

TII TLS1.3

1.1

Generated by Doxygen 1.9.8

1 TIGER TLS C++	1
1.1 Building	1
1.1.1 Miracl	1
1.1.2 Miracl + LibSodium	2
1.1.3 Miracl + TLSECC	2
1.2 Try it out	2
1.2.1 Client side Authentication	5
1.2.2 Testing Pre-shared keys	5
2 Configure the Arduino Nano RP2040	7
2.1 Building the client application on the Arduino Nano RP2040 board.	7
3 Data Structure Index	9
3.1 Data Structures	9
4 File Index	11
4.1 File List	11
5 Data Structure Documentation	13
5.1 credential Struct Reference	13
5.1.1 Detailed Description	13
5.1.2 Field Documentation	13
5.1.2.1 certchain	13
5.1.2.2 publickey	14
5.1.2.3 secretkey	14
5.1.2.4 CERTCHAIN	14
5.1.2.5 PUBLICKEY	14
5.1.2.6 SECRETKEY	14
5.1.2.7 requirements	14
5.1.2.8 nreqs	14
5.1.2.9 nreqsraw	14
5.1.2.10 sigalg	15
5.2 crypto Struct Reference	15
5.2.1 Detailed Description	15
5.2.2 Field Documentation	15
5.2.2.1 active	15
5.2.2.2 k	15
5.2.2.3 iv	16
5.2.2.4 K	16
5.2.2.5 IV	16
5.2.2.6 record	16
5.2.2.7 suite	16
5.2.2.8 taglen	16
5.3 ECCX08Class Class Reference	17

5.4 ee_status Struct Reference	17
5.4.1 Detailed Description	18
5.4.2 Field Documentation	18
5.4.2.1 early_data	18
5.4.2.2 alpn	18
5.4.2.3 server_name	18
5.4.2.4 max_frag_length	18
5.5 octad Struct Reference	18
5.5.1 Detailed Description	19
5.5.2 Field Documentation	19
5.5.2.1 len	19
5.5.2.2 max	19
5.5.2.3 val	19
5.6 pktype Struct Reference	19
5.6.1 Detailed Description	19
5.6.2 Field Documentation	20
5.6.2.1 type	20
5.6.2.2 hash	20
5.6.2.3 curve	20
5.7 ret Struct Reference	20
5.7.1 Detailed Description	20
5.7.2 Field Documentation	20
5.7.2.1 val	20
5.7.2.2 err	21
5.8 Socket Class Reference	21
5.8.1 Detailed Description	21
5.9 ticket Struct Reference	21
5.9.1 Detailed Description	22
5.9.2 Field Documentation	22
5.9.2.1 valid	22
5.9.2.2 tick	22
5.9.2.3 nonce	22
5.9.2.4 psk	22
5.9.2.5 TICK	23
5.9.2.6 NONCE	23
5.9.2.7 PSK	23
5.9.2.8 age_obfuscator	23
5.9.2.9 max_early_data	23
5.9.2.10 birth	23
5.9.2.11 lifetime	23
5.9.2.12 cipher_suite	23
5.9.2.13 favourite_group	24

5.9.2.14 origin	24
5.10 TLS_session Struct Reference	24
5.10.1 Detailed Description	25
5.10.2 Field Documentation	25
5.10.2.1 status	25
5.10.2.2 max_record	25
5.10.2.3 sockptr	25
5.10.2.4 id	25
5.10.2.5 hostname	25
5.10.2.6 cipher_suite	25
5.10.2.7 favourite_group	26
5.10.2.8 server_cert_type	26
5.10.2.9 client_cert_type	26
5.10.2.10 K_send	26
5.10.2.11 K_rcv	26
5.10.2.12 HS	26
5.10.2.13 hs	26
5.10.2.14 RMS	26
5.10.2.15 rms	27
5.10.2.16 STS	27
5.10.2.17 sts	27
5.10.2.18 CTS	27
5.10.2.19 cts	27
5.10.2.20 CTX	27
5.10.2.21 ctx	27
5.10.2.22 Ibuff	27
5.10.2.23 Obuff	28
5.10.2.24 ibuff	28
5.10.2.25 obuff	28
5.10.2.26 ptr	28
5.10.2.27 tlshash	28
5.10.2.28 T	28
5.10.2.29 expect_heartbeats	28
5.10.2.30 allowed_to_heartbeat	28
5.10.2.31 heartbeat_req_in_flight	29
5.11 unihash Struct Reference	29
5.11.1 Detailed Description	29
5.11.2 Field Documentation	29
5.11.2.1 state	29
5.11.2.2 htype	29
6 File Documentation	31

6.1 <code>tls1_3.h</code> File Reference	31
6.1.1 Detailed Description	35
6.1.2 Macro Definition Documentation	35
6.1.2.1 <code>IO_NONE</code>	35
6.1.2.2 <code>IO_APPLICATION</code>	36
6.1.2.3 <code>IO_PROTOCOL</code>	36
6.1.2.4 <code>IO_DEBUG</code>	36
6.1.2.5 <code>IO_WIRE</code>	36
6.1.2.6 <code>TINY_ECC</code>	36
6.1.2.7 <code>TYPICAL</code>	36
6.1.2.8 <code>POST_QUANTUM</code>	36
6.1.2.9 <code>HYBRID</code>	36
6.1.2.10 <code>EDDSA</code>	37
6.1.2.11 <code>NO_CERT</code>	37
6.1.2.12 <code>FROM_FILE</code>	37
6.1.2.13 <code>FROM_ROM</code>	37
6.1.2.14 <code>RSA_SS</code>	37
6.1.2.15 <code>ECC_SS</code>	37
6.1.2.16 <code>DLT_SS</code>	37
6.1.2.17 <code>HYB_SS</code>	37
6.1.2.18 <code>HW_1</code>	38
6.1.2.19 <code>HW_2</code>	38
6.1.2.20 <code>EDD_SS</code>	38
6.1.2.21 <code>CRYPTO_SETTING</code>	38
6.1.2.22 <code>VERBOSEITY</code>	38
6.1.2.23 <code>CLIENT_CERT</code>	38
6.1.2.24 <code>CLIENT_CERT_KIND</code>	38
6.1.2.25 <code>POST_HS_AUTH</code>	38
6.1.2.26 <code>TLS_APPLICATION_PROTOCOL</code>	39
6.1.2.27 <code>ALLOW_SELF_SIGNED</code>	39
6.1.2.28 <code>TRY_EARLY_DATA</code>	39
6.1.2.29 <code>MERGE_MESSAGES</code>	39
6.1.2.30 <code>CHECK_NAME_IN_CERT</code>	39
6.1.2.31 <code>TLS_SHA256_T</code>	39
6.1.2.32 <code>TLS_SHA384_T</code>	39
6.1.2.33 <code>TLS_SHA512_T</code>	40
6.1.2.34 <code>TLS_MAX_HASH_STATE</code>	40
6.1.2.35 <code>TLS_MAX_HASH</code>	40
6.1.2.36 <code>TLS_MAX_KEY</code>	40
6.1.2.37 <code>TLS_X509_MAX_FIELD</code>	40
6.1.2.38 <code>TLS_MAX_EXT_LABEL</code>	40
6.1.2.39 <code>TLS_MAX_FRAG</code>	40

