5,
 ProtocolIPP::TLS_ECDH_ECDSA_WITH_NULL_SHA =0xC001, ProtocolIPP::TLS_ECDH_ECDSA_WITH_3DES_EDE_CBC_SHA =0xC003, ProtocolIPP::TLS_ECDH_ECDSA_WITH_AES_128_CBC_SHA =0xC004,
 ProtocolIPP::TLS_ECDH_ECDSA_WITH_AES_256_CBC_SHA =0xC005, ProtocolIPP::TLS_ECDHE_ECDSA_WITH_NULL_SHA =0xC006, ProtocolIPP::TLS_ECDHE_ECDSA_WITH_RC4_128_SHA =0xC007, ProtocolIPP::TLS_ECDHE_ECDSA_WITH_3DES_EDE_CBC_SHA =0xC008,
 ProtocolIPP::TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA =0xC009, ProtocolIPP::TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA =0xC00A, ProtocolIPP::TLS_ECDH_RSA_WITH_NULL_SHA =0xC00B, ProtocolIPP::TLS_ECDH_RSA_WITH_RC4_128_SHA =0xC00C,
 ProtocolIPP::TLS_ECDH_RSA_WITH_3DES_EDE_CBC_SHA =0xC00D, ProtocolIPP::TLS_ECDH_RSA_WITH_AES_128_CBC_SHA =0xC00F, ProtocolIPP::TLS_ECDHE_RSA_WITH_NULL_SHA =0xC010,
 ProtocolIPP::TLS_ECDHE_RSA_WITH_RC4_128_SHA =0xC011, ProtocolIPP::TLS_ECDHE_RSA_WITH_3DES_EDE_CBC_SHA =0xC012, ProtocolIPP::TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA =0xC013, ProtocolIPP::TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA =0xC014,
 ProtocolIPP::TLS_ECDH_ANON_WITH_NULL_SHA =0xC015, ProtocolIPP::TLS_ECDH_ANON_WITH_RC4_128_SHA =0xC016, ProtocolIPP::TLS_ECDH_ANON_WITH_3DES_EDE_CBC_SHA =0xC017, ProtocolIPP::TLS_ECDH_ANON_WITH_AES_128_CBC_SHA =0xC018,
 ProtocolIPP::TLS_ECDH_ANON_WITH_AES_256_CBC_SHA =0xC019, ProtocolIPP::TLS_SRPHASHA_WITH_3DES_EDE_CBC_SHA =0xC01A, ProtocolIPP::TLS_SRPHASHA_RSA_WITH_3DES_EDE_CBC_SHA =0xC01B, ProtocolIPP::TLS_SRPHASHA_DSS_WITH_3DES_EDE_CBC_SHA =0xC01C,
 ProtocolIPP::TLS_SRPHASHA_WITH_AES_128_CBC_SHA =0xC01D, ProtocolIPP::TLS_SRPHASHA_RSA_WITH_AES_128_CBC_SHA =0xC01E, ProtocolIPP::TLS_SRPHASHA_DSS_WITH_AES_128_CBC_SHA =0xC01F, ProtocolIPP::TLS_SRPHASHA_WITH_AES_256_CBC_SHA =0xC020,
 ProtocolIPP::TLS_SRPHASHA_RSA_WITH_AES_256_CBC_SHA =0xC021, ProtocolIPP::TLS_SRPHASHA_DSS_WITH_AES_256_CBC_SHA =0xC022, ProtocolIPP::TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256 =0xC023, ProtocolIPP::TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384 =0xC024,
 ProtocolIPP::TLS_ECDH_ECDSA_WITH_AES_128_CBC_SHA256 =0xC025, ProtocolIPP::TLS_ECDH_ECDSA_WITH_AES_256_CBC_SHA384 =0xC026, ProtocolIPP::TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256 =0xC027, ProtocolIPP::TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384 =0xC028,
 ProtocolIPP::TLS_ECDH_RSA_WITH_AES_128_CBC_SHA256 =0xC029, ProtocolIPP::TLS_ECDH_RSA_WITH_AES_256_CBC_SHA384 =0xC02A, ProtocolIPP::TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256 =0xC02B, ProtocolIPP::TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384 =0xC02C,
 ProtocolIPP::TLS_ECDH_ECDSA_WITH_AES_128_GCM_SHA256 =0xC02D, ProtocolIPP::TLS_ECDH_ECDSA_WITH_AES_256_GCM_SHA384 =0xC02E, ProtocolIPP::TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA384

```

M_SHA256 =0xC02F, ProtocolPP::TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384 =0xC030,
ProtocolPP::TLS_ECDH_RSA_WITH_AES_128_GCM_SHA256 =0xC031, ProtocolPP::TLS_ECDH_RSA_
_WITH_AES_256_GCM_SHA384 =0xC032, ProtocolPP::TLS_ECDHE_PSK_WITH_RC4_128_SHA =0x-
C033, ProtocolPP::TLS_ECDHE_PSK_WITH_3DES_EDE_CBC_SHA =0xC034,
ProtocolPP::TLS_ECDHE_PSK_WITH_AES_128_CBC_SHA =0xC035, ProtocolPP::TLS_ECDHE_PSK-
_WITH_AES_256_CBC_SHA =0xC036, ProtocolPP::TLS_ECDHE_PSK_WITH_AES_128_CBC_SHA256
=0xC037, ProtocolPP::TLS_ECDHE_PSK_WITH_AES_256_CBC_SHA384 =0xC038,
ProtocolPP::TLS_ECDHE_PSK_WITH_NULL_SHA =0xC039, ProtocolPP::TLS_ECDHE_PSK_WITH_NUL-
L_SHA256 =0xC03A, ProtocolPP::TLS_ECDHE_PSK_WITH_NULL_SHA384 =0xC03B, ProtocolPP::TLS_-
RSA_WITH_ARIA_128_CBC_SHA256 =0xC03C,
ProtocolPP::TLS_RSA_WITH_ARIA_256_CBC_SHA384 =0xC03D, ProtocolPP::TLS_DH_DSS_WITH_AR-
IA_128_CBC_SHA256 =0xC03E, ProtocolPP::TLS_DH_DSS_WITH_ARIA_256_CBC_SHA384 =0xC03F,
ProtocolPP::TLS_DH_RSA_WITH_ARIA_128_CBC_SHA256 =0xC040,
ProtocolPP::TLS_DH_RSA_WITH_ARIA_256_CBC_SHA384 =0xC041, ProtocolPP::TLS_DHE_DSS_W-
ITH_ARIA_128_CBC_SHA256 =0xC042, ProtocolPP::TLS_DHE_DSS_WITH_ARIA_256_CBC_SHA384
=0xC043, ProtocolPP::TLS_DHE_RSA_WITH_ARIA_128_CBC_SHA256 =0xC044,
ProtocolPP::TLS_DHE_RSA_WITH_ARIA_256_CBC_SHA384 =0xC045, ProtocolPP::TLS_DH_ANON_W-
ITH_ARIA_128_CBC_SHA256 =0xC046, ProtocolPP::TLS_DH_ANON_WITH_ARIA_256_CBC_SHA384
=0xC047, ProtocolPP::TLS_ECDHE_ECDSA_WITH_ARIA_128_CBC_SHA256 =0xC048,
ProtocolPP::TLS_ECDHE_ECDSA_WITH_ARIA_256_CBC_SHA384 =0xC049, ProtocolPP::TLS_ECDH_E-
CDSA_WITH_ARIA_128_CBC_SHA256 =0xC04A, ProtocolPP::TLS_ECDH_ECDSA_WITH_ARIA_256_C-
BC_SHA384 =0xC04B, ProtocolPP::TLS_ECDHE_RSA_WITH_ARIA_128_CBC_SHA256 =0xC04C,
ProtocolPP::TLS_ECDHE_RSA_WITH_ARIA_256_CBC_SHA384 =0xC04D, ProtocolPP::TLS_ECDH_RS-
A_WITH_ARIA_128_CBC_SHA256 =0xC04E, ProtocolPP::TLS_ECDH_RSA_WITH_ARIA_256_CBC_SH-
A384 =0xC04F, ProtocolPP::TLS_RSA_WITH_ARIA_128_GCM_SHA256 =0xC050,
ProtocolPP::TLS_RSA_WITH_ARIA_256_GCM_SHA384 =0xC051, ProtocolPP::TLS_DHE_RSA_WITH_-
ARIA_128_GCM_SHA256 =0xC052, ProtocolPP::TLS_DHE_RSA_WITH_ARIA_256_GCM_SHA384 =0x-
C053, ProtocolPP::TLS_DH_RSA_WITH_ARIA_128_GCM_SHA256 =0xC054,
ProtocolPP::TLS_DH_RSA_WITH_ARIA_256_GCM_SHA384 =0xC055, ProtocolPP::TLS_DHE_DSS_W-
ITH_ARIA_128_GCM_SHA256 =0xC056, ProtocolPP::TLS_DHE_DSS_WITH_ARIA_256_GCM_SHA384
=0xC057, ProtocolPP::TLS_DH_DSS_WITH_ARIA_128_GCM_SHA256 =0xC058,
ProtocolPP::TLS_DH_DSS_WITH_ARIA_256_GCM_SHA384 =0xC059, ProtocolPP::TLS_DH_ANON_W-
ITH_ARIA_128_GCM_SHA256 =0xC05A, ProtocolPP::TLS_DH_ANON_WITH_ARIA_256_GCM_SHA384
=0xC05B, ProtocolPP::TLS_ECDHE_ECDSA_WITH_ARIA_128_GCM_SHA256 =0xC05C,
ProtocolPP::TLS_ECDHE_ECDSA_WITH_ARIA_256_GCM_SHA384 =0xC05D, ProtocolPP::TLS_ECDH_-
ECDSA_WITH_ARIA_128_GCM_SHA256 =0xC05E, ProtocolPP::TLS_ECDH_ECDSA_WITH_ARIA_256_-
GCM_SHA384 =0xC05F, ProtocolPP::TLS_ECDHE_RSA_WITH_ARIA_128_GCM_SHA256 =0xC060,
ProtocolPP::TLS_ECDHE_RSA_WITH_ARIA_256_GCM_SHA384 =0xC061, ProtocolPP::TLS_ECDH_RS-
A_WITH_ARIA_128_GCM_SHA256 =0xC062, ProtocolPP::TLS_ECDH_RSA_WITH_ARIA_256_GCM_SH-
A384 =0xC063, ProtocolPP::TLS_PSK_WITH_ARIA_128_CBC_SHA256 =0xC064,
ProtocolPP::TLS_PSK_WITH_ARIA_256_CBC_SHA384 =0xC065, ProtocolPP::TLS_DHE_PSK_WITH_-
ARIA_128_CBC_SHA256 =0xC066, ProtocolPP::TLS_DHE_PSK_WITH_ARIA_256_CBC_SHA384 =0x-
C067, ProtocolPP::TLS_RSA_PSK_WITH_ARIA_128_CBC_SHA256 =0xC068,
ProtocolPP::TLS_RSA_PSK_WITH_ARIA_256_CBC_SHA384 =0xC069, ProtocolPP::TLS_PSK_WITH_-
ARIA_128_GCM_SHA256 =0xC06A, ProtocolPP::TLS_PSK_WITH_ARIA_256_GCM_SHA384 =0xC06B,
ProtocolPP::TLS_DHE_PSK_WITH_ARIA_128_GCM_SHA256 =0xC06C,
ProtocolPP::TLS_DHE_PSK_WITH_ARIA_256_GCM_SHA384 =0xC06D, ProtocolPP::TLS_RSA_PSK_W-
ITH_ARIA_128_GCM_SHA256 =0xC06E, ProtocolPP::TLS_RSA_PSK_WITH_ARIA_256_GCM_SHA384
=0xC06F, ProtocolPP::TLS_ECDHE_PSK_WITH_ARIA_128_CBC_SHA256 =0xC070,
ProtocolPP::TLS_ECDHE_PSK_WITH_ARIA_256_CBC_SHA384 =0xC071, ProtocolPP::TLS_ECDHE_E-
CDSA_WITH_CAMELLIA_128_CBC_SHA256 =0xC072, ProtocolPP::TLS_ECDHE_ECDSA_WITH_CAM-
ELLIA_256_CBC_SHA384 =0xC073, ProtocolPP::TLS_ECDH_ECDSA_WITH_CAMELLIA_128_CBC_SH-
A256 =0xC074,
ProtocolPP::TLS_ECDH_ECDSA_WITH_CAMELLIA_256_CBC_SHA384 =0xC075, ProtocolPP::TLS_ECD-
HE_RSA_WITH_CAMELLIA_128_CBC_SHA256 =0xC076, ProtocolPP::TLS_ECDHE_RSA_WITH_CAME-
LLIA_256_CBC_SHA384 =0xC077, ProtocolPP::TLS_ECDH_RSA_WITH_CAMELLIA_128_CBC_SHA256
=0xC078,
ProtocolPP::TLS_ECDH_RSA_WITH_CAMELLIA_256_CBC_SHA384 =0xC079, ProtocolPP::TLS_RSA_-

```

```

WITH_CAMELLIA_128_GCM_SHA256 =0xC07A, ProtocolPP::TLS_RSA_WITH_CAMELLIA_256_GCM_S-
HA384 =0xC07B, ProtocolPP::TLS_DHE_RSA_WITH_CAMELLIA_128_GCM_SHA256 =0xC07C,
ProtocolPP::TLS_DHE_RSA_WITH_CAMELLIA_256_GCM_SHA384 =0xC07D, ProtocolPP::TLS_DH_RS-
A_WITH_CAMELLIA_128_GCM_SHA256 =0xC07E, ProtocolPP::TLS_DH_RSA_WITH_CAMELLIA_256_-
GCM_SHA384 =0xC07F, ProtocolPP::TLS_DHE_DSS_WITH_CAMELLIA_128_GCM_SHA256 =0xC080,
ProtocolPP::TLS_DHE_DSS_WITH_CAMELLIA_256_GCM_SHA384 =0xC081, ProtocolPP::TLS_DH_DS-
S_WITH_CAMELLIA_128_GCM_SHA256 =0xC082, ProtocolPP::TLS_DH_DSS_WITH_CAMELLIA_256_-
GCM_SHA384 =0xC083, ProtocolPP::TLS_DH_ANON_WITH_CAMELLIA_128_GCM_SHA256 =0xC084,
ProtocolPP::TLS_DH_ANON_WITH_CAMELLIA_256_GCM_SHA384 =0xC085, ProtocolPP::TLS_ECDHE_-
ECDSA_WITH_CAMELLIA_128_GCM_SHA256 =0xC086, ProtocolPP::TLS_ECDHE_ECDSA_WITH_CA-
MELLIA_256_GCM_SHA384 =0xC087, ProtocolPP::TLS_ECDH_ECDSA_WITH_CAMELLIA_128_GCM_-
SHA256 =0xC088,
ProtocolPP::TLS_ECDH_ECDSA_WITH_CAMELLIA_256_GCM_SHA384 =0xC089, ProtocolPP::TLS_EC-
DHE_RSA_WITH_CAMELLIA_128_GCM_SHA256 =0xC08A, ProtocolPP::TLS_ECDHE_RSA_WITH_CA-
MELLIA_256_GCM_SHA384 =0xC08B, ProtocolPP::TLS_ECDH_RSA_WITH_CAMELLIA_128_GCM_SH-
A256 =0xC08C,
ProtocolPP::TLS_ECDH_RSA_WITH_CAMELLIA_256_GCM_SHA384 =0xC08D, ProtocolPP::TLS_PSK_-
WITH_CAMELLIA_128_GCM_SHA256 =0xC08E, ProtocolPP::TLS_PSK_WITH_CAMELLIA_256_GCM_S-
HA384 =0xC08F, ProtocolPP::TLS_DHE_PSK_WITH_CAMELLIA_128_GCM_SHA256 =0xC090,
ProtocolPP::TLS_DHE_PSK_WITH_CAMELLIA_256_GCM_SHA384 =0xC091, ProtocolPP::TLS_RSA_PS-
K_WITH_CAMELLIA_128_GCM_SHA256 =0xC092, ProtocolPP::TLS_RSA_PSK_WITH_CAMELLIA_256_-
GCM_SHA384 =0xC093, ProtocolPP::TLS_PSK_WITH_CAMELLIA_128_CBC_SHA256 =0xC094,
ProtocolPP::TLS_PSK_WITH_CAMELLIA_256_CBC_SHA384 =0xC095, ProtocolPP::TLS_DHE_PSK_WI-
TH_CAMELLIA_128_CBC_SHA256 =0xC096, ProtocolPP::TLS_DHE_PSK_WITH_CAMELLIA_256_CBC_-
SHA384 =0xC097, ProtocolPP::TLS_RSA_PSK_WITH_CAMELLIA_128_CBC_SHA256 =0xC098,
ProtocolPP::TLS_RSA_PSK_WITH_CAMELLIA_256_CBC_SHA384 =0xC099, ProtocolPP::TLS_ECDHE_-
PSK_WITH_CAMELLIA_128_CBC_SHA256 =0xC09A, ProtocolPP::TLS_ECDHE_PSK_WITH_CAMELLIA-
_256_CBC_SHA384 =0xC09B, ProtocolPP::TLS_RSA_WITH_AES_128_CCM =0xC09C,
ProtocolPP::TLS_RSA_WITH_AES_256_CCM =0xC09D, ProtocolPP::TLS_DHE_RSA_WITH_AES_128_-
CCM =0xC09E, ProtocolPP::TLS_DHE_RSA_WITH_AES_256_CCM =0xC09F, ProtocolPP::TLS_RSA_WI-
TH_AES_128_CCM_8 =0xC0A0,
ProtocolPP::TLS_RSA_WITH_AES_256_CCM_8 =0xC0A1, ProtocolPP::TLS_DHE_RSA_WITH_AES_128_-
CCM_8 =0xC0A2, ProtocolPP::TLS_DHE_RSA_WITH_AES_256_CCM_8 =0xC0A3, ProtocolPP::TLS_P-
SK_WITH_AES_128_CCM =0xC0A4,
ProtocolPP::TLS_PSK_WITH_AES_256_CCM =0xC0A5, ProtocolPP::TLS_DHE_PSK_WITH_AES_128_C-
CM =0xC0A6, ProtocolPP::TLS_DHE_PSK_WITH_AES_256_CCM =0xC0A7, ProtocolPP::TLS_PSK_WIT-
H_AES_128_CCM_8 =0xC0A8,
ProtocolPP::TLS_PSK_WITH_AES_256_CCM_8 =0xC0A9, ProtocolPP::TLS_PSK_DHE_WITH_AES_128_-
CCM_8 =0xC0AA, ProtocolPP::TLS_PSK_DHE_WITH_AES_256_CCM_8 =0xC0AB, ProtocolPP::TLS_E-
CDHE_ECDSA_WITH_AES_128_CCM =0xC0AC,
ProtocolPP::TLS_ECDHE_ECDSA_WITH_AES_256_CCM =0xC0AD, ProtocolPP::TLS_ECDHE_ECDSA_-
WITH_AES_128_CCM_8 =0xC0AE, ProtocolPP::TLS_ECDHE_ECDSA_WITH_AES_256_CCM_8 =0xC0A-
F, ProtocolPP::TLS_ECDHE_RSA_WITH_CHACHA20_POLY1305_SHA256 =0xCCA8,
ProtocolPP::TLS_ECDHE_ECDSA_WITH_CHACHA20_POLY1305_SHA256 =0xCCA9, ProtocolPP::TLS_-
DHE_RSA_WITH_CHACHA20_POLY1305_SHA256 =0xCCAA, ProtocolPP::TLS_PSK_WITH_CHACH-
A20_POLY1305_SHA256 =0xCCAB, ProtocolPP::TLS_ECDHE_PSK_WITH_CHACHA20_POLY1305_SH-
A256 =0xCCAC,
ProtocolPP::TLS_DHE_PSK_WITH_CHACHA20_POLY1305_SHA256 =0xCCAD, ProtocolPP::TLS_RSA_-
PSK_WITH_CHACHA20_POLY1305_SHA256 =0xCCAE }