6.1.2.40 TLS_MAX_IBUFF_SIZE	41
6.1.2.41 TLS_MAX_PLAIN_FRAG	41
6.1.2.42 TLS_MAX_CIPHER_FRAG	41
6.1.2.43 TLS_MAX_CERT_SIZE	41
6.1.2.44 TLS_MAX_CERT_B64	41
6.1.2.45 TLS_MAX_HELLO	41
6.1.2.46 TLS_MAX_SIG_PUB_KEY_SIZE	41
6.1.2.47 TLS_MAX_SIG_SECRET_KEY_SIZE	42
6.1.2.48 TLS_MAX_SIGNATURE_SIZE	42
6.1.2.49 TLS_MAX_KEX_PUB_KEY_SIZE	42
6.1.2.50 TLS_MAX_KEX_CIPHERTEXT_SIZE	42
6.1.2.51 TLS_MAX_KEX_SECRET_KEY_SIZE	42
6.1.2.52 TLS_MAX_SERVER_CHAIN_LEN	42
6.1.2.53 TLS_MAX_SERVER_CHAIN_SIZE	42
6.1.2.54 TLS_MAX_CLIENT_CHAIN_LEN	42
6.1.2.55 TLS_MAX_CLIENT_CHAIN_SIZE	43
6.1.2.56 TLS_MAX_SHARED_SECRET_SIZE	43
6.1.2.57 TLS_MAX_TICKET_SIZE	43
6.1.2.58 TLS_MAX_EXTENSIONS	43
6.1.2.59 TLS_MAX_ECC_FIELD	43
6.1.2.60 TLS_MAX_IV_SIZE	43
6.1.2.61 TLS_MAX_TAG_SIZE	43
6.1.2.62 TLS_MAX_COOKIE	43
6.1.2.63 TLS_MAX_OUTPUT_RECORD_SIZE	44
6.1.2.64 TLS_MAX_OBUFF_SIZE	44
6.1.2.65 TLS_MAX_SERVER_NAME	44
6.1.2.66 TLS_MAX_SUPPORTED_GROUPS	44
6.1.2.67 TLS_MAX_SUPPORTED_SIGS	44
6.1.2.68 TLS_MAX_PSK_MODES	44
6.1.2.69 TLS_MAX_CIPHER_SUITES	44
6.1.2.70 TLS_AES_128_GCM_SHA256	44
6.1.2.71 TLS_AES_256_GCM_SHA384	45
6.1.2.72 TLS_CHACHA20_POLY1305_SHA256	45
6.1.2.73 TLS_AES_128_CCM_SHA256	45
6.1.2.74 TLS_AES_128_CCM_8_SHA256	45
6.1.2.75 X25519	45
6.1.2.76 SECP256R1	45
6.1.2.77 SECP384R1	45
6.1.2.78 SECP521R1	45
6.1.2.79 X448	46
6.1.2.80 MLKEM512	46
6.1.2.81 MLKEM768	46

6.1.2.82 MLKEM1024	46
6.1.2.83 HYBRID_KX	46
6.1.2.84 ECDSA_SECP256R1_SHA256	46
6.1.2.85 ECDSA_SECP256R1_SHA384	46
6.1.2.86 ECDSA_SECP384R1_SHA384	46
6.1.2.87 RSA_PSS_RSAE_SHA256	47
6.1.2.88 RSA_PSS_RSAE_SHA384	47
6.1.2.89 RSA_PSS_RSAE_SHA512	47
6.1.2.90 RSA_PKCS1_SHA256	47
6.1.2.91 RSA_PKCS1_SHA384	47
6.1.2.92 RSA_PKCS1_SHA512	47
6.1.2.93 ED25519	47
6.1.2.94 ED448	47
6.1.2.95 MLDSA44	48
6.1.2.96 MLDSA65	48
6.1.2.97 MLDSA87	48
6.1.2.98 PSKOK	48
6.1.2.99 PSKWECDHE	48
6.1.2.100 TLS_FULL_HANDSHAKE	48
6.1.2.101 TLS_EXTERNAL_PSK	48
6.1.2.102 TLS1_0	48
6.1.2.103 TLS1_2	49
6.1.2.104 TLS1_3	49
6.1.2.105 TLS13_UPDATE_NOT_REQUESTED	49
6.1.2.106 TLS13_UPDATE_REQUESTED	49
6.1.2.107 SERVER_NAME	49
6.1.2.108 SUPPORTED_GROUPS	49
6.1.2.109 SIG_ALGS	49
6.1.2.110 POST_HANDSHAKE_AUTH	49
6.1.2.111 SIG_ALGS_CERT	50
6.1.2.112 KEY_SHARE	50
6.1.2.113 PSK_MODE	50
6.1.2.114 PRESHARED_KEY	50
6.1.2.115 TLS_VER	50
6.1.2.116 COOKIE	50
6.1.2.117 EARLY_DATA	50
6.1.2.118 MAX_FRAG_LENGTH	50
6.1.2.119 PADDING	51
6.1.2.120 APP_PROTOCOL	51
6.1.2.121 RECORD_SIZE_LIMIT	51
6.1.2.122 CLIENT_CERT_TYPE	51
6.1.2.123 SERVER_CERT_TYPE	51

6.1.2.124 HEARTBEAT	51
6.1.2.125 CERT_AUTHORITIES	51
6.1.2.126 HSHAKE	51
6.1.2.127 APPLICATION	52
6.1.2.128 ALERT	52
6.1.2.129 CHANGE_CIPHER	52
6.1.2.130 HEART_BEAT	52
6.1.2.131 TIMED_OUT	52
6.1.2.132 CLIENT_HELLO	52
6.1.2.133 SERVER_HELLO	52
6.1.2.134 CERTIFICATE	53
6.1.2.135 CERT_REQUEST	53
6.1.2.136 CERT_VERIFY	53
6.1.2.137 FINISHED	53
6.1.2.138 ENCRYPTED_EXTENSIONS	53
6.1.2.139 TICKET	53
6.1.2.140 KEY_UPDATE	53
6.1.2.141 MESSAGE_HASH	53
6.1.2.142 END_OF_EARLY_DATA	54
6.1.2.143 HANDSHAKE_RETRY	54
6.1.2.144 NOT_TLS1_3	54
6.1.2.145 BAD_CERT_CHAIN	54
6.1.2.146 ID_MISMATCH	54
6.1.2.147 UNRECOGNIZED_EXT	54
6.1.2.148 BAD_HELLO	54
6.1.2.149 WRONG_MESSAGE	54
6.1.2.150 MISSING_REQUEST_CONTEXT	55
6.1.2.151 AUTHENTICATION_FAILURE	55
6.1.2.152 BAD_RECORD	55
6.1.2.153 BAD_TICKET	55
6.1.2.154 NOT_EXPECTED	55
6.1.2.155 CA_NOT_FOUND	55
6.1.2.156 CERT_OUTOFDATE	55
6.1.2.157 MEM_OVERFLOW	55
6.1.2.158 FORBIDDEN_EXTENSION	56
6.1.2.159 MAX_EXCEEDED	56
6.1.2.160 EMPTY_CERT_CHAIN	56
6.1.2.161 SELF_SIGNED_CERT	56
6.1.2.162 TIME_OUT	56
6.1.2.163 ERROR_ALERT_RECEIVED	56
6.1.2.164 BAD_MESSAGE	56
6.1.2.165 CERT_VERIFY_FAIL	56

6.1.2.166 BAD_HANDSHAKE	57
6.1.2.167 BAD_REQUEST_UPDATE	57
6.1.2.168 CLOSURE_ALERT_RECEIVED	57
6.1.2.169 MISSING_EXTENSIONS	57
6.1.2.170 ILLEGAL_PARAMETER	57
6.1.2.171 UNEXPECTED_MESSAGE	57
6.1.2.172 DECRYPT_ERROR	57
6.1.2.173 BAD_CERTIFICATE	57
6.1.2.174 UNSUPPORTED_EXTENSION	58
6.1.2.175 UNKNOWN_CA	58
6.1.2.176 CERTIFICATE_EXPIRED	58
6.1.2.177 PROTOCOL_VERSION	58
6.1.2.178 DECODE_ERROR	58
6.1.2.179 RECORD_OVERFLOW	58
6.1.2.180 BAD_RECORD_MAC	58
6.1.2.181 HANDSHAKE_FAILURE	58
6.1.2.182 CLOSE_NOTIFY	59
6.1.2.183 MISSING_EXTENSION	59
6.1.2.184 LOG_OUTPUT_TRUNCATION	59
6.1.2.185 TLS13_DISCONNECTED	59
6.1.2.186 TLS13_CONNECTED	59
6.1.2.187 TLS13_HANDSHAKING	59
6.1.2.188 TLS_FAILURE	59
6.1.2.189 TLS_SUCCESS	59
6.1.2.190 TLS_RESUMPTION_REQUIRED	60
6.1.2.191 TLS_EARLY_DATA_ACCEPTED	60
6.1.2.192 PSK_NOT	60
6.1.2.193 PSK_KEY	60
6.1.2.194 PSK_IBE	60
6.1.2.195 X509_CERT	60
6.1.2.196 RAW_PUBLIC_KEY	60
6.1.3 Typedef Documentation	61
6.1.3.1 byte	61
6.1.3.2 sign8	61
6.1.3.3 sign16	61
6.1.3.4 sign32	61
6.1.3.5 sign64	61
6.1.3.6 unsign16	61
6.1.3.7 unsign32	61
6.1.3.8 unsign64	62
6.2 tls1_3.h	62
6.3 tls_bfibe.h File Reference	67

6.3.1 Detailed Description	67
6.3.2 Function Documentation	67
6.3.2.1 BFIBE_CCA_ENCRYPT()	67
6.3.2.2 BFIBE_CCA_DECRYPT()	68
6.4 <code>tls_bfibe.h</code>	68
6.5 <code>tls_cert_chain.h</code> File Reference	69
6.5.1 Detailed Description	69
6.5.2 Function Documentation	69
6.5.2.1 <code>checkServerCertChain()</code>	69
6.6 <code>tls_cert_chain.h</code>	70
6.7 <code>tls_certs.h</code> File Reference	70
6.7.1 Detailed Description	70
6.7.2 Function Documentation	70
6.7.2.1 <code>setCredential()</code>	70
6.7.3 Variable Documentation	71
6.7.3.1 <code>myprivate</code>	71
6.7.3.2 <code>mycert</code>	71
6.7.3.3 <code>cacerts</code>	71
6.8 <code>tls_certs.h</code>	71
6.9 <code>tls_client_rcv.h</code> File Reference	72
6.9.1 Detailed Description	73
6.9.2 Function Documentation	73
6.9.2.1 <code>parseoctad()</code>	73
6.9.2.2 <code>parsebytes()</code>	73
6.9.2.3 <code>parseInt()</code>	74
6.9.2.4 <code>parseoctadptr()</code>	74
6.9.2.5 <code>getServerRecord()</code>	74
6.9.2.6 <code>parseIntorPull()</code>	75
6.9.2.7 <code>parseoctadorPull()</code>	75
6.9.2.8 <code>parsebytesorPull()</code>	75
6.9.2.9 <code>parseoctadorPullptrX()</code>	76
6.9.2.10 <code>badResponse()</code>	76
6.9.2.11 <code>seeWhatsNext()</code>	77
6.9.2.12 <code>getServerEncryptedExtensions()</code>	77
6.9.2.13 <code>getServerCertVerify()</code>	77
6.9.2.14 <code>getServerFinished()</code>	78
6.9.2.15 <code>getServerHello()</code>	78
6.9.2.16 <code>getCheckServerCertificateChain()</code>	79
6.9.2.17 <code>getCertificateRequest()</code>	79
6.10 <code>tls_client_rcv.h</code>	79
6.11 <code>tls_client_send.h</code> File Reference	80
6.11.1 Detailed Description	82

6.11.2 Function Documentation	82
6.11.2.1 sendCCCS()	82
6.11.2.2 addHeartbeat()	82
6.11.2.3 addCertAuthorities()	83
6.11.2.4 addPreSharedKeyExt()	83
6.11.2.5 addServerNameExt()	83
6.11.2.6 addSupportedGroupsExt()	83
6.11.2.7 addServerRawPublicKey()	84
6.11.2.8 addClientRawPublicKey()	84
6.11.2.9 addSigAlgsExt()	84
6.11.2.10 addSigAlgsCertExt()	85
6.11.2.11 addKeyShareExt()	85
6.11.2.12 addALPNExt()	85
6.11.2.13 addMFLExt()	85
6.11.2.14 addRSLExt()	86
6.11.2.15 addPSKModesExt()	86
6.11.2.16 addVersionExt()	86
6.11.2.17 addPadding()	87
6.11.2.18 addCookieExt()	87
6.11.2.19 addEarlyDataExt()	87
6.11.2.20 addPostHSAuth()	87
6.11.2.21 clientRandom()	88
6.11.2.22 cipherSuites()	88
6.11.2.23 sendRecord()	88
6.11.2.24 sendClientMessage()	89
6.11.2.25 sendHeartbeatRequest()	89
6.11.2.26 sendZeroRecord()	89
6.11.2.27 sendBinder()	89
6.11.2.28 sendClientHello()	90
6.11.2.29 sendAlert()	90
6.11.2.30 sendKeyUpdate()	91
6.11.2.31 sendClientFinish()	91
6.11.2.32 sendClientCertificateChain()	91
6.11.2.33 sendClientCertVerify()	91
6.11.2.34 sendEndOfEarlyData()	92
6.11.2.35 alert_from_cause()	92
6.12 tls_client_send.h	92
6.13 tls_keys_calc.h File Reference	93
6.13.1 Detailed Description	95
6.13.2 Function Documentation	95
6.13.2.1 initTranscriptHash()	95
6.13.2.2 runningHash()	95