```

- enum ProtocolPP::wifitype_t { ProtocolPP::WIFIMNG, ProtocolPP::WIFIDAT, ProtocolPP::WIFICTL }
- enum ProtocolPP::wifisub_t {
 ProtocolPP::WIFI0, ProtocolPP::WIFI8, ProtocolPP::WIFI4, ProtocolPP::WIFI12,
 ProtocolPP::WIFI2, ProtocolPP::WIFI10, ProtocolPP::WIFI6, ProtocolPP::WIFI14,
 ProtocolPP::WIFI1, ProtocolPP::WIFI9, ProtocolPP::WIFI5, ProtocolPP::WIFI13,
 ProtocolPP::WIFI3, ProtocolPP::WIFI11, ProtocolPP::WIFI7, ProtocolPP::WIFI15 }
- enum ProtocolPP::wifictext_t {

```

ProtocolPP::WCTRL0, ProtocolPP::SSW, ProtocolPP::GRANT, ProtocolPP::WCTRL3,
ProtocolPP::POLL, ProtocolPP::SSWACK, ProtocolPP::DMGDTs, ProtocolPP::WCTRL7,
ProtocolPP::WCTRL8, ProtocolPP::SSWFDBCK, ProtocolPP::DMGCTS, ProtocolPP::WCTRL11,
ProtocolPP::SPR, ProtocolPP::WCTRL13, ProtocolPP::GRANTACK }

• enum ProtocolPP::wimaxmode_t { ProtocolPP::OFDM, ProtocolPP::OFDMA, ProtocolPP::AGMH_OFDM,
ProtocolPP::AGMH_OFDMA }

• enum ProtocolPP::macsecmode_t {
ProtocolPP::AES_GCM_128, ProtocolPP::AES_GCM_256, ProtocolPP::AES_GCM_XPN_128, ProtocolPP-
::AES_GCM_XPN_256,
ProtocolPP::NULL_MACSEC }

• enum ProtocolPP::dir_id_t { ProtocolPP::DIR_IN, ProtocolPP::DIR_FWD, ProtocolPP::DIR_OUT }

• enum ProtocolPP::role_id_t { ProtocolPP::ROLE_INITIATOR =0x08, ProtocolPP::ROLE_RESPONDER
=0x20, ProtocolPP::ROLE_ANY =0x28 }

• enum ProtocolPP::ipsec_mode_t { ProtocolPP::TRANSPORT_MODE, ProtocolPP::TUNNEL_MODE }

• enum ProtocolPP::exchg_t {
ProtocolPP::EXCHG_IKE_SA_INIT =34, ProtocolPP::EXCHG_IKE_AUTH, ProtocolPP::EXCHG_CREATE_-_
CHILD_SA, ProtocolPP::EXCHG_INFORMATIONAL,
ProtocolPP::EXCHG_IKE_SESSION_RESUME, ProtocolPP::EXCHG_GSA_AUTH, ProtocolPP::EXCHG_-_
GSA_REGISTRATION, ProtocolPP::EXCHG_GSA_REKEY,
ProtocolPP::EXCHG_IKE_PARSE =253, ProtocolPP::EXCHG_IKE resend, ProtocolPP::EXCHG_IKE_L-
ISTEN }

• enum ProtocolPP::ike_pyld_t {
ProtocolPP::PYLD_NONE =0, ProtocolPP::PYLD_SA =33, ProtocolPP::PYLD_KE, ProtocolPP::PYLD_IDI,
ProtocolPP::PYLD_IDR, ProtocolPP::PYLD_CERT, ProtocolPP::PYLD_CERTREQ, ProtocolPP::PYLD_AU-
TH,
ProtocolPP::PYLD_NINR, ProtocolPP::PYLD_N, ProtocolPP::PYLD_D, ProtocolPP::PYLD_V,
ProtocolPP::PYLD_TSI, ProtocolPP::PYLD_TS, ProtocolPP::PYLD_SK, ProtocolPP::PYLD_CP,
ProtocolPP::PYLD_EAP, ProtocolPP::PYLD_GSPM, ProtocolPP::PYLD_IDG, ProtocolPP::PYLD_GSA,
ProtocolPP::PYLD_KD, ProtocolPP::PYLD_SKF, ProtocolPP::PYLD_PS }

• enum ProtocolPP::transform_type_t {
ProtocolPP::IKE_ENCR =1, ProtocolPP::IKE_PRF, ProtocolPP::IKE_INTEG, ProtocolPP::IKE_DH,
ProtocolPP::IKE_ESN }

• enum ProtocolPP::transform_attr_t { ProtocolPP::ATTR_KEY_LENGTH =14 }

• enum ProtocolPP::encr_id_t {
ProtocolPP::ENCR_DES_IV64 =1, ProtocolPP::ENCR_DES, ProtocolPP::ENCR_3DES, ProtocolPP::ENC-
R_RC5,
ProtocolPP::ENCR_IDEA, ProtocolPP::ENCR_CAST, ProtocolPP::ENCR_BLOWFISH, ProtocolPP::ENCR-
_3IDEA,
ProtocolPP::ENCR_DES_IV32, ProtocolPP::ENCR_RSVD1, ProtocolPP::ENCR_NULL, ProtocolPP::ENCR-
_AES_CBC,
ProtocolPP::ENCR_AES_CTR, ProtocolPP::ENCR_AES_CCM_8, ProtocolPP::ENCR_AES_CCM_12,
ProtocolPP::ENCR_AES_CCM_16,
ProtocolPP::ENCR_UNASSIGNED, ProtocolPP::ENCR_AES_GCM_8, ProtocolPP::ENCR_AES_GCM_12,
ProtocolPP::ENCR_AES_GCM_16,
ProtocolPP::ENCR_NULL_AUTH_AES_GMAC, ProtocolPP::ENCR_RSVD_AES_XTS, ProtocolPP::ENCR-
_CAMELLIA_CBC, ProtocolPP::ENCR_CAMELLIA_CTR,
ProtocolPP::ENCR_CAMELLIA_CCM_8, ProtocolPP::ENCR_CAMELLIA_CCM_12, ProtocolPP::ENCR_C-
AMELLIA_CCM_16, ProtocolPP::ENCR_CHACHA20_POLY1305,
ProtocolPP::ENCR_AES_CCM_8_IIV, ProtocolPP::ENCR_AES_GCM_16_IIV, ProtocolPP::ENCR_CHAC-
HA20_POLY1305_IIV }

• enum ProtocolPP::prf_id_t {
ProtocolPP::PRF_HMAC_MD5 =1, ProtocolPP::PRF_HMAC_SHA1, ProtocolPP::PRF_HMAC_TIGER,
ProtocolPP::PRF_AES128_XCBC,
ProtocolPP::PRF_HMAC_SHA2_256, ProtocolPP::PRF_HMAC_SHA2_384, ProtocolPP::PRF_HMAC_SH-
A2_512, ProtocolPP::PRF_AES128_CMAC }

• enum ProtocolPP::integ_id_t {
ProtocolPP::AUTH_NONE, ProtocolPP::AUTH_HMAC_MD5_96, ProtocolPP::AUTH_HMAC_SHA1_96,

```

```

ProtocolPP::AUTH_DES_MAC,
ProtocolPP::AUTH_KPDK_MD5, ProtocolPP::AUTH_AES_XCBC_96, ProtocolPP::AUTH_HMAC_MD5_-128, ProtocolPP::AUTH_HMAC_SHA1_160,
ProtocolPP::AUTH_AES_CMAC_96, ProtocolPP::AUTH_AES_128_GMAC, ProtocolPP::AUTH_AES_192_-GMAC, ProtocolPP::AUTH_AES_256_GMAC,
ProtocolPP::AUTH_HMAC_SHA2_256_128, ProtocolPP::AUTH_HMAC_SHA2_384_192, ProtocolPP::AUTH_HMAC_SHA2_512_256 }

• enum ProtocolPP::dh_id_t {
ProtocolPP::DH_NONE, ProtocolPP::DH_MODP_768, ProtocolPP::DH_MODP_1024, ProtocolPP::DH_RS-VD3,
ProtocolPP::DH_RSVD4, ProtocolPP::DH_MODP_1536, ProtocolPP::DH_UNASSIGNED6, ProtocolPP::DH_UNASSIGNED7,
ProtocolPP::DH_UNASSIGNED8, ProtocolPP::DH_UNASSIGNED9, ProtocolPP::DH_UNASSIGNED10,
ProtocolPP::DH_UNASSIGNED11,
ProtocolPP::DH_UNASSIGNED12, ProtocolPP::DH_UNASSIGNED13, ProtocolPP::DH_MODP_2048,
ProtocolPP::DH_MODP_3072,
ProtocolPP::DH_MODP_4096, ProtocolPP::DH_MODP_6144, ProtocolPP::DH_MODP_8192, ProtocolPP::DH_ECP_256,
ProtocolPP::DH_ECP_384, ProtocolPP::DH_ECP_521, ProtocolPP::DH_MODP_1024_160, ProtocolPP::DH_MODP_2048_224,
ProtocolPP::DH_MODP_2048_256, ProtocolPP::DH_ECP_192_RANDOM, ProtocolPP::DH_ECP_224_RANDOM, ProtocolPP::DH_BRAINPOOL_P224R1,
ProtocolPP::DH_BRAINPOOL_P256R1, ProtocolPP::DH_BRAINPOOL_P384R1, ProtocolPP::DH_BRAINPOOL_P512R1, ProtocolPP::DH_CURVE_25519,
ProtocolPP::DH_CURVE_448 }

• enum ProtocolPP::esn_id_t { ProtocolPP::ESN_NO, ProtocolPP::ESN_YES }

• enum ProtocolPP::id_type_t {
ProtocolPP::ID_IPV4_ADDR =1, ProtocolPP::ID_FQDN, ProtocolPP::ID_RFC822_ADDR, ProtocolPP::ID_UNASSIGNED4,
ProtocolPP::ID_IPV6_ADDR, ProtocolPP::ID_UNASSIGNED6, ProtocolPP::ID_UNASSIGNED7, ProtocolPP::ID_UNASSIGNED8,
ProtocolPP::ID_DER ASN1_DN, ProtocolPP::ID_DER ASN1_GN, ProtocolPP::ID_KEY_ID, ProtocolPP::ID_FC_NAME,
ProtocolPP::ID_NULL }

• enum ProtocolPP::encoding_t {
ProtocolPP::CERT_X509_PKCS_7 =1, ProtocolPP::CERT_PGP, ProtocolPP::CERT_DNS, ProtocolPP::CERT_X509_SIGNATURE,
ProtocolPP::CERT_RESERVED5, ProtocolPP::CERT_KERBEROS, ProtocolPP::CERT_CRL, ProtocolPP::CERT_ARL,
ProtocolPP::CERT_SPKI, ProtocolPP::CERT_RAW_RSA, ProtocolPP::CERT_X509_ATTRIBUTE, ProtocolPP::CERT_HASH_URL,
ProtocolPP::CERT_HASH_URL_BUNDLE, ProtocolPP::CERT_OSCP_CONTENT, ProtocolPP::CERT_RAW_PUBLIC_KEY }

• enum ProtocolPP::auth_method_t {
ProtocolPP::AUTH_RSA =1, ProtocolPP::AUTH_PSK, ProtocolPP::AUTH_DSS, ProtocolPP::AUTH_UNASSIGNED4,
ProtocolPP::AUTH_UNASSIGNED5, ProtocolPP::AUTH_UNASSIGNED6, ProtocolPP::AUTH_UNASSIGNED7, ProtocolPP::AUTH_UNASSIGNED8,
ProtocolPP::AUTH_ECDSA_P256, ProtocolPP::AUTH_ECDSA_P384, ProtocolPP::AUTH_ECDSA_P512,
ProtocolPP::AUTH_GENERIC_SECURE_PASSWORD,
ProtocolPP::AUTH_NULL, ProtocolPP::AUTH_DIGITAL_SIGNATURE }

• enum ProtocolPP::notify_err_t {
ProtocolPP::NERR_UNSUPPORTED_CRIT_PAYLD =1, ProtocolPP::NERR_RSVD2, ProtocolPP::NERR_RSVD3, ProtocolPP::NERR_INVALID_IKE_SPI,
ProtocolPP::NERR_INVALID_MAJOR_VERSION, ProtocolPP::NERR_RSVD6, ProtocolPP::NERR_INVALID_Syntax, ProtocolPP::NERR_RSVD8,
ProtocolPP::NERR_INVALID_MESSAGE_ID, ProtocolPP::NERR_RSVD10, ProtocolPP::NERR_INVALID_

```

```

SPI, ProtocolPP::NERR_RSVD12,
ProtocolPP::NERR_RSVD13, ProtocolPP::NERR_NO_PROPOSAL_CHOSEN, ProtocolPP::NERR_RSVD15,
ProtocolPP::NERR_RSVD16,
ProtocolPP::NERR_INVALID_KE_PAYLOAD, ProtocolPP::NERR_RSVD18, ProtocolPP::NERR_RSVD19,
ProtocolPP::NERR_RSVD20,
ProtocolPP::NERR_RSVD21, ProtocolPP::NERR_RSVD22, ProtocolPP::NERR_RSVD23, ProtocolPP::NERR_AUTH_FAILED,
ProtocolPP::NERR_RSVD25, ProtocolPP::NERR_RSVD26, ProtocolPP::NERR_RSVD27, ProtocolPP::NERR_RSVD28,
ProtocolPP::NERR_RSVD29, ProtocolPP::NERR_RSVD30, ProtocolPP::NERR_RSVD31, ProtocolPP::NERR_RSVD32,
ProtocolPP::NERR_RSVD33, ProtocolPP::NERR_SINGLE_PAIR_REQUIRED, ProtocolPP::NERR_NO_ADDITIONAL_SAS,
ProtocolPP::NERR_INTERNAL_ADDRESS_FAILURE,
ProtocolPP::NERR_FAILED_CP_REQUIRED, ProtocolPP::NERR_TS_UNACCEPTABLE, ProtocolPP::NERR_INVALID_SELECTORS,
ProtocolPP::NERR_UNACCEPTABLE_ADDRESSES,
ProtocolPP::NERR_UNEXPECTED_NAT_DETECTED, ProtocolPP::NERR_USE_ASSIGNED_HOA,
ProtocolPP::NERR_TEMPORARY_FAILURE, ProtocolPP::NERR_CHILD_SA_NOT_FOUND,
ProtocolPP::NERR_INVALID_GROUP_ID, ProtocolPP::NERR_AUTHORIZATION_FAILED }

• enum ProtocolPP::notify_status_t {
ProtocolPP::NSTAT_INITIAL_CONTACT =16384, ProtocolPP::NSTAT_SET_WINDOW_SIZE, ProtocolPP::NSTAT_ADDITIONAL_TS_POSSIBLE, ProtocolPP::NSTAT_IPCOMP_SUPPORTED,
ProtocolPP::NSTAT_NAT_DETECTION_SOURCE_IP, ProtocolPP::NSTAT_NAT_DETECTION_DESTINATION_IP, ProtocolPP::NSTAT_COOKIE, ProtocolPP::NSTAT_USE_TRANSPORT_MODE,
ProtocolPP::NSTAT_HTTP_CERT_LOOKUP_SUPPORTED, ProtocolPP::NSTAT_REKEY_SA, ProtocolPP::NSTAT_ESP_TFC_PADDING_NOT_SUPPORTED, ProtocolPP::NSTAT_NON_FIRST_FRAGMENTS ALSO,
ProtocolPP::NSTAT_MOBIKE_SUPPORTED, ProtocolPP::NSTAT_ADDITIONAL_IP4_ADDRESS, ProtocolPP::NSTAT_ADDITIONAL_IP6_ADDRESS, ProtocolPP::NSTAT_NO_ADDITIONAL_ADDRESSES,
ProtocolPP::NSTAT_UPDATE_SA_ADDRESSES, ProtocolPP::NSTAT_COOKIE2, ProtocolPP::NSTAT_NO_NATS_ALLOWED, ProtocolPP::NSTAT_AUTH_LIFETIME,
ProtocolPP::NSTAT_MULTIPLE_AUTH_SUPPORTED, ProtocolPP::NSTAT_ANOTHER_AUTH_FOLLOW_S, ProtocolPP::NSTAT_REDIRECT_SUPPORTED, ProtocolPP::NSTAT_REDIRECT,
ProtocolPP::NSTAT_REDIRECTED_FROM, ProtocolPP::NSTAT_TICKET_LT_OPAQUE, ProtocolPP::NSTAT_TICKET_REQUEST, ProtocolPP::NSTAT_TICKET_ACK,
ProtocolPP::NSTAT_TICKET_NACK, ProtocolPP::NSTAT_TICKET_OPAQUE, ProtocolPP::NSTAT_LINK_ID, ProtocolPP::NSTAT_USE_WESP_MODE,
ProtocolPP::NSTAT_ROHC_SUPPORTED, ProtocolPP::NSTAT_EAP_ONLY_AUTHENTICATION, ProtocolPP::NSTAT_CHILDLESS_IKEV2_SUPPORTED, ProtocolPP::NSTAT_QUICK_CRASH_DETECTION,
ProtocolPP::NSTAT_IKEV2_MESSAGE_ID_SYNC_SUPPORTED, ProtocolPP::NSTAT_IPSEC_REPLY_COUNTER_SYNC_SUPPORTED, ProtocolPP::NSTAT_IKEV2_MESSAGE_ID_SYNC, ProtocolPP::NSTAT_IPSEC_REPLY_COUNTER_SYNC,
ProtocolPP::NSTAT_SECURE_PASSWORD_METHODS, ProtocolPP::NSTAT_PSK_PERSIST, ProtocolPP::NSTAT_PSK_CONFIRM, ProtocolPP::NSTAT_ERX_SUPPORTED,
ProtocolPP::NSTAT_IFOM_CAPABILITY, ProtocolPP::NSTAT_SENDER_REQUEST_ID, ProtocolPP::NSTAT_IKEV2_FRAGMENTATION_SUPPORTED, ProtocolPP::NSTAT_SIGNATURE_HASH_ALGORITHMS,
ProtocolPP::NSTAT_CLONE_IKE_SA_SUPPORTED, ProtocolPP::NSTAT_CLONE_IKE_SA, ProtocolPP::NSTAT_PUZZLE, ProtocolPP::NSTAT_USE_PPK,
ProtocolPP::NSTAT_PPK_IDENTITY, ProtocolPP::NSTAT_NO_PPK_AUTH }

• enum ProtocolPP::ike_ipcomp_t { ProtocolPP::IPCOMP_OUI =1, ProtocolPP::IPCOMP_DEFLATE, ProtocolPP::IPCOMP_LZS, ProtocolPP::IPCOMP_LZJH }

• enum ProtocolPP::ip_proto_t {
ProtocolPP::IP_PROTO_ANY, ProtocolPP::IP_PROTO_ICMP, ProtocolPP::IP_PROTO_TCP, ProtocolPP::IP_PROTO_UDP,
ProtocolPP::IP_PROTO_ICMPV6, ProtocolPP::IP_PROTO_MH }

• enum ProtocolPP::protocol_id_t {
ProtocolPP::PROTO_IKE =1, ProtocolPP::PROTO_AH, ProtocolPP::PROTO_ESP, ProtocolPP::PROTO_F_C_ESP_HDR,
ProtocolPP::PROTO_FC_CT_AUTH }