6.13.2.3 runningHashIO()	95
6.13.2.4 rewindIO()	96
6.13.2.5 runningHashIOrewind()	96
6.13.2.6 transcriptHash()	96
6.13.2.7 runningSyntheticHash()	96
6.13.2.8 initCryptoContext()	97
6.13.2.9 updateCryptoContext()	97
6.13.2.10 incrementCryptoContext()	97
6.13.2.11 createCryptoContext()	97
6.13.2.12 createSendCryptoContext()	98
6.13.2.13 createRecvCryptoContext()	98
6.13.2.14 recoverPSK()	98
6.13.2.15 deriveEarlySecrets()	98
6.13.2.16 deriveLaterSecrets()	99
6.13.2.17 deriveHandshakeSecrets()	99
6.13.2.18 deriveApplicationSecrets()	100
6.13.2.19 deriveUpdatedKeys()	100
6.13.2.20 checkVerifierData()	100
6.13.2.21 deriveVerifierData()	101
6.13.2.22 checkServerCertVerifier()	101
6.13.2.23 createClientCertVerifier()	101
6.14 tls_keys_calc.h	103
6.15 tls_logger.h File Reference	103
6.15.1 Detailed Description	104
6.15.2 Function Documentation	104
6.15.2.1 myprintf()	104
6.15.2.2 log()	105
6.15.2.3 logServerHello()	105
6.15.2.4 logTicket()	105
6.15.2.5 logEncExt()	106
6.15.2.6 logCert()	106
6.15.2.7 logCertDetails()	106
6.15.2.8 logServerResponse()	107
6.15.2.9 logAlert()	107
6.15.2.10 logCipherSuite()	107
6.15.2.11 logKeyExchange()	107
6.15.2.12 logSigAlg()	108
6.16 tls_logger.h	108
6.17 tls_octads.h File Reference	108
6.17.1 Detailed Description	109
6.17.2 Function Documentation	110
6.17.2.1 millis()	110

6.17.2.2 OCT_append_int()	110
6.17.2.3 OCT_append_octad()	110
6.17.2.4 OCT_compare()	110
6.17.2.5 OCT_shift_left()	111
6.17.2.6 OCT_kill()	111
6.17.2.7 OCT_from_hex()	111
6.17.2.8 OCT_append_string()	112
6.17.2.9 OCT_append_byte()	112
6.17.2.10 OCT_append_bytes()	112
6.17.2.11 OCT_from_base64()	112
6.17.2.12 OCT_reverse()	113
6.17.2.13 OCT_truncate()	113
6.17.2.14 OCT_copy()	113
6.17.2.15 OCT_output_hex()	114
6.17.2.16 OCT_output_string()	114
6.17.2.17 OCT_output_base64()	114
6.18 tls_octads.h	115
6.19 tls_pqibe.h File Reference	115
6.19.1 Detailed Description	116
6.19.2 Function Documentation	116
6.19.2.1 PQIBE_CCA_ENCRYPT()	116
6.19.2.2 PQIBE_CCA_DECRYPT()	116
6.20 tls_pqibe.h	116
6.21 tls_protocol.h File Reference	117
6.21.1 Detailed Description	117
6.21.2 Function Documentation	118
6.21.2.1 TLS13_start()	118
6.21.2.2 TLS13_end()	119
6.21.2.3 TLS13_stop()	119
6.21.2.4 TLS13_connect()	119
6.21.2.5 TLS13_send()	120
6.21.2.6 TLS13_recv()	120
6.21.2.7 TLS13_recv_and_check()	120
6.21.2.8 TLS13_clean()	121
6.22 tls_protocol.h	121
6.23 tls_sal.h File Reference	121
6.23.1 Detailed Description	123
6.23.2 Function Documentation	123
6.23.2.1 SAL_name()	123
6.23.2.2 SAL_ciphers()	123
6.23.2.3 SAL_groups()	123
6.23.2.4 SAL_sigs()	124

6.23.2.5 SAL_sigCerts()	124
6.23.2.6 SAL_initLib()	124
6.23.2.7 SAL_hashType()	124
6.23.2.8 SAL_hashLen()	125
6.23.2.9 SAL_aeadKeylen()	125
6.23.2.10 SAL_aeadTaglen()	125
6.23.2.11 SAL_randomByte()	126
6.23.2.12 SAL_randomOctad()	126
6.23.2.13 SAL_hkdfExtract()	126
6.23.2.14 SAL_hkdfExpand()	127
6.23.2.15 SAL_hmac()	127
6.23.2.16 SAL_hashNull()	127
6.23.2.17 SAL_hashInit()	128
6.23.2.18 SAL_hashProcessArray()	128
6.23.2.19 SAL_hashOutput()	128
6.23.2.20 SAL_aeadEncrypt()	128
6.23.2.21 SAL_aeadDecrypt()	129
6.23.2.22 SAL_generateKeyPair()	129
6.23.2.23 SAL_generateSharedSecret()	130
6.23.2.24 SAL_tlsSignatureVerify()	130
6.23.2.25 SAL_tlsSignature()	131
6.24 tls_sal.h	131
6.25 tls_sockets.h File Reference	132
6.25.1 Detailed Description	133
6.25.2 Function Documentation	133
6.25.2.1 setclientsock()	133
6.25.2.2 getIPaddress()	133
6.25.2.3 sendOctad()	133
6.25.2.4 sendLen()	134
6.25.2.5 getBytes()	134
6.25.2.6 getInt16()	134
6.25.2.7 getInt24()	135
6.25.2.8 getByte()	135
6.25.2.9 getOctad()	135
6.26 tls_sockets.h	136
6.27 tls_tickets.h File Reference	137
6.27.1 Detailed Description	138
6.27.2 Function Documentation	138
6.27.2.1 parseTicket()	138
6.27.2.2 initTicketContext()	138
6.27.2.3 endTicketContext()	139
6.27.2.4 ticket_still_good()	139

6.28	tls_tickets.h	139
6.29	tls_wifi.h File Reference	139
6.29.1	Detailed Description	140
6.30	tls_wifi.h	140
6.31	tls_x509.h File Reference	140
6.31.1	Detailed Description	142
6.31.2	Macro Definition Documentation	142
6.31.2.1	X509_ECC	142
6.31.2.2	X509_RSA	142
6.31.2.3	X509_ECD	142
6.31.2.4	X509_PQ	142
6.31.2.5	X509_HY	142
6.31.2.6	X509_H256	142
6.31.2.7	X509_H384	143
6.31.2.8	X509_H512	143
6.31.2.9	X509_SHAKE256	143
6.31.2.10	USE_NIST256	143
6.31.2.11	USE_ED25519	143
6.31.2.12	USE_ED448	143
6.31.2.13	USE_NIST384	143
6.31.2.14	USE_NIST521	143
6.31.3	Function Documentation	143
6.31.3.1	ecdsa_sig_encode()	143
6.31.3.2	ecdsa_sig_decode()	144
6.31.3.3	X509_extract_private_key()	144
6.31.3.4	X509_extract_cert_sig()	144
6.31.3.5	X509_extract_cert()	145
6.31.3.6	X509_find_public_key()	145
6.31.3.7	X509_get_public_key()	145
6.31.3.8	X509_extract_public_key()	146
6.31.3.9	X509_find_issuer()	146
6.31.3.10	X509_find_validity()	146
6.31.3.11	X509_find_subject()	147
6.31.3.12	X509_self_signed()	147
6.31.3.13	X509_find_entity_property()	147
6.31.3.14	X509_find_start_date()	148
6.31.3.15	X509_find_expiry_date()	148
6.31.3.16	X509_find_extensions()	148
6.31.3.17	X509_find_extension()	148
6.31.3.18	X509_find_alt_name()	149
6.31.4	Variable Documentation	149
6.31.4.1	X509_CN	149

6.31.4.2 X509_ON	149
6.31.4.3 X509_EN	150
6.31.4.4 X509_LN	150
6.31.4.5 X509_UN	150
6.31.4.6 X509_MN	150
6.31.4.7 X509_SN	150
6.31.4.8 X509_AN	150
6.31.4.9 X509_KU	150
6.31.4.10 X509_BC	150
6.32 tls_x509.h	151
6.33 ECCX08.h	152
Index	155

Chapter 1

TIIGER TLS C++

This C++ project implements a TLS1.3 client. There is also a Rust version available from this site. This C++ version is really just C plus namespaces plus pass-by-reference. These are the only features of C++ that are used. Documentation can be found in the doxygen generated file `doc/refman.pdf`

1.1 Building

The TLS library is designed to support crypto agility by allowing a mix of cryptographic providers. This functionality is provided by the SAL (Security Abstraction Layer). Below are three examples to choose from. The SAL API documentation is provided in `sal/sal.pdf`, and guided by this it should be possible to create your own SAL. To build the client on an IoT node like the Arduino RP2040, see the `readme` file in the `src/arduino` directory.

A Client private key and certificate chain (only required for client-side authentication) is either fixed in the code or read from external files. To create your own, see *MAKECERTS.md*

The long-term global store of Certificate Authority root certificates can be found in the source code file *tls_cacert.cpp*, created using the *convert.cpp* utility.

Ideally keys, chains and key stores should all be kept in external files, but in an IoT setting there may not be a file system. So in this C++ code the client private key and certificate can be stored directly in the source code file *tls_client_cert.cpp*. When using secure hardware, the client private key may not be embedded in the source code, rather it exists in secure on-board memory.

The installation process requires the *cmake* utility to be installed. Copy all files and subdirectories from this directory to a working directory. If necessary make configuration changes in *include/tls1_3.h*, in particular to select the `CRYPTO_SETTING`. For now leave it unchanged as `TYPICAL`

1.1.1 Miracl

This build gets all of its cryptography from the MIRACL core library <https://github.com/miracl/core/cpp>

```
bash ./scripts/build.sh -1
```

1.1.2 Miracl + LibSodium

To use a SAL which includes some functionality from the well known sodium crypto library <https://libsodium.gitbook.io/doc/>, install sodium, then

```
bash ./scripts/build.sh -2
```

1.1.3 Miracl + TLSECC

For a SAL with much faster elliptic curve cryptography, use the TLSECC library <https://github.com/mcarrickscott/TLSECC>. Recommended.

```
bash ./scripts/build.sh -3
```

1.2 Try it out

After the build complete successfully, the example executable *client* and the TiigerTLS library *libtiitls.a* are generated in the build directory.

The client application can be modified and rebuilt from this directory at any time unless changes are made that impact the SAL. For example if using the Miracl+TLSECC SAL, then

```
g++ -O2 client.cpp tls*.cpp libtiitls.a core.a tlsecc.a -o client
```

For example you can edit the file [tls1_3.h](#) and set VERBOSITY to IO_DEBUG, and re-compile as above.

To see the Security Abstraction Layer (SAL) capabilities, navigate to the build directory

```
./client -s
```