```

- enum `ProtocolPP::ike_ts_t` { `ProtocolPP::TS_IPV4_ADDR_RANGE` =7, `ProtocolPP::TS_IPV6_ADDR_RANGE`, `ProtocolPP::TS_FC_ADDR_RANGE` }
- enum `ProtocolPP::cfg_resp_t` { `ProtocolPP::CFG_REQUEST` =1, `ProtocolPP::CFG_REPLY`, `ProtocolPP::CFG_SET`, `ProtocolPP::CFG_ACK` }
- enum `ProtocolPP::cfg_attr_t` {
`ProtocolPP::CFG_INTERNAL_IP4_ADDRESS` =1, `ProtocolPP::CFG_INTERNAL_IP4_NETMASK`, `ProtocolPP::CFG_INTERNAL_IP4_DNS`, `ProtocolPP::CFG_INTERNAL_IP4_NBNS`,
`ProtocolPP::CFG_RSVD5`, `ProtocolPP::CFG_INTERNAL_IP4_DHCP`, `ProtocolPP::CFG_APPLICATION_VERSION`, `ProtocolPP::CFG_INTERNAL_IP6_ADDRESS`,
`ProtocolPP::CFG_RSVD9`, `ProtocolPP::CFG_INTERNAL_IP6_DNS`, `ProtocolPP::CFG_RSVD11`, `ProtocolPP::CFG_INTERNAL_IP6_DHCP`,
`ProtocolPP::CFG_INTERNAL_IP4_SUBNET`, `ProtocolPP::CFG_SUPPORTED_ATTRIBUTES`, `ProtocolPP::CFG_INTERNAL_IP6_SUBNET`, `ProtocolPP::CFG_MIP6_HOME_PREFIX`,
`ProtocolPP::CFG_INTERNAL_IP6_LINK`, `ProtocolPP::CFG_INTERNAL_IP6_PREFIX`, `ProtocolPP::CFG_HOME_AGENT_ADDR`, `ProtocolPP::CFG_P_CSCG_IP4_ADDR`,
`ProtocolPP::CFG_P_CSCF_IP6_ADDR`, `ProtocolPP::CFG_FTT_KAT`, `ProtocolPP::CFG_EXT_SRC_IP4_NAT`, `ProtocolPP::CFG_TIMEOUT_LIVENESS`,
`ProtocolPP::CFG_INTERNAL_DNS_DOMAIN`, `ProtocolPP::CFG_INTERNAL_DNS_SEC_TA` }
- enum `ProtocolPP::ike_gateway_t` { `ProtocolPP::GATEWAY_IPV4_VPN_ADDR` =1, `ProtocolPP::GATEWAY_IPV6_VPN_ADDR`, `ProtocolPP::GATEWAY_FQDN_VPN_ADDR` }
- enum `ProtocolPP::rohc_attr_t` {
`ProtocolPP::ROHC_MAX_CID`=1, `ProtocolPP::ROHC_PROFILE`, `ProtocolPP::ROHC_INTEG`, `ProtocolPP::ROHC_ICV_LEN`,
`ProtocolPP::ROHC_MRRU` }
- enum `ProtocolPP::ike_secpass_t` { `ProtocolPP::SECURE_PASSWORD_PACE` =1, `ProtocolPP::SECURE_PASSWORD0RD_AUGPAKE`, `ProtocolPP::SECURE_PASSWORD_PSK_AUTH` }
- enum `ProtocolPP::ike_hash_t` {
`ProtocolPP::HASH_SHA1`, `ProtocolPP::HASH_SHA2_256`, `ProtocolPP::HASH_SHA2_384`, `ProtocolPP::HASH_SHA2_512`,
`ProtocolPP::HASH_IDENTITY` }
- enum `ProtocolPP::rsapadtype_t` { `ProtocolPP::PKCS15`, `ProtocolPP::PSS` }

Functions

- `ProtocolPP::Begin_Enum_String` (`field_t`)
- `ProtocolPP::Begin_Enum_String` (`endian_t`)
- `ProtocolPP::Begin_Enum_String` (`platform_t`)
- `ProtocolPP::Begin_Enum_String` (`pad_t`)
- `ProtocolPP::Begin_Enum_String` (`protocol_t`)
- *param ETH_P PPP_MP Dummy type
for PPP MP frames(0x0008)*@param ETH_P_LOCALTALK-Localtalk pseudo type(0x0009)*@param ETH_P_CAN-CAN `ProtocolPP::Begin_Enum_String` (`ethid_t`)
- `ProtocolPP::Begin_Enum_String` (`direction_t`)
- `ProtocolPP::Begin_Enum_String` (`cipher_t`)
- `ProtocolPP::Begin_Enum_String` (`auth_t`)
- `ProtocolPP::Begin_Enum_String` (`iana_t`)
- `ProtocolPP::Begin_Enum_String` (`err_t`)
- `ProtocolPP::Begin_Enum_String` (`replay_t`)
- `ProtocolPP::Begin_Enum_String` (`ipmode_t`)
- `ProtocolPP::Begin_Enum_String` (`icmppmsg_t`)
- `ProtocolPP::Begin_Enum_String` (`icmpcode_t`)
- `ProtocolPP::Begin_Enum_String` (`tlsver_t`)
- `ProtocolPP::Begin_Enum_String` (`tlstype_t`)
- `ProtocolPP::Begin_Enum_String` (`tslvl_t`)
- `ProtocolPP::Begin_Enum_String` (`tls_handshake_t`)
- `ProtocolPP::Begin_Enum_String` (`tls_error_t`)

- [ProtocolPP::Begin_Enum_String \(srtpcipher_t\)](#)
- [ProtocolPP::Begin_Enum_String \(tls_ciphersuite_t\)](#)
- [ProtocolPP::Begin_Enum_String \(wifitype_t\)](#)
- [ProtocolPP::Begin_Enum_String \(wifisub_t\)](#)
- [ProtocolPP::Begin_Enum_String \(wifictext_t\)](#)
- [ProtocolPP::Begin_Enum_String \(wimaxmode_t\)](#)
- [ProtocolPP::Begin_Enum_String \(macsecmode_t\)](#)
- [ProtocolPP::Begin_Enum_String \(dir_id_t\)](#)
- [ProtocolPP::Begin_Enum_String \(role_id_t\)](#)
- [ProtocolPP::Begin_Enum_String \(ipsec_mode_t\)](#)
- [ProtocolPP::Begin_Enum_String \(exchg_t\)](#)
- [ProtocolPP::Begin_Enum_String \(ike_pyld_t\)](#)
- [ProtocolPP::Begin_Enum_String \(transform_type_t\)](#)
- [ProtocolPP::Begin_Enum_String \(transform_attr_t\)](#)
- [ProtocolPP::Begin_Enum_String \(encr_id_t\)](#)
- [ProtocolPP::Begin_Enum_String \(prf_id_t\)](#)
- [ProtocolPP::Begin_Enum_String \(integ_id_t\)](#)
- [ProtocolPP::Begin_Enum_String \(dh_id_t\)](#)
- [ProtocolPP::Begin_Enum_String \(esn_id_t\)](#)
- [ProtocolPP::Begin_Enum_String \(id_type_t\)](#)
- [ProtocolPP::Begin_Enum_String \(encoding_t\)](#)
- [ProtocolPP::Begin_Enum_String \(auth_method_t\)](#)
- [ProtocolPP::Begin_Enum_String \(notify_err_t\)](#)
- [ProtocolPP::Begin_Enum_String \(notify_status_t\)](#)
- [ProtocolPP::Begin_Enum_String \(ike_ipcomp_t\)](#)
- [ProtocolPP::Begin_Enum_String \(ip_proto_t\)](#)
- [ProtocolPP::Begin_Enum_String \(protocol_id_t\)](#)
- [ProtocolPP::Begin_Enum_String \(ike_ts_t\)](#)
- [ProtocolPP::Begin_Enum_String \(cfg_resp_t\)](#)
- [ProtocolPP::Begin_Enum_String \(cfg_attr_t\)](#)
- [ProtocolPP::Begin_Enum_String \(ike_gateway_t\)](#)
- [ProtocolPP::Begin_Enum_String \(rohc_attr_t\)](#)
- [ProtocolPP::Begin_Enum_String \(ike_secpass_t\)](#)
- [ProtocolPP::Begin_Enum_String \(ike_hash_t\)](#)
- [ProtocolPP::Begin_Enum_String \(rsapadtype_t\)](#)

Variables

- [ProtocolPP::End_Enum_String](#)

8.15.1 Macro Definition Documentation

```
8.15.1.1 #define HEX( a, b ) "0x" << std::setw(b) << std::setfill('0') << std::hex << static_cast<unsigned long long>(a)
           << std::dec
```

Converts numbers into hexadecimal representation

Parameters

<code>a</code>	- value to convert to hexadecimal
----------------	-----------------------------------

<i>b</i>	- number of hexadecimal characters to use for representation
----------	--

8.15.1.2 #define HEXLOG(*a*, *b*) "0x", std::setw(*b*), std::setfill('0'), std::hex, static_cast<unsigned long long>(*a*), std::dec

8.16 include/jicmp.h File Reference

```
#include "jicmpsah.h"
#include "jprotocol.h"
```

Classes

- class [ProtocolPP::jicmp](#)

Namespaces

- [ProtocolPP](#)

8.17 include/jicmpsah.h File Reference

```
#include "jsecass.h"
```

Classes

- class [ProtocolPP::jicmpsah](#)

Namespaces

- [ProtocolPP](#)

8.18 include/jip.h File Reference

```
#include "jipsah.h"
#include "jprotocol.h"
```

Classes

- class [ProtocolPP::jip](#)

Namespaces

- [ProtocolPP](#)

8.19 include/jipsa.h File Reference

```
#include "jsecass.h"
```

Classes

- class [ProtocolPP::jipsa](#)

Namespaces

- [ProtocolPP](#)

8.20 include/jipsec.h File Reference

```
#include "jipsecsa.h"
#include "jprotocol.h"
```

Classes

- class [ProtocolPP::jipsec](#)

Namespaces

- [ProtocolPP](#)

8.21 include/jipsecsa.h File Reference

```
#include "jsecass.h"
```

Classes

- class [ProtocolPP::jipsecsa](#)

Namespaces

- [ProtocolPP](#)

8.22 include/jlte.h File Reference

```
#include "jltesa.h"
#include "jprotocol.h"
```

Classes

- class [ProtocolPP::jlte](#)

Namespaces

- [ProtocolPP](#)

8.23 include/jltesa.h File Reference

```
#include "jsecass.h"
```

Classes

- class [ProtocolPP::jltesa](#)

Namespaces

- [ProtocolPP](#)

8.24 include/jmacsec.h File Reference

```
#include "jmacsecsa.h"
#include "jprotocol.h"
```

Classes

- class [ProtocolPP::jmacsec](#)

Namespaces

- [ProtocolPP](#)

8.25 include/jmacsecsa.h File Reference

```
#include "jsecass.h"
```

Classes

- class [ProtocolPP::jmacsecsa](#)

Namespaces

- [ProtocolPP](#)

8.26 include/jmodes.h File Reference

```
#include <iostream>
#include <fstream>
#include <exception>
#include <cstring>
#include <memory>
#include "cryptlib.h"
#include "hex.h"
#include "aes.h"
#include "sm4.h"
#include "gcm.h"
#include "des.h"
#include "ccm.h"
#include "camellia.h"
#include "aria.h"
#include "seed.h"
#include "chacha.h"
#include "modes.h"
#include "filters.h"
#include "md5.h"
#include "sha.h"
#include "sha3.h"
#include "hmac.h"
#include "poly1305.h"
#include "sm3.h"
#include "assert.h"
#include "jenum.h"
#include "jrand.h"
#include "jarray.h"
#include "arc4.h"
#include "crc.h"
#include "jsnow3g.h"
#include "jzuc.h"
#include "jpoly1305.h"
#include "chacha20.h"
```

Classes

- class [ProtocolPP::jmodes](#)

Namespaces

- [ProtocolPP](#)

Macros

- `#define CRYPTOPP_ENABLE_NAMESPACE_WEAK 1`

Typedefs

- `typedef unsigned char Byte`
- `typedef unsigned int Word`

8.26.1 Macro Definition Documentation

8.26.1.1 `#define CRYPTOPP_ENABLE_NAMESPACE_WEAK 1`

8.26.2 Typedef Documentation

8.26.2.1 `typedef unsigned char Byte`

8.26.2.2 `typedef unsigned int Word`

8.27 include/jpacket.h File Reference

```
#include <memory>
#include "jarray.h"
```

Classes

- class [ProtocolPP::jpacket](#)

Namespaces

- [ProtocolPP](#)

8.28 include/jpoly1305.h File Reference

```
#include <stddef.h>
#include <iostream>
#include <cstring>
```

Classes

- class [ProtocolPP::jpoly1305](#)

Namespaces

- [ProtocolPP](#)

Macros

- `#define HAS_SIZEOF_INT128_64BIT (defined(__SIZEOF_INT128__) && defined(__LP64__))`
- `#define HAS_MSVC_64BIT (defined(_MSC_VER) && defined(_M_X64))`
- `#define HAS_GCC_4_4_64BIT (defined(__GNUC__) && defined(__LP64__) && ((__GNUC__ > 4) || ((__GNUC__ == 4) && (__GNUC_MINOR__ >= 4))))`
- `#define JPOLY1305_32BIT`

8.28.1 Macro Definition Documentation

- 8.28.1.1 `#define HAS_GCC_4_4_64BIT (defined(__GNUC__) && defined(__LP64__) && ((__GNUC__ > 4) || (__GNUC__ == 4) && (__GNUC_MINOR__ >= 4)))`
- 8.28.1.2 `#define HAS_MSVC_64BIT (defined(_MSC_VER) && defined(_M_X64))`
- 8.28.1.3 `#define HAS_SIZEOF_INT128_64BIT (defined(__SIZEOF_INT128__) && defined(__LP64__))`
- 8.28.1.4 `#define JPOLY1305_32BIT`

8.29 include/jprotocol.h File Reference

```
#include <mutex>
#include <random>
#include <cstdint>
#include <sys/types.h>
#include <exception>
#include <ratio>
#include <chrono>
#include <memory>
#include <string>
#include <fstream>
#include <iostream>
#include <sstream>
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <unistd.h>
#include <sys/wait.h>
#include "assert.h"
#include "EnumString.h"
#include "jenum.h"
#include "jarray.h"
#include "jrand.h"
#include "jreplay.h"
#include "ciphers.h"
#include "tinyxml2.h"
```

Classes

- class [ProtocolPP::jprotocol](#)

Namespaces

- [ProtocolPP](#)