To connect to a Website

```
./client swifttls.org
```

The output should (if VERBOSITY has been set to IO_DEBUG in [tls1_3.h](#)) look something like this

```
Hostname= swifttls.org
Private key= 0373AF7D060E0E80959254DC071A068FCBEDA5F0C1B6FFFC02C7EB56AE6B00CD
Client Public key= 93CDD4247C90CBC1920E53C4333BE444C0F13E96A077D8D1EF485FE0F9D9D703
Client Hello sent
Cipher Suite is TLS_AES_128_GCM_SHA256
Server HelloRetryRequest= 020000540303CF21AD74E59A6111BE1D8C021E65B891C2A211167ABB8C5E079E09E2C8A8339C20557742
Client Hello re-sent
Server Hello= 020000970303268C697006F0AC66287680A88C6DB34C2804CD9884B2B0BD087A0F3DE2495F5120A0E658C6A5BB912768
Parsing serverHello
Cipher Suite is TLS_AES_128_GCM_SHA256
Key Exchange Group is SECP256R1
Server Public Key= 04F87B11F808F92B9D4DAE8AE83389257F04B3697181F3CD1479B7214E7D76B108B650A57494D15C5F673EDB05D

Shared Secret= 99A5F3B6F8BE0938AB6D74A99E8FD42DEFD71F25445BD703F0D429DA6CC4AA12
Handshake Secret= 093388E25C3F8468DF3A0544683036CBACF5157874CE995C080807559834CBCA
```

```

Client handshake traffic secret= 5B383ED973C7324E267B16A1A7507C380846FFB5397B41E3199C305C23A2C430
Server handshake traffic secret= 71A23E7184F1AA8F228504D3FA735EC8E70FFEC54E0922D553A64800A32C2853
Warning - ALPN extension NOT acknowledged by server
Server Name NOT acknowledged
Max frag length request NOT acknowledged
Encrypted Extensions Processed
Certificate Chain Length= 2458
Parsing Server certificate
Signature is 0A5C155DB6DD9F7F6ABE005D351D6E3FF9DEBA799F7479BD33E1C784B63CB4CA695A76815C9B666C24B6E989EE85009A6
RSA signature of length 2048
Public key= E2AB76AE1A676E3268E39BB9B8AE9CA19DD8BC0BFED0A4275E13C191D716794B48F47766A6B6AD17F19764F48D459E8271
RSA public key of length 2048
Issuer is R3/Let's Encrypt/
Subject is swifttls.org//
Parsing Intermediate certificate
Signature is D94CE0C9F584883731DBBB13E2B3FC8B6B62126C58B7497E3C02B7A81F2861EBCEE02E73EF49077A35841F1DAD68F0D8F
RSA signature of length 2048
Public key= BB021528CCF6A094D30F12EC8D5592C3F882F199A67A4288A75D26AAB52BB9C54CB1AF8E6BF975C8A3D70F479414553557
RSA public key of length 2048
Issuer is DST Root CA X3/Digital Signature Trust Co./
Subject is R3/Let's Encrypt/
Signature = 0A5C155DB6DD9F7F6ABE005D351D6E3FF9DEBA799F7479BD33E1C784B63CB4CA695A76815C9B666C24B6E989EE85009A6
Public key = BB021528CCF6A094D30F12EC8D5592C3F882F199A67A4288A75D26AAB52BB9C54CB1AF8E6BF975C8A3D70F47941455355
Checking Signature on Cert
Signature Algorithm is RSA_PKCS1_SHA256
Cert Signature Verification succeeded
Intermediate Certificate Chain sig is OK

Public Key from root cert= DFAFE99750088357B4CC6265F69082ECC7D32C6B30CA5BECD9C37DC740C118148BE0E83376492AE33F2
Signature = D94CE0C9F584883731DBBB13E2B3FC8B6B62126C58B7497E3C02B7A81F2861EBCEE02E73EF49077A35841F1DAD68F0D8F
Public key = DFAFE99750088357B4CC6265F69082ECC7D32C6B30CA5BECD9C37DC740C118148BE0E83376492AE33F214993AC4E0EAF3
Checking Signature on Cert
Signature Algorithm is RSA_PKCS1_SHA256
Cert Signature Verification succeeded
Root Certificate sig is OK
Certificate Chain is valid
Transcript Hash (CH+SH+EE+CT) = 7CECF69D794C20FB7551BA5C4B986E1F501011328225CDD740A8EB54B728E31B
Transcript Hash (CH+SH+EE+SCT+SCV) = 8EC0EE587717BAEB401992622E3F31CBE151CC6C489104E68B5A83E96284E1E7
Server Certificate Signature= B5B74CF6026CF16FA866BA7E7562C53F67A74949FF040319B0BD2149CF4EF97CAD482463F1746D20
Signature Algorithm is RSA_PSS_RSAE_SHA256
Server Cert Verification OK

Server Data is verified
Transcript Hash (CH+SH+EE+SCT+SCV+SF+[CCT+CSV]) = 299C505CBD66E8CCCF1934AC5398EFAB7DCF239D9A9C95CF0A5384B5902E
Client Verify Data= 9D20AD7C24238C5B77B72D40EC355C41C5859B6851639EA9920986EDF50DF032
Transcript Hash (CH+SH+EE+SCT+SCV+SF+[CCT+CSV]+CF) = 50AC5EA2A163FD5A3CE92D7D98E8CB56D763514148A30213784612F9E
Client application traffic secret= 7DE3D4B470FBCA72FEECBA1A1B938F4AF85F0E4D84C8E06E4218A92DF3EE67CF
Server application traffic secret= 11FFA6345BE788BBF8C1948E4F499D852A07A77B74C74F560BC9E399AB41ABC8
Full Handshake concluded
... after handshake resumption
Sending Application Message

GET / HTTP/1.1
Host: swifttls.org

Waiting for Server input
Got a ticket
Waiting for Server input
Receiving application data (truncated HTML) = 485454502F312E3120323030204F4B0D0A5365727665723A2053776966745440
Alert sent to Server - Close notify
Connection closed

```

To attempt a fast resumption, based on a resumption ticket (generated and stored in a file cookie.txt), connect again

```
./client swifttls.org
```

The output should look something like

```

Attempting resumption
Hostname= swifttls.org

Parsing Ticket
Ticket = 6CE7CD561F03F6E3CDD9A0DD4A7F37181861F51A17E8FF6930AAA02C6C5DAFD9
life time in minutes = 3600
Pre-Shared Key = 41301AAD7DAADCF43D700CD71E1198DD2C8DFF5C61B91BEA35116B96762C8B7E
max_early_data = 40960

PSK= 41301AAD7DAADCF43D700CD71E1198DD2C8DFF5C61B91BEA35116B96762C8B7E
Binder Key= 3CC796B38A7FEB226D9B0CD6B6BB4994253298DDF9FF43060C5C30834D75EE79
Early Secret= 610B9D95E512F6E199046C93E600D5CE10BB98517F9A81096E653C13B2D0F17D
Private key= 7373AF7D060E0E80959254DC071A06905E067B07367C49D86B48A10F3923CC49
Client Public key= 04EA04CDA74C1A1942BB8C56C0BD8AE1A4CB9D9B76B5AC64C24CFE7C367B46FA6F06037D945835019D3F1220803
Ticket age= feff
obfuscated age = 447e2e62
Client Hello sent
BND= 258FA2CE9D69253C83646641266B2A81FCEED47348D60E0C7BBB27D2557D1BD2
Sending Binders
Client Early Traffic Secret= CF7D980E8213205CFD35C2194FB75F6D1E98215860BB1F7FA5CFDC8DAE48E9F5
Sending some early data
Sending Application Message

GET / HTTP/1.1
Host: swifttls.org

Parsing serverHello
Cipher Suite is TLS_AES_128_GCM_SHA256
Key Exchange Group is SECP256R1
PSK Identity= 0
Server Public Key= 0401D908F018811AF140E2D417EB2713492C146C2B73F78A81DEC6C3F6E2A31D5114207D93EC92AEB03D64DAD11

serverHello= 0200009D0303268C69B38026464DFFE72A496662627EC35798DA3F98437042E39CAF404C888520557742FB051C8ADC0A4
Shared Secret= 8C7784C539C0144B8FADCBF065637418F190C49995E79660919E204F05287C2D
Handshake Secret= 4025A7EE2C1B634C9FC83FDF5CFB2FCB5498EA3F5D019EEDC6D3C1D751C87C47
Client handshake traffic secret= 5FC1307F4E7ED84B4196B83EA19D69724812C25A571061FB53B5B6E9FD7FCABE
Server handshake traffic secret= 1E84FEBA7F8D75F756408906C608925F9A6445292BA614BB398E634CF5854B2A
Early Data Accepted
Warning - ALPN extension NOT acknowledged by server
Server Name NOT acknowledged
Max frag length request NOT acknowledged
Transcript Hash (CH+SH+EE) = DCB73D7B5416D91546EF7D625FBB6A84105CCCE5F054D753275325A822D394E9
Send End of Early Data
Transcript Hash (CH+SH+EE+SF+ED) = FE1FADC8085B3B41A9146647FC9A40F6F2A303533B237112564A2F51F82B64C4
Server Data is verified
Client Verify Data= 350E968A15D36F16BC20D80789E9DB2792A2975765F9BE537407165F7E7366B8
Client application traffic secret= 536F912C98CF4C2D9672DEA57AC8136519607014EFEBBA289FCED97929EA9633
Server application traffic secret= 6B797DBC7FB2D9F75A877F1D34EE7CACC6D65C847C085331F8941C81F2884E83
Resumption Handshake concluded
Early data was accepted
Waiting for Server input
Receiving application data (truncated HTML) = 485454502F312E3120323030204F4B0D0A5365727665723A2053776966745440
Alert sent to Server - Close notify
Connection closed

```

Try it out on your favourite websites. It will abort if TLS1.3 is not supported. In a small number of cases it will fail due to receiving a malformed certificate chain from the Server. It is not forgiving of badly formed certificate chains, and makes no attempt to fix them.

Also try

```
./client tls13.1d.pw
```

Try it a few times - it randomly asks for a HelloRetryRequest and a Key Update, testing this code (but it does not allow resumption). This site also requires that short records are not padded (that is PAD_SHORT_RECORDS is not defined in [tls1_3.h](#)).

A resumption ticket can be deleted by

```
./client -r
```

See doc/list.txt for some websites that work OK and test different functionality.

1.2.1 Client side Authentication

A self-signed client certificate and private key can be generated by

```
openssl req -x509 -nodes -days 365 -newkey ec:<(openssl ecparam -name secp256r1) -keyout mykey.pem -out mycert
```

and inserted into the file *tls_client_cert.cpp*. Or use the example certificate that is there already.

A way to test less common options is to set up a local openssl server. First generate a self-signed server certificate using something like

```
openssl req -x509 -nodes -newkey rsa:2048 -keyout key.pem -out cert.pem -days 365
```

then for example

```
openssl s_server -tls1_3 -debug -key key.pem -cert cert.pem -accept 4433 -www
```

acts as a normal Website, while

```
openssl s_server -tls1_3 -debug -verify 0 -key key.pem -cert cert.pem -accept 4433 -www
```

looks for client side certificate authentication - and the server makes a Certificate Request to the client. We can't control the openssl debug output, but its better than nothing! The client connects to this local server via

```
./client localhost
```

1.2.2 Testing Pre-shared keys

Again we will use OpenSSL to mimic a TLS1.3 server

```
openssl s_server -tls1_3 -cipher PSK-AES128-GCM-SHA256 -psk_identity 42 -psk 0102030405060708090a0b0c0d0e0f10
```

and connect via

```
./client -p 42 localhost
```

An important setting in [tls1_3.h](#) is CRYPTO_SETTING. For the above tests it is assumed that this is set to the default TYPICAL, which allows interaction with standard websites. However it may also be set to TINY_ECC, EDDSA, POST_QUANTUM and HYBRID. This setting (supported for interaction with our own rust server) impacts code size and memory resource allocation. It also controls the type of the self-signed certificate provided by the client if it is asked to authenticate. Remember that any changes like this which impact the SAL requires a fresh build.

Note that the HYBRID setting now works using X25519+MLKEM768 for key exchange with an OpenSSL server, and some online servers like www.cloudflare.com

The client preference for key exchange algorithms, and their preferred ordering, is set in the sal (*tls_sal.cpp*). The chosen CRYPTO_SETTING impacts on this ordering. With the default setting the X25519 elliptic curve is preferred.

To test our IBE version of TLS, simply run the TiigerTLS rust server (not openssl!), and then

```
./client -i localhost
```


Chapter 2

Configure the Arduino Nano RP2040

This build is specifically for the Arduino Nano version of the Raspberry Pi Pico (RP2040). Please use version 2.x.x of the Arduino IDE.

First the board needs to be initialised and locked. To do this install the ArduinoECCX08 library and run the ECCX08SelfSignedCert example program.

(This example program appears when an MKR1000 board is suggested, and may not appear for the RP2040. However it runs fine on the RP2040).

This program (a) locks the board, and (b) generates a self-signed X.509 certificate, with an associated private key hidden in Slot 0. Copy the self-signed certificate and place it into `tls_client_cert.cpp` where indicated.

Note that the ECC608A chip does a lot of the heavy crypto lifting, especially if the `secp256r1` curve is used for certificate signature verification.

The key exchange secret is generated in Slot 1. Slot 9 is used for the HMAC calculation. See the ECC608A documentation for more detail.

2.1 Building the client application on the Arduino Nano RP2040 board.

1. Create working directory with name `tiitls`
2. Copy in all from the `cpp` directory of <https://github.com/miracl/core>
3. Copy in all from the `arduino` directory of <https://github.com/miracl/core>
4. (If ever asked to overwrite a file, go ahead and overwrite it)
5. Copy in all of the TLS1.3 C++ code from the `lib/`, `lib/ibe`, `include/`, `sal/` and `src/arduino` directories (but not from subdirectories)
6. Edit the file `core.h` to define `CORE_ARDUINO` (line 31)
7. Edit the file [tls_octads.h](#) to define `TLS_ARDUINO` (line 13).
8. Edit [tls1_3.h](#). Define `VERBOSITY` as `IO_DEBUG` for more debug output. Decide on `CRYPTO_SETTING`. Stack only, or Stack plus heap.
9. Edit the file `client.cpp` to set your wifi SSID and password (near line 150)

10. Run `python3 config.py`, and select options 2, 8, 31, 42 and 44. This creates the default SAL (in this case using `miracl` + ECC608A hardware).
11. Drop the working directory into where the Arduino IDE expects it.
12. (In the IDE select File->Preferences and find the Sketchbook location - its the libraries directory off that.)
13. Open the Arduino app, and look in File->Examples->tiitls, and look for the example "client"
14. Upload to the board and run it. Open Tools->Serial Monitor to see the output.
15. Enter URL (e.g. `www.bbc.co.uk`) when prompted, and press return. A full TLS1.3 handshake followed by a resumption is attempted.
16. Click on Clear Output and Send to repeat for a different URL (or click Send again to see SAL capabilities).

or before executing step 10, search for `!!!!!!!` in `config.py` (around line 1020) and make changes as indicated. If using `miracl` alone, without hardware support, option 3 must be selected as well. If using assembly language code for X25519, copy `x25519.S` from <https://github.com/pornin/x25519-cm0/blob/main/src/x25519-cm0.S> into working directory and remove option 2. This creates the SAL (in this case using `miracl` + ECC608A hardware + Pornin's x25519). If experimenting with post-quantum primitives, also select options 46 and 47, for Dilithium and Kyber support.

or before executing step 10, copy into the project all from `c32`, include and include32 directories of <https://github.com/mcarrickscott/TLSECC>, edit `config.py` at `!!!!!!!` to use faster elliptic curve code from TLSECC project, and select only options 31, 42 and 44. This method works well with the Raspberry Pi Pico 2 W device, which does not support ECC608A hardware. We observe that on this device the crypto is about 1.5 times faster on the ARM core compared to the RISC-V core.

(Sometimes the wifi works better from `wifiNINA.h` rather than `wifi.h`. See [tls_wifi.h](#))

The example TLS1.3 client code first connects to the wireless network, and after that it should connect to standard websites, as long as they support TLS1.3. The example program first makes a full TLS handshake, and exits after receiving some HTML from the server. Then after a few seconds, if it has received a resumption ticket, it attempts a resumption handshake.