8.30 include/jprotocolpp.h File Reference

```
#include "jprotocol.h"
#include "judp.h"
#include "jtcp.h"
#include "jip.h"
#include "jicmp.h"
#include "jipsec.h"
#include "jtls.h"
#include "jmacsec.h"
#include "jsrtp.h"
#include "jwifi.h"
#include "jwimax.h"
#include "jlte.h"
#include "jstream.h"
#include "jpacket.h"
#include "jdata.h"
```

Classes

- class [ProtocolPP::jprotocolpp](#)

Namespaces

- [ProtocolPP](#)

Macros

- `#define PROTOCOLPP_VERSION 301`

8.30.1 Macro Definition Documentation

8.30.1.1 `#define PROTOCOLPP_VERSION 301`

8.31 include/jrand.h File Reference

```
#include <string.h>
#include "jarray.h"
#include <random>
#include "mtrand.h"
#include <stdio.h>
#include <limits.h>
#include <time.h>
#include <stdlib.h>
#include "SFMT.h"
```

Classes

- class [ProtocolPP::jrand](#)

Namespaces

- [ProtocolPP](#)

8.32 include/jreplay.h File Reference

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <string>
#include "jarray.h"
#include "jenum.h"
```

Classes

- class [ProtocolPP::jreplay< T, TE >](#)

Namespaces

- [ProtocolPP](#)

8.33 include/jrsa.h File Reference

```
#include <sys/types.h>
#include <unistd.h>
#include <netdb.h>
#include <string.h>
#include <crypt.h>
#include <memory>
#include "jlogger.h"
#include "jenum.h"
#include "jarray.h"
#include "jrand.h"
#include "nbtheory.h"
#include "osrng.h"
#include "sha.h"
#include "aes.h"
#include "rsa.h"
#include "pssr.h"
#include "secblock.h"
#include "oids.h"
#include "asn.h"
#include "integer.h"
```

Classes

- class [ProtocolPP::jrsa](#)

Namespaces

- [ProtocolPP](#)

8.34 include/jsecass.h File Reference

```
#include "jprotocol.h"
```

Classes

- class [ProtocolPP::jsecass](#)

Namespaces

- [ProtocolPP](#)

8.35 include/jsnow3g.h File Reference

```
#include <cmath>
#include <cstring>
#include <iostream>
#include <vector>
#include "jenum.h"
```

Classes

- class [ProtocolPP::jsnow3g](#)

Namespaces

- [ProtocolPP](#)

8.36 include/jsrtp.h File Reference

```
#include "jsrtpsa.h"
#include "jprotocol.h"
```

Classes

- class [ProtocolPP::jsrtp](#)

Namespaces

- [ProtocolPP](#)

8.37 include/jsrtpsa.h File Reference

```
#include "jsecass.h"
```

Classes

- class [ProtocolPP::jsrtpsa](#)

Namespaces

- [ProtocolPP](#)

8.38 include/jstream.h File Reference

```
#include <memory>
#include <cstddef>
#include "jsecass.h"
#include "jprotocolpp.h"
```

Classes

- class [ProtocolPP::jstream](#)

Namespaces

- [ProtocolPP](#)

8.39 include/jtcp.h File Reference

```
#include "jtcp.h"
#include "jprotocol.h"
```

Classes

- class [ProtocolPP::jtcp](#)

Namespaces

- [ProtocolPP](#)

8.40 include/jtcsa.h File Reference

```
#include "jsecass.h"
```

Classes

- class [ProtocolPP::jtcpса](#)

Namespaces

- [ProtocolPP](#)

8.41 include/jtls.h File Reference

```
#include <regex>
#include "jtlsa.h"
#include "jprotocol.h"
```

Classes

- class [ProtocolPP::jtls](#)

Namespaces

- [ProtocolPP](#)

8.42 include/jtlsa.h File Reference

```
#include <regex>
#include "jsecass.h"
```

Classes

- class [ProtocolPP::jtlsa](#)

Namespaces

- [ProtocolPP](#)

8.43 include/judp.h File Reference

```
#include "judpsa.h"
#include "jprotocol.h"
```

Classes

- class [ProtocolPP::judp](#)

Namespaces

- [ProtocolPP](#)

8.44 include/judpsa.h File Reference

```
#include "jsecass.h"
```

Classes

- class [ProtocolPP::judpsa](#)

Namespaces

- [ProtocolPP](#)

8.45 include/jwifi.h File Reference

```
#include "jwifisa.h"
#include "jprotocol.h"
```

Classes

- class [ProtocolPP::jwifi](#)

Namespaces

- [ProtocolPP](#)

8.46 include/jwifisa.h File Reference

```
#include "jsecass.h"
```

Classes

- class [ProtocolPP::jwifisa](#)

Namespaces

- [ProtocolPP](#)

8.47 include/jwimax.h File Reference

```
#include "jwimaxsa.h"
#include "jprotocol.h"
```

Classes

- class [ProtocolPP::jwimax](#)

Namespaces

- [ProtocolPP](#)

8.48 include/jwimaxsa.h File Reference

```
#include "jsecass.h"
```

Classes

- class [ProtocolPP::jwimaxsa](#)

Namespaces

- [ProtocolPP](#)

8.49 include/jzuc.h File Reference

```
#include <cstring>
#include <iostream>
#include "jenum.h"
```

Classes

- class [ProtocolPP::jzuc](#)

Namespaces

- [ProtocolPP](#)

Macros

- `#define MulByPow2(x, k) (((x) << k) | ((x) >> (31 - k))) & 0xFFFFFFFF`
- `#define ROT(a, k) (((a) << k) | ((a) >> (32 - k)))`
- `#define MAKEuint32_t(a, b, c, d) (((uint32_t)(a) << 24) | ((uint32_t)(b) << 16) | ((uint32_t)(c) << 8) | ((uint32_t)(d)))`
- `#define MAKEU31(a, b, c) (((uint32_t)(a) << 23) | ((uint32_t)(b) << 8) | (uint32_t)(c))`

8.49.1 Macro Definition Documentation

- 8.49.1.1 `#define MAKEU31(a, b, c) (((uint32_t)(a) << 23) | ((uint32_t)(b) << 8) | (uint32_t)(c))`
- 8.49.1.2 `#define MAKEuint32_t(a, b, c, d) (((uint32_t)(a) << 24) | ((uint32_t)(b) << 16) | ((uint32_t)(c) << 8) | ((uint32_t)(d)))`
- 8.49.1.3 `#define MulByPow2(x, k) (((x) << k) | ((x) >> (31 - k))) & 0x7FFFFFFF`
- 8.49.1.4 `#define ROT(a, k) (((a) << k) | ((a) >> (32 - k)))`

8.50 include/mtrand.h File Reference

Classes

- class [ProtocolPP::MTRand_int32](#)
Mersenne Twister random number generator.
- class [ProtocolPP::MTRand](#)
- class [ProtocolPP::MTRand_closed](#)
- class [ProtocolPP::MTRand_open](#)
- class [ProtocolPP::MTRand53](#)

Namespaces

- [ProtocolPP](#)

8.51 include/poly1305-16.h File Reference

Classes

- struct [ProtocolPP::jpoly1305_state_internal_t](#)

Namespaces

- [ProtocolPP](#)

Macros

- `#define JPOLY1305_NOINLINE`
- `#define jpoly1305_block_size 16`

TypeDefs

- typedef struct [ProtocolPP::jpoly1305_state_internal_t](#) [ProtocolPP::jpoly1305_state_internal_t](#)

Functions

- static unsigned short [ProtocolPP::U8TO16](#) (const unsigned char *p)
- static void [ProtocolPP::U16TO8](#) (unsigned char *p, unsigned short v)

8.51.1 Macro Definition Documentation

8.51.1.1 `#define jpoly1305_block_size 16`

8.51.1.2 `#define JPOLY1305_NOINLINE`

8.52 include/poly1305-32.h File Reference

Classes

- struct [ProtocolPP::jpoly1305_state_internal_t](#)

Namespaces

- [ProtocolPP](#)

Macros

- `#define JPOLY1305_NOINLINE`
- `#define jpoly1305_block_size 16`

Functions

- static unsigned long [ProtocolPP::U8TO32](#) (const unsigned char *p)
- static void [ProtocolPP::U32TO8](#) (unsigned char *p, unsigned long v)
- static void [ProtocolPP::jpoly1305_blocks](#) (jpoly1305_state_internal_t *st, const unsigned char *m, size_t bytes)

8.52.1 Macro Definition Documentation

8.52.1.1 `#define jpoly1305_block_size 16`

8.52.1.2 `#define JPOLY1305_NOINLINE`

8.53 include/poly1305-64.h File Reference

Classes

- struct [ProtocolPP::jpoly1305_state_internal_t](#)

Namespaces

- [ProtocolPP](#)

Macros

- `#define jpoly1305_block_size 16`

Functions

- static unsigned long long `ProtocolPP::U8TO64` (const unsigned char *p)
- static void `ProtocolPP::U64TO8` (unsigned char *p, unsigned long long v)
- static void `ProtocolPP::jpoly1305_blocks` (jpoly1305_state_internal_t *st, const unsigned char *m, size_t bytes)

8.53.1 Macro Definition Documentation

8.53.1.1 `#define jpoly1305_block_size 16`

8.54 include/poly1305-8.h File Reference

Classes

- struct `ProtocolPP::jpoly1305_state_internal_t`

Namespaces

- `ProtocolPP`

Macros

- `#define JPOLY1305_NOINLINE`
- `#define jpoly1305_block_size 16`

8.54.1 Macro Definition Documentation

8.54.1.1 `#define jpoly1305_block_size 16`

8.54.1.2 `#define JPOLY1305_NOINLINE`

8.55 include/SFMT-alti.h File Reference

SIMD oriented Fast Mersenne Twister(SFMT) pseudorandom number generator.

Namespaces

- `ProtocolPP`

Macros

- `#define SFMT_ALTI_H`
- `#define SFMT_ALTI_SWAP {4, 5, 6, 7, 0, 1, 2, 3, 12, 13, 14, 15, 8, 9, 10, 11}`

Functions

- static vector unsigned int `ProtocolPP::vec_recursion` (vector unsigned int a, vector unsigned int b, vector unsigned int c, vector unsigned int d)
- void `ProtocolPP::sfmt_gen_rand_all ()`

- static void [ProtocolPP::gen_rand_array](#) (w128_t *array, int size)
- static void [ProtocolPP::swap](#) (w128_t *array, int size)

8.55.1 Detailed Description

SIMD oriented Fast Mersenne Twister(SFMT) pseudorandom number generator.

Author

Mutsuo Saito (Hiroshima University)
Makoto Matsumoto (Hiroshima University)

Copyright (C) 2007 Mutsuo Saito, Makoto Matsumoto and Hiroshima University. All rights reserved.

The new BSD License is applied to this software. see LICENSE.txt

8.55.2 Macro Definition Documentation

8.55.2.1 #define SFMT_ALTI_H

8.55.2.2 #define SFMT_ALTI_SWAP {4, 5, 6, 7, 0, 1, 2, 3, 12, 13, 14, 15, 8, 9, 10, 11}

8.56 include/SFMT-common.h File Reference

SIMD oriented Fast Mersenne Twister(SFMT) pseudorandom number generator with jump function. This file includes common functions used in random number generation and jump.

```
#include "SFMT.h"
```

Namespaces

- [ProtocolPP](#)

Macros

- #define [SFMT_COMMON_H](#)

Functions

- static void [ProtocolPP::do_recursion](#) (w128_t *r, w128_t *a, w128_t *b, w128_t *c, w128_t *d)
- static void [ProtocolPP::rshift128](#) (w128_t *out, w128_t const *in, int shift)
- static void [ProtocolPP::lshift128](#) (w128_t *out, w128_t const *in, int shift)

8.56.1 Detailed Description

SIMD oriented Fast Mersenne Twister(SFMT) pseudorandom number generator with jump function. This file includes common functions used in random number generation and jump.

Author

Mutsuo Saito (Hiroshima University)
Makoto Matsumoto (The University of Tokyo)

Copyright (C) 2006, 2007 Mutsuo Saito, Makoto Matsumoto and Hiroshima University. Copyright (C) 2012 Mutsuo Saito, Makoto Matsumoto, Hiroshima University and The University of Tokyo. All rights reserved.

The 3-clause BSD License is applied to this software, see LICENSE.txt

8.56.2 Macro Definition Documentation

8.56.2.1 #define SFMT_COMMON_H

8.57 include/SFMT-neon.h File Reference

SIMD oriented Fast Mersenne Twister(SFMT) for ARM with 128b NEON.

Namespaces

- [ProtocolPP](#)

Macros

- `#define rotate_bytes(A, B, C) vreinterpretq_u32_u8(vextq_u8(vreinterpretq_u8_u32(A),vreinterpretq_u8_u32(B),(C)))`

Functions

- `static void ProtocolPP::neon_recursion (uint32x4_t *r, uint32x4_t a, uint32x4_t b, uint32x4_t c, uint32x4_t d)`
- `void ProtocolPP::sfmt_gen_rand_all ()`
- `static void ProtocolPP::gen_rand_array (w128_t *array, int size)`

8.57.1 Detailed Description

SIMD oriented Fast Mersenne Twister(SFMT) for ARM with 128b NEON.

Author

Masaki Ota

Note

We assume LITTLE ENDIAN in this file

8.57.2 Macro Definition Documentation

8.57.2.1 #define rotate_bytes(A, B, C) vreinterpretq_u32_u8(vextq_u8(vreinterpretq_u8_u32(A),vreinterpretq_u8_u32(B),(C)))

8.58 include/SFMT-params.h File Reference

```
#include "SFMT-params19937.h"
```

Namespaces

- [ProtocolPP](#)

Macros

- `#define SFMT_PARAMS_H`
- `#define SFMT_MEXP 19937`
- `#define SFMT_N (SFMT_MEXP / 128 + 1)`
- `#define SFMT_N8 (SFMT_N * 16)`
- `#define SFMT_N16 (SFMT_N * 8)`
- `#define SFMT_N32 (SFMT_N * 4)`
- `#define SFMT_N64 (SFMT_N * 2)`

8.58.1 Macro Definition Documentation

8.58.1.1 `#define SFMT_MEXP 19937`

8.58.1.2 `#define SFMT_N (SFMT_MEXP / 128 + 1)`

Mersenne Exponent. The period of the sequence is a multiple of $2^{\text{MEXP}} - 1$. `#define SFMT_MEXP 19937` SFMT generator has an internal state array of 128-bit integers, and N is its size.

8.58.1.3 `#define SFMT_N16 (SFMT_N * 8)`

N16 is the size of internal state array when regarded as an array of 16-bit integers.

8.58.1.4 `#define SFMT_N32 (SFMT_N * 4)`

N32 is the size of internal state array when regarded as an array of 32-bit integers.

8.58.1.5 `#define SFMT_N64 (SFMT_N * 2)`

N64 is the size of internal state array when regarded as an array of 64-bit integers.

8.58.1.6 `#define SFMT_N8 (SFMT_N * 16)`

N8 is the size of internal state array when regarded as an array of 8-bit integers.

8.58.1.7 `#define SFMT_PARAMS_H`

8.59 include/SFMT-params11213.h File Reference

Namespaces

- [ProtocolPP](#)

Macros

- #define SFMT_PARAMS11213_H
- #define SFMT_POS1 68
- #define SFMT_SL1 14
- #define SFMT_SL2 3
- #define SFMT_SR1 7
- #define SFMT_SR2 3
- #define SFMT_MSK1 0xfffff7fbU
- #define SFMT_MSK2 0xffffffffU
- #define SFMT_MSK3 0xdffffbfffU
- #define SFMT_MSK4 0x7ffffdbfdU
- #define SFMT_PARITY1 0x00000001U
- #define SFMT_PARITY2 0x00000000U
- #define SFMT_PARITY3 0xe8148000U
- #define SFMT_PARITY4 0xd0c7afa3U
- #define SFMT_ALTI_SL1 {SFMT_SL1, SFMT_SL1, SFMT_SL1, SFMT_SL1}
- #define SFMT_ALTI_SR1 {SFMT_SR1, SFMT_SR1, SFMT_SR1, SFMT_SR1}
- #define SFMT_ALTI_MSK {SFMT_MSK1, SFMT_MSK2, SFMT_MSK3, SFMT_MSK4}
- #define SFMT_ALTI_MSK64 {SFMT_MSK2, SFMT_MSK1, SFMT_MSK4, SFMT_MSK3}
- #define SFMT_ALTI_SL2_PERM {3,21,21,21,7,0,1,2,11,4,5,6,15,8,9,10}
- #define SFMT_ALTI_SL2_PERM64 {3,4,5,6,7,29,29,29,11,12,13,14,15,0,1,2}
- #define SFMT_ALTI_SR2_PERM {5,6,7,0,9,10,11,4,13,14,15,8,19,19,19,12}
- #define SFMT_ALTI_SR2_PERM64 {13,14,15,0,1,2,3,4,19,19,19,8,9,10,11,12}
- #define SFMT_IDSTR "SFMT-11213:68-14-3-7-3:fffff7fb-fffffeef-dfdfbfff-7ffffdbfd"

8.59.1 Macro Definition Documentation

- 8.59.1.1 #define SFMT_ALTI_MSK {SFMT_MSK1, SFMT_MSK2, SFMT_MSK3, SFMT_MSK4}
- 8.59.1.2 #define SFMT_ALTI_MSK64 {SFMT_MSK2, SFMT_MSK1, SFMT_MSK4, SFMT_MSK3}
- 8.59.1.3 #define SFMT_ALTI_SL1 {SFMT_SL1, SFMT_SL1, SFMT_SL1, SFMT_SL1}
- 8.59.1.4 #define SFMT_ALTI_SL2_PERM {3,21,21,21,7,0,1,2,11,4,5,6,15,8,9,10}
- 8.59.1.5 #define SFMT_ALTI_SL2_PERM64 {3,4,5,6,7,29,29,29,11,12,13,14,15,0,1,2}
- 8.59.1.6 #define SFMT_ALTI_SR1 {SFMT_SR1, SFMT_SR1, SFMT_SR1, SFMT_SR1}
- 8.59.1.7 #define SFMT_ALTI_SR2_PERM {5,6,7,0,9,10,11,4,13,14,15,8,19,19,19,12}
- 8.59.1.8 #define SFMT_ALTI_SR2_PERM64 {13,14,15,0,1,2,3,4,19,19,19,8,9,10,11,12}
- 8.59.1.9 #define SFMT_IDSTR "SFMT-11213:68-14-3-7-3:fffff7fb-fffffeef-dfdfbfff-7ffffdbfd"
- 8.59.1.10 #define SFMT_MSK1 0xfffff7fbU
- 8.59.1.11 #define SFMT_MSK2 0xffffffffU
- 8.59.1.12 #define SFMT_MSK3 0xdffffbfffU
- 8.59.1.13 #define SFMT_MSK4 0x7ffffdbfdU
- 8.59.1.14 #define SFMT_PARAMS11213_H

8.59.1.15 #define SFMT_PARITY1 0x00000001U

8.59.1.16 #define SFMT_PARITY2 0x00000000U

8.59.1.17 #define SFMT_PARITY3 0xe8148000U

8.59.1.18 #define SFMT_PARITY4 0xd0c7afa3U

8.59.1.19 #define SFMT_POS1 68

8.59.1.20 #define SFMT_SL1 14

8.59.1.21 #define SFMT_SL2 3

8.59.1.22 #define SFMT_SR1 7

8.59.1.23 #define SFMT_SR2 3

8.60 include/SFMT-params1279.h File Reference

Namespaces

- [ProtocolPP](#)

Macros

- #define SFMT_PARAMS1279_H
- #define SFMT_POS1 7
- #define SFMT_SL1 14
- #define SFMT_SL2 3
- #define SFMT_SR1 5
- #define SFMT_SR2 1
- #define SFMT_MSK1 0xf7feffffU
- #define SFMT_MSK2 0x7fefcffffU
- #define SFMT_MSK3 0xaff3ef3fU
- #define SFMT_MSK4 0xb5ffff7fU
- #define SFMT_PARITY1 0x00000001U
- #define SFMT_PARITY2 0x00000000U
- #define SFMT_PARITY3 0x00000000U
- #define SFMT_PARITY4 0x20000000U
- #define SFMT_ALTI_SL1 {SFMT_SL1, SFMT_SL1, SFMT_SL1, SFMT_SL1}
- #define SFMT_ALTI_SR1 {SFMT_SR1, SFMT_SR1, SFMT_SR1, SFMT_SR1}
- #define SFMT_ALTI_MSK {SFMT_MSK1, SFMT_MSK2, SFMT_MSK3, SFMT_MSK4}
- #define SFMT_ALTI_MSK64 {SFMT_MSK2, SFMT_MSK1, SFMT_MSK4, SFMT_MSK3}
- #define SFMT_ALTI_SL2_PERM {3,21,21,21,7,0,1,2,11,4,5,6,15,8,9,10}
- #define SFMT_ALTI_SL2_PERM64 {3,4,5,6,7,29,29,29,11,12,13,14,15,0,1,2}
- #define SFMT_ALTI_SR2_PERM {7,0,1,2,11,4,5,6,15,8,9,10,17,12,13,14}
- #define SFMT_ALTI_SR2_PERM64 {15,0,1,2,3,4,5,6,17,8,9,10,11,12,13,14}
- #define SFMT_IDSTR "SFMT-1279:7-14-3-5-1:f7feffff-7fefcffff-aff3ef3f-b5ffff7f"

8.60.1 Macro Definition Documentation

```
8.60.1.1 #define SFMT_ALTI_MSK {SFMT_MSK1, SFMT_MSK2, SFMT_MSK3, SFMT_MSK4}

8.60.1.2 #define SFMT_ALTI_MSK64 {SFMT_MSK2, SFMT_MSK1, SFMT_MSK4, SFMT_MSK3}

8.60.1.3 #define SFMT_ALTI_SL1 {SFMT_SL1, SFMT_SL1, SFMT_SL1, SFMT_SL1}

8.60.1.4 #define SFMT_ALTI_SL2_PERM {3,21,21,21,7,0,1,2,11,4,5,6,15,8,9,10}

8.60.1.5 #define SFMT_ALTI_SL2_PERM64 {3,4,5,6,7,29,29,29,11,12,13,14,15,0,1,2}

8.60.1.6 #define SFMT_ALTI_SR1 {SFMT_SR1, SFMT_SR1, SFMT_SR1, SFMT_SR1}

8.60.1.7 #define SFMT_ALTI_SR2_PERM {7,0,1,2,11,4,5,6,15,8,9,10,17,12,13,14}

8.60.1.8 #define SFMT_ALTI_SR2_PERM64 {15,0,1,2,3,4,5,6,17,8,9,10,11,12,13,14}

8.60.1.9 #define SFMT_IDSTR "SFMT-1279:7-14-3-5-1:f7feffffd-7fefcffff-aff3ef3f-b5ffff7f"

8.60.1.10 #define SFMT_MSK1 0xf7feffffU

8.60.1.11 #define SFMT_MSK2 0x7fefcffffU

8.60.1.12 #define SFMT_MSK3 0xaaff3ef3fU

8.60.1.13 #define SFMT_MSK4 0xb5ffff7fU

8.60.1.14 #define SFMT_PARAMS1279_H

8.60.1.15 #define SFMT_PARITY1 0x00000001U

8.60.1.16 #define SFMT_PARITY2 0x00000000U

8.60.1.17 #define SFMT_PARITY3 0x00000000U

8.60.1.18 #define SFMT_PARITY4 0x20000000U

8.60.1.19 #define SFMT_POS1 7

8.60.1.20 #define SFMT_SL1 14

8.60.1.21 #define SFMT_SL2 3

8.60.1.22 #define SFMT_SR1 5

8.60.1.23 #define SFMT_SR2 1
```

8.61 include/SFMT-params132049.h File Reference

Namespaces

- [ProtocolPP](#)

Macros

- #define SFMT_PARAMS132049_H
- #define SFMT_POS1 110
- #define SFMT_SL1 19
- #define SFMT_SL2 1
- #define SFMT_SR1 21
- #define SFMT_SR2 1
- #define SFMT_MSK1 0xffffbb5fU
- #define SFMT_MSK2 0xfb6ebf95U
- #define SFMT_MSK3 0xffffefffaU
- #define SFMT_MSK4 0xcff77fffU
- #define SFMT_PARITY1 0x00000001U
- #define SFMT_PARITY2 0x00000000U
- #define SFMT_PARITY3 0xcb520000U
- #define SFMT_PARITY4 0xc7e91c7dU
- #define SFMT_ALTI_SL1 {SFMT_SL1, SFMT_SL1, SFMT_SL1, SFMT_SL1}
- #define SFMT_ALTI_SR1 {SFMT_SR1, SFMT_SR1, SFMT_SR1, SFMT_SR1}
- #define SFMT_ALTI_MSK {SFMT_MSK1, SFMT_MSK2, SFMT_MSK3, SFMT_MSK4}
- #define SFMT_ALTI_MSK64 {SFMT_MSK2, SFMT_MSK1, SFMT_MSK4, SFMT_MSK3}
- #define SFMT_ALTI_SL2_PERM {1,2,3,23,5,6,7,0,9,10,11,4,13,14,15,8}
- #define SFMT_ALTI_SL2_PERM64 {1,2,3,4,5,6,7,31,9,10,11,12,13,14,15,0}
- #define SFMT_ALTI_SR2_PERM {7,0,1,2,11,4,5,6,15,8,9,10,17,12,13,14}
- #define SFMT_ALTI_SR2_PERM64 {15,0,1,2,3,4,5,6,17,8,9,10,11,12,13,14}
- #define SFMT_IDSTR "SFMT-132049:110-19-1-21-1:ffffbb5f-fb6ebf95-ffffefffa-cff77fff"

8.61.1 Macro Definition Documentation

- 8.61.1.1 #define SFMT_ALTI_MSK {SFMT_MSK1, SFMT_MSK2, SFMT_MSK3, SFMT_MSK4}
- 8.61.1.2 #define SFMT_ALTI_MSK64 {SFMT_MSK2, SFMT_MSK1, SFMT_MSK4, SFMT_MSK3}
- 8.61.1.3 #define SFMT_ALTI_SL1 {SFMT_SL1, SFMT_SL1, SFMT_SL1, SFMT_SL1}
- 8.61.1.4 #define SFMT_ALTI_SL2_PERM {1,2,3,23,5,6,7,0,9,10,11,4,13,14,15,8}
- 8.61.1.5 #define SFMT_ALTI_SL2_PERM64 {1,2,3,4,5,6,7,31,9,10,11,12,13,14,15,0}
- 8.61.1.6 #define SFMT_ALTI_SR1 {SFMT_SR1, SFMT_SR1, SFMT_SR1, SFMT_SR1}
- 8.61.1.7 #define SFMT_ALTI_SR2_PERM {7,0,1,2,11,4,5,6,15,8,9,10,17,12,13,14}
- 8.61.1.8 #define SFMT_ALTI_SR2_PERM64 {15,0,1,2,3,4,5,6,17,8,9,10,11,12,13,14}
- 8.61.1.9 #define SFMT_IDSTR "SFMT-132049:110-19-1-21-1:ffffbb5f-fb6ebf95-ffffefffa-cff77fff"
- 8.61.1.10 #define SFMT_MSK1 0xffffbb5fU
- 8.61.1.11 #define SFMT_MSK2 0xfb6ebf95U
- 8.61.1.12 #define SFMT_MSK3 0xffffefffaU
- 8.61.1.13 #define SFMT_MSK4 0xcff77fffU
- 8.61.1.14 #define SFMT_PARAMS132049_H

```
8.61.1.15 #define SFMT_PARITY1 0x00000001U  
8.61.1.16 #define SFMT_PARITY2 0x00000000U  
8.61.1.17 #define SFMT_PARITY3 0xcb520000U  
8.61.1.18 #define SFMT_PARITY4 0xc7e91c7dU  
8.61.1.19 #define SFMT_POS1 110  
8.61.1.20 #define SFMT_SL1 19  
8.61.1.21 #define SFMT_SL2 1  
8.61.1.22 #define SFMT_SR1 21  
8.61.1.23 #define SFMT_SR2 1
```

8.62 include/SFMT-params19937.h File Reference

Namespaces

- [ProtocolPP](#)

Macros

- [#define SFMT_PARAMS19937_H](#)
- [#define SFMT_POS1 122](#)
- [#define SFMT_SL1 18](#)
- [#define SFMT_SL2 1](#)
- [#define SFMT_SR1 11](#)
- [#define SFMT_SR2 1](#)
- [#define SFMT_MSK1 0xdfffffefU](#)
- [#define SFMT_MSK2 0xddfecb7fU](#)
- [#define SFMT_MSK3 0xbfffafffU](#)
- [#define SFMT_MSK4 0xbfffff6U](#)
- [#define SFMT_PARITY1 0x00000001U](#)
- [#define SFMT_PARITY2 0x00000000U](#)
- [#define SFMT_PARITY3 0x00000000U](#)
- [#define SFMT_PARITY4 0x13c9e684U](#)
- [#define SFMT_ALTI_SL1 {SFMT_SL1, SFMT_SL1, SFMT_SL1, SFMT_SL1}](#)
- [#define SFMT_ALTI_SR1 {SFMT_SR1, SFMT_SR1, SFMT_SR1, SFMT_SR1}](#)
- [#define SFMT_ALTI_MSK {SFMT_MSK1, SFMT_MSK2, SFMT_MSK3, SFMT_MSK4}](#)
- [#define SFMT_ALTI_MSK64 {SFMT_MSK2, SFMT_MSK1, SFMT_MSK4, SFMT_MSK3}](#)
- [#define SFMT_ALTI_SL2_PERM {1,2,3,23,5,6,7,0,9,10,11,4,13,14,15,8}](#)
- [#define SFMT_ALTI_SL2_PERM64 {1,2,3,4,5,6,7,31,9,10,11,12,13,14,15,0}](#)
- [#define SFMT_ALTI_SR2_PERM {7,0,1,2,11,4,5,6,15,8,9,10,17,12,13,14}](#)
- [#define SFMT_ALTI_SR2_PERM64 {15,0,1,2,3,4,5,6,17,8,9,10,11,12,13,14}](#)
- [#define SFMT_IDSTR "SFMT-19937:122-18-1-11-1:dfffffef-ddfecb7f-bfffafff-bfffff6"](#)

8.62.1 Macro Definition Documentation

```
8.62.1.1 #define SFMT_ALTI_MSK {SFMT_MSK1, SFMT_MSK2, SFMT_MSK3, SFMT_MSK4}

8.62.1.2 #define SFMT_ALTI_MSK64 {SFMT_MSK2, SFMT_MSK1, SFMT_MSK4, SFMT_MSK3}

8.62.1.3 #define SFMT_ALTI_SL1 {SFMT_SL1, SFMT_SL1, SFMT_SL1, SFMT_SL1}

8.62.1.4 #define SFMT_ALTI_SL2_PERM {1,2,3,23,5,6,7,0,9,10,11,4,13,14,15,8}

8.62.1.5 #define SFMT_ALTI_SL2_PERM64 {1,2,3,4,5,6,7,31,9,10,11,12,13,14,15,0}

8.62.1.6 #define SFMT_ALTI_SR1 {SFMT_SR1, SFMT_SR1, SFMT_SR1, SFMT_SR1}

8.62.1.7 #define SFMT_ALTI_SR2_PERM {7,0,1,2,11,4,5,6,15,8,9,10,17,12,13,14}

8.62.1.8 #define SFMT_ALTI_SR2_PERM64 {15,0,1,2,3,4,5,6,17,8,9,10,11,12,13,14}

8.62.1.9 #define SFMT_IDSTR "SFMT-19937:122-18-1-11-1:dfffffef-ddfecb7f-bfffafff-bfffff6"

8.62.1.10 #define SFMT_MSK1 0xdfffffefU

8.62.1.11 #define SFMT_MSK2 0xddfecb7fU

8.62.1.12 #define SFMT_MSK3 0xbfffffffU

8.62.1.13 #define SFMT_MSK4 0xbfffffff6U

8.62.1.14 #define SFMT_PARAMS19937_H

8.62.1.15 #define SFMT_PARITY1 0x00000001U

8.62.1.16 #define SFMT_PARITY2 0x00000000U

8.62.1.17 #define SFMT_PARITY3 0x00000000U

8.62.1.18 #define SFMT_PARITY4 0x13c9e684U

8.62.1.19 #define SFMT_POS1 122

8.62.1.20 #define SFMT_SL1 18

8.62.1.21 #define SFMT_SL2 1

8.62.1.22 #define SFMT_SR1 11

8.62.1.23 #define SFMT_SR2 1
```

8.63 include/SFMT-params216091.h File Reference

Namespaces

- [ProtocolPP](#)

Macros

- #define SFMT_PARAMS216091_H
- #define SFMT_POS1 627
- #define SFMT_SL1 11
- #define SFMT_SL2 3
- #define SFMT_SR1 10
- #define SFMT_SR2 1
- #define SFMT_MSK1 0xbff7bff7U
- #define SFMT_MSK2 0xffffffffU
- #define SFMT_MSK3 0xbfffffa7fU
- #define SFMT_MSK4 0xffddfbfbU
- #define SFMT_PARITY1 0xf8000001U
- #define SFMT_PARITY2 0x89e80709U
- #define SFMT_PARITY3 0x3bd2b64bU
- #define SFMT_PARITY4 0x0c64b1e4U
- #define SFMT_ALTI_SL1 {SFMT_SL1, SFMT_SL1, SFMT_SL1, SFMT_SL1}
- #define SFMT_ALTI_SR1 {SFMT_SR1, SFMT_SR1, SFMT_SR1, SFMT_SR1}
- #define SFMT_ALTI_MSK {SFMT_MSK1, SFMT_MSK2, SFMT_MSK3, SFMT_MSK4}
- #define SFMT_ALTI_MSK64 {SFMT_MSK2, SFMT_MSK1, SFMT_MSK4, SFMT_MSK3}
- #define SFMT_ALTI_SL2_PERM {3,21,21,21,7,0,1,2,11,4,5,6,15,8,9,10}
- #define SFMT_ALTI_SL2_PERM64 {3,4,5,6,7,29,29,29,11,12,13,14,15,0,1,2}
- #define SFMT_ALTI_SR2_PERM {7,0,1,2,11,4,5,6,15,8,9,10,17,12,13,14}
- #define SFMT_ALTI_SR2_PERM64 {15,0,1,2,3,4,5,6,17,8,9,10,11,12,13,14}
- #define SFMT_IDSTR "SFMT-216091:627-11-3-10-1:bff7bff7-bfffffff-bfffffa7f-ffddfbfb"

8.63.1 Macro Definition Documentation

- 8.63.1.1 #define SFMT_ALTI_MSK {SFMT_MSK1, SFMT_MSK2, SFMT_MSK3, SFMT_MSK4}
- 8.63.1.2 #define SFMT_ALTI_MSK64 {SFMT_MSK2, SFMT_MSK1, SFMT_MSK4, SFMT_MSK3}
- 8.63.1.3 #define SFMT_ALTI_SL1 {SFMT_SL1, SFMT_SL1, SFMT_SL1, SFMT_SL1}
- 8.63.1.4 #define SFMT_ALTI_SL2_PERM {3,21,21,21,7,0,1,2,11,4,5,6,15,8,9,10}
- 8.63.1.5 #define SFMT_ALTI_SL2_PERM64 {3,4,5,6,7,29,29,29,11,12,13,14,15,0,1,2}
- 8.63.1.6 #define SFMT_ALTI_SR1 {SFMT_SR1, SFMT_SR1, SFMT_SR1, SFMT_SR1}
- 8.63.1.7 #define SFMT_ALTI_SR2_PERM {7,0,1,2,11,4,5,6,15,8,9,10,17,12,13,14}
- 8.63.1.8 #define SFMT_ALTI_SR2_PERM64 {15,0,1,2,3,4,5,6,17,8,9,10,11,12,13,14}
- 8.63.1.9 #define SFMT_IDSTR "SFMT-216091:627-11-3-10-1:bff7bff7-bfffffff-bfffffa7f-ffddfbfb"
- 8.63.1.10 #define SFMT_MSK1 0xbff7bff7U
- 8.63.1.11 #define SFMT_MSK2 0xffffffffU
- 8.63.1.12 #define SFMT_MSK3 0xbfffffa7fU
- 8.63.1.13 #define SFMT_MSK4 0xffddfbfbU
- 8.63.1.14 #define SFMT_PARAMS216091_H

```
8.63.1.15 #define SFMT_PARITY1 0xf8000001U  
8.63.1.16 #define SFMT_PARITY2 0x89e80709U  
8.63.1.17 #define SFMT_PARITY3 0x3bd2b64bU  
8.63.1.18 #define SFMT_PARITY4 0x0c64b1e4U  
8.63.1.19 #define SFMT_POS1 627  
8.63.1.20 #define SFMT_SL1 11  
8.63.1.21 #define SFMT_SL2 3  
8.63.1.22 #define SFMT_SR1 10  
8.63.1.23 #define SFMT_SR2 1
```

8.64 include/SFMT-params2281.h File Reference

Namespaces

- [ProtocolPP](#)

Macros

- `#define SFMT_PARAMS2281_H`
- `#define SFMT_POS1 12`
- `#define SFMT_SL1 19`
- `#define SFMT_SL2 1`
- `#define SFMT_SR1 5`
- `#define SFMT_SR2 1`
- `#define SFMT_MSK1 0xbff7ffbfU`
- `#define SFMT_MSK2 0xfdfffffeU`
- `#define SFMT_MSK3 0xf7fef7fU`
- `#define SFMT_MSK4 0xf2f7cbbfU`
- `#define SFMT_PARITY1 0x00000001U`
- `#define SFMT_PARITY2 0x00000000U`
- `#define SFMT_PARITY3 0x00000000U`
- `#define SFMT_PARITY4 0x41dfa600U`
- `#define SFMT_ALTI_SL1 {SFMT_SL1, SFMT_SL1, SFMT_SL1, SFMT_SL1}`
- `#define SFMT_ALTI_SR1 {SFMT_SR1, SFMT_SR1, SFMT_SR1, SFMT_SR1}`
- `#define SFMT_ALTI_MSK {SFMT_MSK1, SFMT_MSK2, SFMT_MSK3, SFMT_MSK4}`
- `#define SFMT_ALTI_MSK64 {SFMT_MSK2, SFMT_MSK1, SFMT_MSK4, SFMT_MSK3}`
- `#define SFMT_ALTI_SL2_PERM {1,2,3,23,5,6,7,0,9,10,11,4,13,14,15,8}`
- `#define SFMT_ALTI_SL2_PERM64 {1,2,3,4,5,6,7,31,9,10,11,12,13,14,15,0}`
- `#define SFMT_ALTI_SR2_PERM {7,0,1,2,11,4,5,6,15,8,9,10,17,12,13,14}`
- `#define SFMT_ALTI_SR2_PERM64 {15,0,1,2,3,4,5,6,17,8,9,10,11,12,13,14}`
- `#define SFMT_IDSTR "SFMT-2281:12-19-1-5-1:bff7ffbf-fdfffffe-f7fef7f-f2f7cbbf"`

8.64.1 Macro Definition Documentation

```
8.64.1.1 #define SFMT_ALTI_MSK {SFMT_MSK1, SFMT_MSK2, SFMT_MSK3, SFMT_MSK4}

8.64.1.2 #define SFMT_ALTI_MSK64 {SFMT_MSK2, SFMT_MSK1, SFMT_MSK4, SFMT_MSK3}

8.64.1.3 #define SFMT_ALTI_SL1 {SFMT_SL1, SFMT_SL1, SFMT_SL1, SFMT_SL1}

8.64.1.4 #define SFMT_ALTI_SL2_PERM {1,2,3,23,5,6,7,0,9,10,11,4,13,14,15,8}

8.64.1.5 #define SFMT_ALTI_SL2_PERM64 {1,2,3,4,5,6,7,31,9,10,11,12,13,14,15,0}

8.64.1.6 #define SFMT_ALTI_SR1 {SFMT_SR1, SFMT_SR1, SFMT_SR1, SFMT_SR1}

8.64.1.7 #define SFMT_ALTI_SR2_PERM {7,0,1,2,11,4,5,6,15,8,9,10,17,12,13,14}

8.64.1.8 #define SFMT_ALTI_SR2_PERM64 {15,0,1,2,3,4,5,6,17,8,9,10,11,12,13,14}

8.64.1.9 #define SFMT_IDSTR "SFMT-2281:12-19-1-5-1:bff7ffbf-fdfffffe-f7ffef7f-f2f7cbbf"

8.64.1.10 #define SFMT_MSK1 0xbff7ffbfU

8.64.1.11 #define SFMT_MSK2 0xfdfffffeU

8.64.1.12 #define SFMT_MSK3 0xf7ffef7fU

8.64.1.13 #define SFMT_MSK4 0xf2f7cbbfU

8.64.1.14 #define SFMT_PARAMS2281_H

8.64.1.15 #define SFMT_PARITY1 0x00000001U

8.64.1.16 #define SFMT_PARITY2 0x00000000U

8.64.1.17 #define SFMT_PARITY3 0x00000000U

8.64.1.18 #define SFMT_PARITY4 0x41dfa600U

8.64.1.19 #define SFMT_POS1 12

8.64.1.20 #define SFMT_SL1 19

8.64.1.21 #define SFMT_SL2 1

8.64.1.22 #define SFMT_SR1 5

8.64.1.23 #define SFMT_SR2 1
```

8.65 include/SFMT-params4253.h File Reference

Namespaces

- [ProtocolPP](#)

Macros

- `#define SFMT_PARAMS4253_H`
- `#define SFMT_POS1 17`
- `#define SFMT_SL1 20`
- `#define SFMT_SL2 1`
- `#define SFMT_SR1 7`
- `#define SFMT_SR2 1`
- `#define SFMT_MSK1 0x9f7bffffU`
- `#define SFMT_MSK2 0x9fffff5fU`
- `#define SFMT_MSK3 0x3efffffbU`
- `#define SFMT_MSK4 0xfffff7bbU`
- `#define SFMT_PARITY1 0xa8000001U`
- `#define SFMT_PARITY2 0xaf5390a3U`
- `#define SFMT_PARITY3 0xb740b3f8U`
- `#define SFMT_PARITY4 0x6c11486dU`
- `#define SFMT_ALTI_SL1 {SFMT_SL1, SFMT_SL1, SFMT_SL1, SFMT_SL1}`
- `#define SFMT_ALTI_SR1 {SFMT_SR1, SFMT_SR1, SFMT_SR1, SFMT_SR1}`
- `#define SFMT_ALTI_MSK {SFMT_MSK1, SFMT_MSK2, SFMT_MSK3, SFMT_MSK4}`
- `#define SFMT_ALTI_MSK64 {SFMT_MSK2, SFMT_MSK1, SFMT_MSK4, SFMT_MSK3}`
- `#define SFMT_ALTI_SL2_PERM {1,2,3,23,5,6,7,0,9,10,11,4,13,14,15,8}`
- `#define SFMT_ALTI_SL2_PERM64 {1,2,3,4,5,6,7,31,9,10,11,12,13,14,15,0}`
- `#define SFMT_ALTI_SR2_PERM {7,0,1,2,11,4,5,6,15,8,9,10,17,12,13,14}`
- `#define SFMT_ALTI_SR2_PERM64 {15,0,1,2,3,4,5,6,17,8,9,10,11,12,13,14}`
- `#define SFMT_IDSTR "SFMT-4253:17-20-1-7-1:9f7bffff-9fffff5f-3efffffb-fffff7bb"`

8.65.1 Macro Definition Documentation

8.65.1.1 `#define SFMT_ALTI_MSK {SFMT_MSK1, SFMT_MSK2, SFMT_MSK3, SFMT_MSK4}`

8.65.1.2 `#define SFMT_ALTI_MSK64 {SFMT_MSK2, SFMT_MSK1, SFMT_MSK4, SFMT_MSK3}`

8.65.1.3 `#define SFMT_ALTI_SL1 {SFMT_SL1, SFMT_SL1, SFMT_SL1, SFMT_SL1}`

8.65.1.4 `#define SFMT_ALTI_SL2_PERM {1,2,3,23,5,6,7,0,9,10,11,4,13,14,15,8}`

8.65.1.5 `#define SFMT_ALTI_SL2_PERM64 {1,2,3,4,5,6,7,31,9,10,11,12,13,14,15,0}`

8.65.1.6 `#define SFMT_ALTI_SR1 {SFMT_SR1, SFMT_SR1, SFMT_SR1, SFMT_SR1}`

8.65.1.7 `#define SFMT_ALTI_SR2_PERM {7,0,1,2,11,4,5,6,15,8,9,10,17,12,13,14}`

8.65.1.8 `#define SFMT_ALTI_SR2_PERM64 {15,0,1,2,3,4,5,6,17,8,9,10,11,12,13,14}`

8.65.1.9 `#define SFMT_IDSTR "SFMT-4253:17-20-1-7-1:9f7bffff-9fffff5f-3efffffb-fffff7bb"`

8.65.1.10 `#define SFMT_MSK1 0x9f7bffffU`

8.65.1.11 `#define SFMT_MSK2 0x9fffff5fU`

8.65.1.12 `#define SFMT_MSK3 0x3efffffbU`

8.65.1.13 `#define SFMT_MSK4 0xfffff7bbU`

8.65.1.14 `#define SFMT_PARAMS4253_H`

```
8.65.1.15 #define SFMT_PARITY1 0xa8000001U  
8.65.1.16 #define SFMT_PARITY2 0xaf5390a3U  
8.65.1.17 #define SFMT_PARITY3 0xb740b3f8U  
8.65.1.18 #define SFMT_PARITY4 0x6c11486dU  
8.65.1.19 #define SFMT_POS1 17  
8.65.1.20 #define SFMT_SL1 20  
8.65.1.21 #define SFMT_SL2 1  
8.65.1.22 #define SFMT_SR1 7  
8.65.1.23 #define SFMT_SR2 1
```

8.66 include/SFMT-params44497.h File Reference

Namespaces

- [ProtocolPP](#)

Macros

- [#define SFMT_PARAMS44497_H](#)
- [#define SFMT_POS1 330](#)
- [#define SFMT_SL1 5](#)
- [#define SFMT_SL2 3](#)
- [#define SFMT_SR1 9](#)
- [#define SFMT_SR2 3](#)
- [#define SFMT_MSK1 0xefffffffU](#)
- [#define SFMT_MSK2 0xdfbebfffU](#)
- [#define SFMT_MSK3 0xbfbf7befU](#)
- [#define SFMT_MSK4 0x9ffd7bffU](#)
- [#define SFMT_PARITY1 0x00000001U](#)
- [#define SFMT_PARITY2 0x00000000U](#)
- [#define SFMT_PARITY3 0xa3ac4000U](#)
- [#define SFMT_PARITY4 0xecc1327aU](#)
- [#define SFMT_ALTI_SL1 {SFMT_SL1, SFMT_SL1, SFMT_SL1, SFMT_SL1}](#)
- [#define SFMT_ALTI_SR1 {SFMT_SR1, SFMT_SR1, SFMT_SR1, SFMT_SR1}](#)
- [#define SFMT_ALTI_MSK {SFMT_MSK1, SFMT_MSK2, SFMT_MSK3, SFMT_MSK4}](#)
- [#define SFMT_ALTI_MSK64 {SFMT_MSK2, SFMT_MSK1, SFMT_MSK4, SFMT_MSK3}](#)
- [#define SFMT_ALTI_SL2_PERM {3,21,21,21,7,0,1,2,11,4,5,6,15,8,9,10}](#)
- [#define SFMT_ALTI_SL2_PERM64 {3,4,5,6,7,29,29,29,11,12,13,14,15,0,1,2}](#)
- [#define SFMT_ALTI_SR2_PERM {5,6,7,0,9,10,11,4,13,14,15,8,19,19,19,12}](#)
- [#define SFMT_ALTI_SR2_PERM64 {13,14,15,0,1,2,3,4,19,19,19,8,9,10,11,12}](#)
- [#define SFMT_IDSTR "SFMT-44497:330-5-3-9-3:effffffb-dfbefbe-bfbf7bef-9ffd7bff"](#)

8.66.1 Macro Definition Documentation

```
8.66.1.1 #define SFMT_ALTI_MSK {SFMT_MSK1, SFMT_MSK2, SFMT_MSK3, SFMT_MSK4}

8.66.1.2 #define SFMT_ALTI_MSK64 {SFMT_MSK2, SFMT_MSK1, SFMT_MSK4, SFMT_MSK3}

8.66.1.3 #define SFMT_ALTI_SL1 {SFMT_SL1, SFMT_SL1, SFMT_SL1, SFMT_SL1}

8.66.1.4 #define SFMT_ALTI_SL2_PERM {3,21,21,21,7,0,1,2,11,4,5,6,15,8,9,10}

8.66.1.5 #define SFMT_ALTI_SL2_PERM64 {3,4,5,6,7,29,29,29,11,12,13,14,15,0,1,2}

8.66.1.6 #define SFMT_ALTI_SR1 {SFMT_SR1, SFMT_SR1, SFMT_SR1, SFMT_SR1}

8.66.1.7 #define SFMT_ALTI_SR2_PERM {5,6,7,0,9,10,11,4,13,14,15,8,19,19,19,12}

8.66.1.8 #define SFMT_ALTI_SR2_PERM64 {13,14,15,0,1,2,3,4,19,19,19,8,9,10,11,12}

8.66.1.9 #define SFMT_IDSTR "SFMT-44497:330-5-3-9-3:effffffb-dfbefbff-bfbf7bef-9ffd7bff"

8.66.1.10 #define SFMT_MSK1 0xffffffffU

8.66.1.11 #define SFMT_MSK2 0xdfbeffffU

8.66.1.12 #define SFMT_MSK3 0xbfbf7befU

8.66.1.13 #define SFMT_MSK4 0x9ffd7bffU

8.66.1.14 #define SFMT_PARAMS44497_H

8.66.1.15 #define SFMT_PARITY1 0x00000001U

8.66.1.16 #define SFMT_PARITY2 0x00000000U

8.66.1.17 #define SFMT_PARITY3 0xa3ac4000U

8.66.1.18 #define SFMT_PARITY4 0xeccc1327aU

8.66.1.19 #define SFMT_POS1 330

8.66.1.20 #define SFMT_SL1 5

8.66.1.21 #define SFMT_SL2 3

8.66.1.22 #define SFMT_SR1 9

8.66.1.23 #define SFMT_SR2 3
```

8.67 include/SFMT-params607.h File Reference

Namespaces

- [ProtocolPP](#)

Macros

- #define SFMT_PARAMS607_H
- #define SFMT_POS1 2
- #define SFMT_SL1 15
- #define SFMT_SL2 3
- #define SFMT_SR1 13
- #define SFMT_SR2 3
- #define SFMT_MSK1 0xfdff37ffU
- #define SFMT_MSK2 0xef7f3f7dU
- #define SFMT_MSK3 0xff777b7dU
- #define SFMT_MSK4 0x7ff7fb2fU
- #define SFMT_PARITY1 0x00000001U
- #define SFMT_PARITY2 0x00000000U
- #define SFMT_PARITY3 0x00000000U
- #define SFMT_PARITY4 0x5986f054U
- #define SFMT_ALTI_SL1 {SFMT_SL1, SFMT_SL1, SFMT_SL1, SFMT_SL1}
- #define SFMT_ALTI_SR1 {SFMT_SR1, SFMT_SR1, SFMT_SR1, SFMT_SR1}
- #define SFMT_ALTI_MSK {SFMT_MSK1, SFMT_MSK2, SFMT_MSK3, SFMT_MSK4}
- #define SFMT_ALTI_MSK64 {SFMT_MSK2, SFMT_MSK1, SFMT_MSK4, SFMT_MSK3}
- #define SFMT_ALTI_SL2_PERM {3,21,21,21,7,0,1,2,11,4,5,6,15,8,9,10}
- #define SFMT_ALTI_SL2_PERM64 {3,4,5,6,7,29,29,29,11,12,13,14,15,0,1,2}
- #define SFMT_ALTI_SR2_PERM {5,6,7,0,9,10,11,4,13,14,15,8,19,19,19,12}
- #define SFMT_ALTI_SR2_PERM64 {13,14,15,0,1,2,3,4,19,19,19,8,9,10,11,12}
- #define SFMT_IDSTR "SFMT-607:2-15-3-13-3:fdff37ff-ef7f3f7d-ff777b7d-7ff7fb2f"

8.67.1 Macro Definition Documentation

- 8.67.1.1 #define SFMT_ALTI_MSK {SFMT_MSK1, SFMT_MSK2, SFMT_MSK3, SFMT_MSK4}
- 8.67.1.2 #define SFMT_ALTI_MSK64 {SFMT_MSK2, SFMT_MSK1, SFMT_MSK4, SFMT_MSK3}
- 8.67.1.3 #define SFMT_ALTI_SL1 {SFMT_SL1, SFMT_SL1, SFMT_SL1, SFMT_SL1}
- 8.67.1.4 #define SFMT_ALTI_SL2_PERM {3,21,21,21,7,0,1,2,11,4,5,6,15,8,9,10}
- 8.67.1.5 #define SFMT_ALTI_SL2_PERM64 {3,4,5,6,7,29,29,29,11,12,13,14,15,0,1,2}
- 8.67.1.6 #define SFMT_ALTI_SR1 {SFMT_SR1, SFMT_SR1, SFMT_SR1, SFMT_SR1}
- 8.67.1.7 #define SFMT_ALTI_SR2_PERM {5,6,7,0,9,10,11,4,13,14,15,8,19,19,19,12}
- 8.67.1.8 #define SFMT_ALTI_SR2_PERM64 {13,14,15,0,1,2,3,4,19,19,19,8,9,10,11,12}
- 8.67.1.9 #define SFMT_IDSTR "SFMT-607:2-15-3-13-3:fdff37ff-ef7f3f7d-ff777b7d-7ff7fb2f"
- 8.67.1.10 #define SFMT_MSK1 0xfdff37ffU
- 8.67.1.11 #define SFMT_MSK2 0xef7f3f7dU
- 8.67.1.12 #define SFMT_MSK3 0xff777b7dU
- 8.67.1.13 #define SFMT_MSK4 0x7ff7fb2fU
- 8.67.1.14 #define SFMT_PARAMS607_H

8.67.1.15 #define SFMT_PARITY1 0x00000001U

8.67.1.16 #define SFMT_PARITY2 0x00000000U

8.67.1.17 #define SFMT_PARITY3 0x00000000U

8.67.1.18 #define SFMT_PARITY4 0x5986f054U

8.67.1.19 #define SFMT_POS1 2

8.67.1.20 #define SFMT_SL1 15

8.67.1.21 #define SFMT_SL2 3

8.67.1.22 #define SFMT_SR1 13

8.67.1.23 #define SFMT_SR2 3

8.68 include/SFMT-params86243.h File Reference

Namespaces

- [ProtocolPP](#)

Macros

- #define SFMT_PARAMS86243_H
- #define SFMT_POS1 366
- #define SFMT_SL1 6
- #define SFMT_SL2 7
- #define SFMT_SR1 19
- #define SFMT_SR2 1
- #define SFMT_MSK1 0xfdffffbfU
- #define SFMT_MSK2 0xbff7ff3fU
- #define SFMT_MSK3 0xfd77efffU
- #define SFMT_MSK4 0xbf9ff3ffU
- #define SFMT_PARITY1 0x00000001U
- #define SFMT_PARITY2 0x00000000U
- #define SFMT_PARITY3 0x00000000U
- #define SFMT_PARITY4 0xe9528d85U
- #define SFMT_ALTI_SL1 {SFMT_SL1, SFMT_SL1, SFMT_SL1, SFMT_SL1}
- #define SFMT_ALTI_SR1 {SFMT_SR1, SFMT_SR1, SFMT_SR1, SFMT_SR1}
- #define SFMT_ALTI_MSK {SFMT_MSK1, SFMT_MSK2, SFMT_MSK3, SFMT_MSK4}
- #define SFMT_ALTI_MSK64 {SFMT_MSK2, SFMT_MSK1, SFMT_MSK4, SFMT_MSK3}
- #define SFMT_ALTI_SL2_PERM {25,25,25,25,3,25,25,25,7,0,1,2,11,4,5,6}
- #define SFMT_ALTI_SL2_PERM64 {7,25,25,25,25,25,25,15,0,1,2,3,4,5,6}
- #define SFMT_ALTI_SR2_PERM {7,0,1,2,11,4,5,6,15,8,9,10,17,12,13,14}
- #define SFMT_ALTI_SR2_PERM64 {15,0,1,2,3,4,5,6,17,8,9,10,11,12,13,14}
- #define SFMT_IDSTR "SFMT-86243:366-6-7-19-1:fdbffbf-bff7ff3f-fd77efff-bf9ff3ff"

8.68.1 Macro Definition Documentation

```
8.68.1.1 #define SFMT_ALTI_MSK {SFMT_MSK1, SFMT_MSK2, SFMT_MSK3, SFMT_MSK4}

8.68.1.2 #define SFMT_ALTI_MSK64 {SFMT_MSK2, SFMT_MSK1, SFMT_MSK4, SFMT_MSK3}

8.68.1.3 #define SFMT_ALTI_SL1 {SFMT_SL1, SFMT_SL1, SFMT_SL1, SFMT_SL1}

8.68.1.4 #define SFMT_ALTI_SL2_PERM {25,25,25,25,3,25,25,25,7,0,1,2,11,4,5,6}

8.68.1.5 #define SFMT_ALTI_SL2_PERM64 {7,25,25,25,25,25,25,25,15,0,1,2,3,4,5,6}

8.68.1.6 #define SFMT_ALTI_SR1 {SFMT_SR1, SFMT_SR1, SFMT_SR1, SFMT_SR1}

8.68.1.7 #define SFMT_ALTI_SR2_PERM {7,0,1,2,11,4,5,6,15,8,9,10,17,12,13,14}

8.68.1.8 #define SFMT_ALTI_SR2_PERM64 {15,0,1,2,3,4,5,6,17,8,9,10,11,12,13,14}

8.68.1.9 #define SFMT_IDSTR "SFMT-86243:366-6-7-19-1:fdbffbf-bff7ff3f-fd77efff-bf9ff3ff"

8.68.1.10 #define SFMT_MSK1 0fdbffbfU

8.68.1.11 #define SFMT_MSK2 0xbff7ff3fU

8.68.1.12 #define SFMT_MSK3 0xfd77efffU

8.68.1.13 #define SFMT_MSK4 0xbff9ff3ffU

8.68.1.14 #define SFMT_PARAMS86243_H

8.68.1.15 #define SFMT_PARITY1 0x00000001U

8.68.1.16 #define SFMT_PARITY2 0x00000000U

8.68.1.17 #define SFMT_PARITY3 0x00000000U

8.68.1.18 #define SFMT_PARITY4 0xe9528d85U

8.68.1.19 #define SFMT_POS1 366

8.68.1.20 #define SFMT_SL1 6

8.68.1.21 #define SFMT_SL2 7

8.68.1.22 #define SFMT_SR1 19

8.68.1.23 #define SFMT_SR2 1
```

8.69 include/SFMT-sse2-msc.h File Reference

SIMD oriented Fast Mersenne Twister(SFMT) for Intel SSE2 for MSC.

Namespaces

- [ProtocolPP](#)

Macros

- `#define SFMT_SSE2_MSC_H`

Functions

- static void `ProtocolPP::mm_recursion` (`__m128i *r, __m128i a, __m128i b, __m128i c, __m128i *d)`
- void `ProtocolPP::fmt_gen_rand_all` ()
- static void `ProtocolPP::gen_rand_array` (`w128_t *array, int size)`

8.69.1 Detailed Description

SIMD oriented Fast Mersenne Twister(SFMT) for Intel SSE2 for MSC.

Author

Mutsuo Saito (Hiroshima University)
Makoto Matsumoto (Hiroshima University)

Note

We assume LITTLE ENDIAN in this file

Copyright (C) 2006, 2007 Mutsuo Saito, Makoto Matsumoto and Hiroshima University. All rights reserved. Copyright (C) 2013 Mutsuo Saito, Makoto Matsumoto and Hiroshima University.

The new BSD License is applied to this software, see LICENSE.txt

8.69.2 Macro Definition Documentation

8.69.2.1 `#define SFMT_SSE2_MSC_H`

8.70 include/SFMT-sse2.h File Reference

SIMD oriented Fast Mersenne Twister(SFMT) for Intel SSE2.

Namespaces

- `ProtocolPP`

Macros

- `#define SFMT_SSE2_H`

Functions

- static void `ProtocolPP::mm_recursion` (`__m128i *r, __m128i a, __m128i b, __m128i c, __m128i d)`
- void `ProtocolPP::fmt_gen_rand_all` ()
- static void `ProtocolPP::gen_rand_array` (`w128_t *array, int size)`

8.70.1 Detailed Description

SIMD oriented Fast Mersenne Twister(SFMT) for Intel SSE2.

Author

Mutsuo Saito (Hiroshima University)
Makoto Matsumoto (Hiroshima University)

Note

We assume LITTLE ENDIAN in this file

Copyright (C) 2006, 2007 Mutsuo Saito, Makoto Matsumoto and Hiroshima University. All rights reserved.

The new BSD License is applied to this software, see LICENSE.txt

8.70.2 Macro Definition Documentation

8.70.2.1 #define SFMT_SSE2_H

8.71 include/SFMT.h File Reference

```
#include <stdio.h>
#include <string.h>
#include <assert.h>
#include <inttypes.h>
#include "SFMT-params.h"
```

Classes

- union [ProtocolPP::W128_T](#)
- class [ProtocolPP::sfmt](#)

Namespaces

- [ProtocolPP](#)

Macros

- #define [SFMT_MEXP](#) 19937
- #define [PRIu64](#) "%lu"
- #define [PRIx64](#) "%lx"

Typedefs

- typedef union W128_T [ProtocolPP::w128_t](#)

8.71.1 Macro Definition Documentation

- 8.71.1.1 `#define PRlu64 "llu"`
- 8.71.1.2 `#define PRlx64 "llx"`
- 8.71.1.3 `#define SFMT_MEXP 19937`

8.72 include/tinyxml2.h File Reference

```
#include "jarray.h"
#include <cctype>
#include <climits>
#include <cstdio>
#include <cstdlib>
#include <cstring>
#include <stdint.h>
```

Classes

- class [tinyxml2::StrPair](#)
- class [tinyxml2::DynArray< T, INITIAL_SIZE >](#)
- class [tinyxml2::MemPool](#)
- class [tinyxml2::MemPoolT< ITEM_SIZE >](#)
- class [tinyxml2::XMLVisitor](#)
- class [tinyxml2::XMLUtil](#)
- class [tinyxml2::XMLNode](#)
- class [tinyxml2::XMLText](#)
- class [tinyxml2::XMLComment](#)
- class [tinyxml2::XMLDeclaration](#)
- class [tinyxml2::XMLUnknown](#)
- class [tinyxml2::XmlAttribute](#)
- class [tinyxml2::XMLElement](#)
- class [tinyxml2::XMLDocument](#)
- class [tinyxml2::XMLHandle](#)
- class [tinyxml2::XMLConstHandle](#)
- class [tinyxml2::XMLPrinter](#)

Namespaces

- [tinyxml2](#)

Macros

- `#define HAS_SIZEOF_INT128_64BIT (defined(__SIZEOF_INT128__) && defined(__LP64__))`
- `#define HAS_MSVC_64BIT (defined(_MSC_VER) && defined(_M_X64))`
- `#define HAS_GCC_4_4_64BIT (defined(__GNUC__) && defined(__LP64__) && ((__GNUC__ > 4) || ((__GNUC__ == 4) && (__GNUC_MINOR__ >= 4))))`
- `#define TINYXML2_LIB`
- `#define TIXMLASSERT(x) {}`
- `#define TINYXML2_MAJOR_VERSION 7`
- `#define TINYXML2_MINOR_VERSION 0`
- `#define TINYXML2_PATCH_VERSION 1`

Enumerations

- enum `tinyxml2::XMLError` {
 `tinyxml2::XML_SUCCESS` = 0, `tinyxml2::XML_NO_ATTRIBUTE`, `tinyxml2::XML_WRONG_ATTRIBUTE_TYPE`, `tinyxml2::XML_ERROR_FILE_NOT_FOUND`,
 `tinyxml2::XML_ERROR_FILE_COULD_NOT_BE_OPENED`, `tinyxml2::XML_ERROR_FILE_READ_ERROR`,
 `tinyxml2::XML_ERROR_PARSING_ELEMENT`, `tinyxml2::XML_ERROR_PARSING_ATTRIBUTE`,
 `tinyxml2::XML_ERROR_PARSING_TEXT`, `tinyxml2::XML_ERROR_PARSING_CDATA`, `tinyxml2::XML_ERROR_PARSING_COMMENT`,
 `tinyxml2::XML_ERROR_PARSING_UNKNOWN`, `tinyxml2::XML_ERROR_EMPTY_DOCUMENT`, `tinyxml2::XML_ERROR_MISMATCHED_ELEMENT`, `tinyxml2::XML_ERROR_PARSING`,
 `tinyxml2::XML_CAN_NOT_CONVERT_TEXT`, `tinyxml2::XML_NO_TEXT_NODE`, `tinyxml2::XML_ELEMENT_DEPTH_EXCEEDED`, `tinyxml2::XML_ERROR_COUNT` }
- enum `tinyxml2::Whitespace` { `tinyxml2::PRESERVE_WHITESPACE`, `tinyxml2::COLLAPSE_WHITESPACE` }

Variables

- static const int `TIXML2_MAJOR_VERSION` = 7
- static const int `TIXML2_MINOR_VERSION` = 0
- static const int `TIXML2_PATCH_VERSION` = 1
- static const int `TINYXML2_MAX_ELEMENT_DEPTH` = 100

8.72.1 Macro Definition Documentation

- 8.72.1.1 `#define HAS_GCC_4_4_64BIT (defined(__GNUC__) && defined(__LP64__) && ((__GNUC__ > 4) || (__GNUC__ == 4) && (__GNUC_MINOR__ >= 4)))`
- 8.72.1.2 `#define HAS_MSVC_64BIT (defined(_MSC_VER) && defined(_M_X64))`
- 8.72.1.3 `#define HAS_SIZEOF_INT128_64BIT (defined(__SIZEOF_INT128__) && defined(__LP64__))`

Original code by Lee Thomason (www.grinninglizard.com)

This software is provided 'as-is', without any express or implied warranty. In no event will the authors be held liable for any damages arising from the use of this software.

Permission is granted to anyone to use this software for any purpose, including commercial applications, and to alter it and redistribute it freely, subject to the following restrictions:

1. The origin of this software must not be misrepresented; you must not claim that you wrote the original software. If you use this software in a product, an acknowledgment in the product documentation would be appreciated but is not required.
2. Altered source versions must be plainly marked as such, and must not be misrepresented as being the original software.
3. This notice may not be removed or altered from any source distribution.

Protocol++ modified by : John Peter Greninger © John Peter Greninger 2015-2019 All Rights Reserved

All copyrights and trademarks are the property of their respective owners

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer

- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution
- Any and all modifications must be returned to John Peter Greninger at GitHub.com <https://github.com/jpgreninger/ProtocolPP> for evaluation. Inclusion of modifications in the source code shall be determined solely by John Peter Greninger. Failure to provide modifications shall render this license NULL and VOID and revoke any rights to use of Protocol++
- Commercial use requires a fee-based license obtainable at www.ProtocolPP.com
- Academic use requires written and notarized permission from John Peter and Sheila Rocha Greninger
- US Copyrights at <https://www.copyright.gov/>
 - TXu002059872 (Version 1.0.0)
 - TXu002066632 (Version 1.2.7)
 - TXu002082674 (Version 1.4.0)
 - TXu002097880 (Version 2.0.0)

The names of its contributors may not be used to endorse or promote products derived from this software without specific prior written permission

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE

8.72.1.4 #define TINYXML2_LIB

8.72.1.5 #define TINYXML2_MAJOR_VERSION 7

8.72.1.6 #define TINYXML2_MINOR_VERSION 0

8.72.1.7 #define TINYXML2_PATCH_VERSION 1

8.72.1.8 #define TIXMLASSERT(x) {}

8.72.2 Variable Documentation

8.72.2.1 const int TINYXML2_MAX_ELEMENT_DEPTH = 100 [static]

8.72.2.2 const int TINYXML2_MAJOR_VERSION = 7 [static]

8.72.2.3 const int TINYXML2_MINOR_VERSION = 0 [static]

8.72.2.4 const int TINYXML2_PATCH_VERSION = 1 [static]

8.73 include/wasp.h File Reference

```
#include <time.h>
```

```
#include <sys/types.h>
#include <unistd.h>
#include <sys/wait.h>
#include <exception>
#include "jlogger.h"
#include "jrand.h"
#include "jikeprf.h"
#include "tinyxml2.h"
#include "jdata.h"
#include "jpacket.h"
#include "jstream.h"
#include "jprotocolpp.h"
```

Classes

- class [ProtocolPP::wasp](#)

Namespaces

- [ProtocolPP](#)

8.74 jikev2/include/jikencrypt.h File Reference

```
#include <string>
#include <memory>
#include "kcapi.h"
#include "jenum.h"
#include "jlogger.h"
#include "jarray.h"
#include "jrand.h"
#include "jsecass.h"
#include "jikev2sa.h"
```

Classes

- class [ProtocolPP::jikencrypt](#)

Namespaces

- [ProtocolPP](#)

8.75 jikev2/include/jikeparse.h File Reference

```
#include <time.h>
```

```
#include <sys/types.h>
#include <unistd.h>
#include <sys/wait.h>
#include <exception>
#include <unordered_map>
#include "jrand.h"
#include "tinyxml2.h"
#include "jdata.h"
#include "jpacket.h"
#include "jstream.h"
#include "jprotocolpp.h"
```

Classes

- class [ProtocolPP::jikeparse](#)
- struct [ProtocolPP::jikeparse::jikepolicy](#)
- struct [ProtocolPP::jikeparse::jikecfg](#)

Namespaces

- [ProtocolPP](#)

8.76 jikev2/include/jikeprf.h File Reference

```
#include <string>
#include "ciphers.h"
```

Classes

- class [ProtocolPP::jikeprf](#)

Namespaces

- [ProtocolPP](#)

8.77 jikev2/include/jikev2.h File Reference

```
#include <keyutils.h>
```

```
#include <sys/types.h>
#include <unistd.h>
#include <netdb.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <linux/if_alg.h>
#include <string.h>
#include <crypt.h>
#include <memory>
#include <unordered_map>
#include <thread>
#include "jlogger.h"
#include "jenum.h"
#include "jarray.h"
#include "jrand.h"
#include "jsecass.h"
#include "jipsecsa.h"
#include "jikev2dh.h"
#include "jikeprf.h"
#include "jikencrypt.h"
#include "jikeparse.h"
#include "jikev2sa.h"
#include "jdsa.h"
#include "jecdsa.h"
#include "jrsa.h"
#include "nbtheory.h"
#include "osrng.h"
#include "aes.h"
#include "md5.h"
#include "sha.h"
#include "dh.h"
#include "eccrypto.h"
#include "ecp.h"
#include "ec2n.h"
#include "rsa.h"
#include "dsa.h"
#include "pssr.h"
#include "secblock.h"
#include "oids.h"
#include "asn.h"
#include "integer.h"
```

Classes

- class [ProtocolPP::jikev2](#)

Namespaces

- [ProtocolPP](#)

Macros

- #define [AF_ALG](#) 38
- #define [SOL_ALG](#) 279

- #define CRYPTOPP_ENABLE_NAMESPACE_WEAK 1

8.77.1 Macro Definition Documentation

8.77.1.1 #define AF_ALG 38

8.77.1.2 #define CRYPTOPP_ENABLE_NAMESPACE_WEAK 1

8.77.1.3 #define SOL_ALG 279

8.78 jikev2/include/jikev2dh.h File Reference

```
#include <sys/types.h>
#include <unistd.h>
#include <string.h>
#include <crypt.h>
#include <memory>
#include "jlogger.h"
#include "jenum.h"
#include "jarray.h"
#include "nbtheory.h"
#include "osrng.h"
#include "aes.h"
#include "dh.h"
#include "eccrypto.h"
#include "secblock.h"
#include "oids.h"
#include "asn.h"
#include "integer.h"
```

Classes

- class [ProtocolPP::jikev2dh](#)

Namespaces

- [ProtocolPP](#)

8.79 jikev2/include/jikev2sa.h File Reference

```
#include "jsecass.h"
```

Classes

- class [ProtocolPP::jikev2sa](#)

Namespaces

- [ProtocolPP](#)

8.80 jlogger/include/jlogger.h File Reference

```
#include <mutex>
#include <iostream>
#include <chrono>
#include <ctime>
#include <memory>
#include <fstream>
#include <sstream>
```

Classes

- class [InterfacePP::log_policy_interface](#)
- class [InterfacePP::file_log_policy](#)
- class [InterfacePP::file_stdout_log_policy](#)
- class [InterfacePP::file_quiet_log_policy](#)
- class [InterfacePP::jlogger](#)

Namespaces

- [InterfacePP](#)

Macros

- `#define COLORIZE`
- `#define DARK_THEME`

8.80.1 Macro Definition Documentation

8.80.1.1 `#define COLORIZE`

8.80.1.2 `#define DARK_THEME`

8.81 jresponder/include/jexec.h File Reference

```
#include <unistd.h>
#include "jprotocolpp.h"
#include "jlogger.h"
```

Classes

- class [InterfacePP::jexec](#)

Namespaces

- [InterfacePP](#)

8.82 jresponder/include/jresponder.h File Reference

```
#include <iostream>
#include <memory>
#include <unistd.h>
#include "jprotocolpp.h"
#include "jring.h"
#include "jexec.h"
#include "jlogger.h"
```

Classes

- class [InterfacePP::jresponder](#)

Namespaces

- [InterfacePP](#)

8.83 jresponder/include/jsecresponder.h File Reference

```
#include <iostream>
#include <memory>
#include <unistd.h>
#include "jprotocolpp.h"
#include "jring.h"
#include "jexec.h"
#include "jlogger.h"
```

Classes

- class [InterfacePP::jsecresponder](#)

Namespaces

- [InterfacePP](#)

8.84 jtestbench/include/interfacepp.h File Reference

```
#include <thread>
#include "wasp.h"
#include "optionparser.h"
#include "jlogger.h"
#include "jring.h"
#include "jmmu.h"
```

8.85 jtestbench/include/jmmu.h File Reference

```
#include <iostream>
#include <memory>
#include <string>
#include <map>
#include <vector>
#include "jsecass.h"
#include "jprotocolpp.h"
```

Classes

- class [InterfacePP::jmmu](#)

Namespaces

- [InterfacePP](#)

8.86 jtestbench/include/jproducer.h File Reference

```
#include <time.h>
#include <sys/types.h>
#include <unistd.h>
#include <sys/wait.h>
#include <exception>
#include <ratio>
#include <chrono>
#include <regex>
#include "jdata.h"
#include "jpacket.h"
#include "jstream.h"
#include "jlogger.h"
#include "jring.h"
#include "jmmu.h"
#include "jprotocolpp.h"
```

Classes

- class [InterfacePP::jproducer](#)

Namespaces

- [InterfacePP](#)

8.87 jtestbench/include/jring.h File Reference

```
#include <memory>
```

```
#include <string.h>
#include <iostream>
#include <mutex>
#include "jlogger.h"
```

Classes

- class [InterfacePP::ringin](#)
- class [InterfacePP::ringout](#)
- class [InterfacePP::ringflow](#)
- class [InterfacePP::secin](#)
- class [InterfacePP::secout](#)
- class [InterfacePP::jring< TR >](#)

Namespaces

- [InterfacePP](#)

8.88 jtestbench/include/jsecproducer.h File Reference

```
#include <time.h>
#include <sys/types.h>
#include <unistd.h>
#include <sys/wait.h>
#include <exception>
#include <ratio>
#include <chrono>
#include "jdata.h"
#include "jpacket.h"
#include "jstream.h"
#include "jlogger.h"
#include "jring.h"
#include "jsec.h"
#include "jmmu.h"
#include "jprotocolpp.h"
```

Classes

- class [InterfacePP::jsecproducer](#)

Namespaces

- [InterfacePP](#)

8.89 jtestbench/include/jsectestbench.h File Reference

```
#include "jsecproducer.h"
```

Classes

- class [InterfacePP::jsectestbench](#)

Namespaces

- [InterfacePP](#)

8.90 jtestbench/include/jtestbench.h File Reference

```
#include "jproducer.h"
```

Classes

- class [InterfacePP::jtestbench](#)

Namespaces

- [InterfacePP](#)

8.91 jtestbench/include/optionparser.h File Reference

This is the only file required to use The Lean Mean C++ Option Parser. Just #include it and you're set.

Classes

- struct [option::Descriptor](#)
Describes an option, its help text (usage) and how it should be parsed.
- class [option::Option](#)
A parsed option from the command line together with its argument if it has one.
- struct [option::Arg](#)
Functions for checking the validity of option arguments.
- struct [option::Stats](#)
Determines the minimum lengths of the buffer and options arrays used for Parser.
- class [option::Parser](#)
Checks argument vectors for validity and parses them into data structures that are easier to work with.
- struct [option::Parser::Action](#)
- class [option::Stats::CountOptionsAction](#)
- class [option::Parser::StoreOptionAction](#)
- struct [option::PrintUsageImplementation](#)
- struct [option::PrintUsageImplementation::IStringWriter](#)
- struct [option::PrintUsageImplementation::FunctionWriter< Function >](#)
- struct [option::PrintUsageImplementation::OStreamWriter< OStream >](#)
- struct [option::PrintUsageImplementation::TemporaryWriter< Temporary >](#)
- struct [option::PrintUsageImplementation::SyscallWriter< Syscall >](#)
- struct [option::PrintUsageImplementation::StreamWriter< Function, Stream >](#)
- class [option::PrintUsageImplementation::LinePartIterator](#)
- class [option::PrintUsageImplementation::LineWrapper](#)

Namespaces

- [option](#)

The namespace of The Lean Mean C++ Option Parser.

Typedefs

- [typedef ArgStatus\(* option::CheckArg \)\(const Option &option, bool msg\)](#)
Signature of functions that check if an argument is valid for a certain type of option.

Enumerations

- [enum option::ArgStatus { option::ARG_NONE, option::ARG_OK, option::ARG_REQUIRED, option::ARG_IGNORE, option::ARG_ILLEGAL }](#)

Possible results when checking if an argument is valid for a certain option.

Functions

- [template<typename OStream > void option::printUsage \(OStream &prn, const Descriptor usage\[\], int width=80, int last_column_min_percent=50, int last_column_own_line_max_percent=75\)](#)
Outputs a nicely formatted usage string with support for multi-column formatting and line-wrapping.
- [template<typename Function > void option::printUsage \(Function *prn, const Descriptor usage\[\], int width=80, int last_column_min_percent=50, int last_column_own_line_max_percent=75\)](#)
- [template<typename Temporary > void option::printUsage \(const Temporary &prn, const Descriptor usage\[\], int width=80, int last_column_min_percent=50, int last_column_own_line_max_percent=75\)](#)
- [template<typename Syscall > void option::printUsage \(Syscall *prn, int fd, const Descriptor usage\[\], int width=80, int last_column_min_percent=50, int last_column_own_line_max_percent=75\)](#)
- [template<typename Function , typename Stream > void option::printUsage \(Function *prn, Stream *stream, const Descriptor usage\[\], int width=80, int last_column_min_percent=50, int last_column_own_line_max_percent=75\)](#)

8.91.1 Detailed Description

This is the only file required to use The Lean Mean C++ Option Parser. Just #include it and you're set. The Lean Mean C++ Option Parser handles the program's command line arguments (argc, argv). It supports the short and long option formats of getopt(), getopt_long() and getopt_long_only() but has a more convenient interface. The following features set it apart from other option parsers:

Highlights:

- It is a header-only library. Just #include "optionparser.h" and you're set.
- It is freestanding. There are no dependencies whatsoever, not even the C or C++ standard library.
- It has a usage message formatter that supports column alignment and line wrapping. This aids localization because it adapts to translated strings that are shorter or longer (even if they contain Asian wide characters).
- Unlike getopt() and derivatives it doesn't force you to loop through options sequentially. Instead you can access options directly like this:
 - Test for presence of a switch in the argument vector:

```

    if ( options[QUIET] ) ...
- Evaluate --enable-foo/--disable-foo pair where the last one used wins:
    if ( options[FOO].last()->type() == DISABLE ) ...
- Cumulative option (-v verbose, -vv more verbose, -vvv even more verbose):
    int verbosity = options[VERBOSE].count();
- Iterate over all --file=<fname> arguments:
    for (Option* opt = options[FILE]; opt; opt = opt->next() )
        * fname = opt->arg; ...
- If you really want to, you can still process all arguments in order:
    * for (int i = 0; i < p.optionsCount(); ++i) {
    *     Option& opt = buffer[i];
    *     switch(opt.index()) {
    *         case HELP:   ...
    *         case VERBOSE: ...
    *         case FILE:   fname = opt.arg; ...
    *         case UNKNOWN: ...
    *

```

Despite these features the code size remains tiny. It is smaller than uClibc's GNU getopt() and just a couple 100 bytes larger than uClibc's SUSv3 getopt().

(This does not include the usage formatter, of course. But you don't have to use that.)

Download:

Tarball with examples and test programs: [optionparser-1.4.tar.gz](#)

Just the header (this is all you really need): [optionparser.h](#)

Changelog:

Version 1.4: Fixed 2 `printUsage()` bugs that messed up output with small COLUMNS values

Version 1.3: Compatible with Microsoft Visual C++.

Version 1.2: Added `Option::namelen` and removed the extraction of short option characters into a special buffer.

Changed `Arg::Optional` to accept arguments if they are attached rather than separate. This is what GNU getopt() does and how POSIX recommends utilities should interpret their arguments.

Version 1.1: Optional mode with argument reordering as done by GNU getopt(), so that options and non-options can be mixed. See `Parser::parse()`.

Feedback:

Send questions, bug reports, feature requests etc. to: [optionparser-feedback \(a\) lists.sourceforge.net](#)

Example program:

(Note: `option::*` identifiers are links that take you to their documentation.)

```

* #error EXAMPLE SHORTENED FOR READABILITY. BETTER EXAMPLES ARE IN THE .TAR.GZ!
* #include <iostream>
* #include "optionparser.h"
*
* enum optionIndex { UNKNOWN, HELP, PLUS };
* const option::Descriptor usage[] =
* {
*     {UNKNOWN, 0,"", "", option::Arg::None, "USAGE: example [options]\n\n"
*      "Options: " },
*     {HELP,     0,"", "help",option::Arg::None, " --help \tPrint usage and exit." },
*     {PLUS,     0,"p", "plus",option::Arg::None, " --plus, -p \tIncrement count." },
*     {UNKNOWN, 0,"", "", option::Arg::None, "\nExamples:\n"
*      " example --unknown -- --this_is_no_option\n"
*      " example -unk --plus -ppp file1 file2\n" },
*     {0,0,0,0,0,0}
* };
*
* int main(int argc, char* argv[])
* {
*     argc-=(argc>0); argv+=(argc>0); // skip program name argv[0] if present
*     option::Stats stats(usage, argc, argv);
*     option::Option options[stats.options_max], buffer[stats.buffer_max];
*     option::Parser parse(usage, argc, argv, options, buffer);
*     *
*     if (parse.error())

```

```

*     return 1;
*
*     if (options[HELP] || argc == 0) {
*         option::printUsage(std::cout, usage);
*         return 0;
*     }
*
*     std::cout << "--plus count: " <<
*         options[PLUS].count() << "\n";
*
*     for (option::Option* opt = options[UNKNOWN]; opt; opt = opt->
*         next())
*         std::cout << "Unknown option: " << opt->name << "\n";
*
*     for (int i = 0; i < parse.nonOptionsCount(); ++i)
*         std::cout << "Non-option #" << i << ":" << parse.nonOption(i) << "\n";
* }
*

```

Option syntax:

- The Lean Mean C++ Option Parser follows POSIX `getopt()` conventions and supports GNU-style `getopt_long()` long options as well as Perl-style single-minus long options (`getopt_long_-only()`).
- short options have the format `-X` where X is any character that fits in a char.
- short options can be grouped, i.e. `-X -Y` is equivalent to `-XY`.
- a short option may take an argument either separate (`-X foo`) or attached (`-Xfoo`). You can make the parser accept the additional format `-X=foo` by registering X as a long option (in addition to being a short option) and enabling single-minus long options.
- an argument-taking short option may be grouped if it is the last in the group, e.g. `-ABCXfoo` or `-ABCX foo` (`foo` is the argument to the `-X` option).
- a lone minus character '`'-`' is not treated as an option. It is customarily used where a file name is expected to refer to `stdin` or `stdout`.
- long options have the format `--option-name`.
- the option-name of a long option can be anything and include any characters. Even = characters will work, but don't do that.
- [optional] long options may be abbreviated as long as the abbreviation is unambiguous. You can set a minimum length for abbreviations.
- [optional] long options may begin with a single minus. The double minus form is always accepted, too.
- a long option may take an argument either separate (`--option arg`) or attached (`--option=arg`). In the attached form the equals sign is mandatory.
- an empty string can be passed as an attached long option argument: `--option-name= .` Note the distinction between an empty string as argument and no argument at all.
- an empty string is permitted as separate argument to both long and short options.
- Arguments to both short and long options may start with a '`'-`' character. E.g. `--X-X`, `-X -X` or `--long-X=-X`. If `-X` and `--long-X` take an argument, that argument will be "`-X`" in all 3 cases.
- If using the built-in [Arg::Optional](#), optional arguments must be attached.
- the special option `-` (i.e. without a name) terminates the list of options. Everything that follows is a non-option argument, even if it starts with a '`'-`' character. The `-` itself will not appear in the parse results.
- the first argument that doesn't start with '`'-`' or '`'--'`' and does not belong to a preceding argument-taking option, will terminate the option list and is the first non-option argument. All following command line arguments are treated as non-option arguments, even if they start with '`'-`'.

NOTE: This behaviour is mandated by POSIX, but GNU `getopt()` only honours this if it is explicitly requested (e.g. by setting `POSIXLY_CORRECT`).

You can enable the GNU behaviour by passing `true` as first argument to e.g. [Parser::parse\(\)](#).

- Arguments that look like options (i.e. '`'-`' followed by at least 1 character) but aren't, are NOT treated as non-option arguments. They are treated as unknown options and are collected into a list of unknown options for error reporting.

This means that in order to pass a first non-option argument beginning with the minus character it is required to use the `-` special option, e.g.

```
*      program -x -- --strange-filename
*
```

In this example, `--strange-filename` is a non-option argument. If the `-` were omitted, it would be treated as an unknown option.

See [option::Descriptor::longopt](#) for information on how to collect unknown options.

8.92 platforms/SEC/include/jsec.h File Reference

```
#include "assert.h"
#include "jprotocolpp.h"
```

Classes

- class [PlatformPP::jsec](#)
- struct [PlatformPP::jsec::sgt_t](#)

Namespaces

- [PlatformPP](#)

8.93 platforms/SEC/include/jsgt.h File Reference

```
#include "assert.h"
#include "jprotocol.h"
```

Classes

- class [PlatformPP::jsgt](#)

Namespaces

- [PlatformPP](#)

Functions

- class [PlatformPP::jsgt](#) [PlatformPP::jsgt \(sgt_t *sgt\)](#)
- [sgt_t \(\)](#)
- [jsgt \(sgt_t *sgt\)](#)
- virtual [~jsgt \(\)](#)
- void [get \(unsigned int size, uint8_t *data, uint8_t *sgtout\)](#)
- static void [read \(unsigned int sgtsize, uint8_t *sgtin, uint8_t *data\)](#)

Variables

- struct `sgt_t`
- std::string `chunksize`
- unsigned int `addrsize`
- unsigned int `extend`
- unsigned int `sgtfinal`
- unsigned int `length`
- unsigned int `bpid`
- unsigned int `offset`
- unsigned int `entrysize`
- std::vector< std::string > `order`
- `sgt_t * m_sgt`

8.93.1 Function Documentation

8.93.1.1 void `jsgt::get(unsigned int size, uint8_t * data, uint8_t * sgtout)`

8.93.1.2 `jsgt::jsgt(sgt_t * sgt)`

8.93.1.3 static void `jsgt::read(unsigned int sgtsize, uint8_t * sgtin, uint8_t * data) [static]`

8.93.1.4 `jsgt::sgt_t()`

8.93.1.5 virtual `jsgt::~jsgt() [virtual]`

8.93.2 Variable Documentation

8.93.2.1 unsigned int `addrsize`

8.93.2.2 unsigned int `bpid`

8.93.2.3 std::string `chunksize`

8.93.2.4 unsigned int `entrysize`

8.93.2.5 unsigned int `extend`

8.93.2.6 unsigned int `length`

8.93.2.7 `sgt_t* m_sgt`

8.93.2.8 unsigned int `offset`

8.93.2.9 std::vector<std::string> `order`

8.93.2.10 struct `sgt_t`

Initial value:

```
{
    platform_t platform
```

8.93.2.11 unsigned int sgffinal

8.94 schema/ikev2.xsd File Reference

8.95 schema/protocolpp.xsd File Reference