The client can also be run in conjunction with our Rust server. Make sure that the `CRYPTO_SETTING` parameter is the same for both client and server. In our experimental set-up, the rust server runs directly from Windows (not WSL), looking for connections on port 4433. Run `ipconfig` to get the IP address of the server on the local network, which might look something like 192.168.1.186. Then run the client from the Arduino IDE, and when prompted enter for example 192.168.1.186:4433. The client should now connect to the server. It may however be necessary to undefine `CHECK_NAME_IN_CERT` in [tls1_3.h](#)

Note that some servers may reject a resumption handshake if the device's internal clock is not set to the current time. So the client attempts to use an online time server to access epoch time, and use that to set clock.

Chapter 3

Data Structure Index

3.1 Data Structures

Here are the data structures with brief descriptions:

credential	Client credential structure	13
crypto	Crypto context structure	15
ECCX08Class	17
ee_status	Server encrypted extensions expectations/responses	17
octad	Safe representation of an octad	18
pktype	Public key type	19
ret	Function return structure	20
Socket	Socket instance	21
ticket	Ticket context structure	21
TLS_session	TLS1.3 session state	24
unihash	Universal Hash Function	29

Chapter 4

File Index

4.1 File List

Here is a list of all documented files with brief descriptions:

tls1_3.h	Main TLS 1.3 Header File for constants and structures	31
tls_bfibe.h	Boneh and Franklin IBE	67
tls_cert_chain.h	Process Certificate Chain	69
tls_certs.h	Certificate Authority root certificate store	70
tls_client_recv.h	Process Input received from the Server	72
tls_client_send.h	Process Output to be sent to the Server	80
tls_keys_calc.h	TLS 1.3 crypto support functions	93
tls_logger.h	TLS 1.3 logging	103
tls_octads.h	Octad handling routines - octads don't overflow, they truncate	108
tls_pqibe.h	Ducas et al. IBE	115
tls_protocol.h	TLS 1.3 main client-side protocol functions	117
tls_sal.h	Security Abstraction Layer for TLS	121
tls_sockets.h	Set up sockets for reading and writing	132
tls_tickets.h	TLS 1.3 process resumption tickets	137
tls_wifi.h	Define Socket structure depending on processor context	139
tls_x509.h	X509 function Header File	140
ECCX08.h	152

Chapter 5

Data Structure Documentation

5.1 credential Struct Reference

client credential structure

```
#include <tls1_3.h>
```

Data Fields

- char [certchain](#) [TLS_MAX_CLIENT_CHAIN_SIZE]
- char [publickey](#) [TLS_MAX_SIG_PUB_KEY_SIZE]
- char [secretkey](#) [TLS_MAX_SIG_SECRET_KEY_SIZE]
- [octad CERTCHAIN](#)
- [octad PUBLICKEY](#)
- [octad SECRETKEY](#)
- [unsign16 requirements](#) [16]
- int [nreqs](#)
- int [nreqsraw](#)
- int [sigalg](#)

5.1.1 Detailed Description

client credential structure

5.1.2 Field Documentation

5.1.2.1 certchain

```
char credential::certchain[TLS_MAX_CLIENT_CHAIN_SIZE]
```

Client 1 or 2-link Certificate chain

5.1.2.2 publickey

```
char credential::publickey[TLS_MAX_SIG_PUB_KEY_SIZE]
```

Client public key extracted from certificate chain

5.1.2.3 secretkey

```
char credential::secretkey[TLS_MAX_SIG_SECRET_KEY_SIZE]
```

Client private key

5.1.2.4 CERTCHAIN

```
octad credential::CERTCHAIN
```

certchain as octad

5.1.2.5 PUBLICKEY

```
octad credential::PUBLICKEY
```

public key as octad

5.1.2.6 SECRETKEY

```
octad credential::SECRETKEY
```

private key as octad

5.1.2.7 requirements

```
unsigned short credential::requirements[16]
```

signature verification algorithms that will be needed by the server

5.1.2.8 nreqs

```
int credential::nreqs
```

Number of such verification algorithms required

5.1.2.9 nreqsraw

```
int credential::nreqsraw
```

Number required if using raw public keys

5.1.2.10 sigalg

```
int credential::sigalg
```

signature algorithm client will use to sign TLS1.3 transcript

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

- [tls1_3.h](#)

5.2 crypto Struct Reference

crypto context structure

```
#include <tls1_3.h>
```

Data Fields

- bool [active](#)
- char [k](#) [TLS_MAX_KEY]
- char [iv](#) [12]
- octad [K](#)
- octad [IV](#)
- [unsign32](#) [record](#)
- int [suite](#)
- int [taglen](#)

5.2.1 Detailed Description

crypto context structure

5.2.2 Field Documentation

5.2.2.1 active

```
bool crypto::active
```

Indicates if encryption has been activated

5.2.2.2 k

```
char crypto::k[TLS_MAX_KEY]
```

AEAD cryptographic Key bytes

5.2.2.3 iv

```
char crypto::iv[12]
```

AEAD cryptographic IV bytes

5.2.2.4 K

```
octad crypto::K
```

Key as octad

5.2.2.5 IV

```
octad crypto::IV
```

IV as octad

5.2.2.6 record

```
unsign32 crypto::record
```

current record number - to be incremented

5.2.2.7 suite

```
int crypto::suite
```

Cipher Suite

5.2.2.8 taglen

```
int crypto::taglen
```

Tag Length

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

- [tls1_3.h](#)

5.3 ECCX08Class Class Reference

Public Member Functions

- **ECCX08Class** (TwoWire &wire, uint8_t address)
- int **begin** ()
- void **end** ()
- int **serialNumber** (byte sn[])
- String **serialNumber** ()
- long **random** (long max)
- long **random** (long min, long max)
- int **random** (byte data[], size_t length)
- int **generatePrivateKey** (int slot, byte publicKey[])
- int **generatePublicKey** (int slot, byte publicKey[])
- int **generateSharedKey** (int slot, byte publicKey[], byte sharedKey[])
- int **ecdsaVerify** (const byte message[], const byte signature[], const byte pubkey[])
- int **ecSign** (int slot, const byte message[], byte signature[])
- int **challenge** (const byte message[])
- int **aesEncrypt** (byte block[])
- int **aesGFM** (byte state[], byte H[])
- int **beginSHA256** ()
- int **beginHMAC** (int slot)
- int **updateSHA256** (const byte data[], int len)
- int **endSHA256** (byte result[])
- int **endSHA256** (const byte data[], int length, byte result[])
- int **readSHA256** (byte context[])
- int **writeSHA256** (byte context[], int length)
- int **readSlot** (int slot, byte data[], int length)
- int **writeSlot** (int slot, const byte data[], int length)
- int **locked** ()
- int **writeConfiguration** (const byte data[])
- int **readConfiguration** (byte data[])
- int **lock** ()

The documentation for this class was generated from the following files:

- ECCX08.h
- ECCX08.cpp

5.4 ee_status Struct Reference

server encrypted extensions expectations/responses

```
#include <tls1_3.h>
```

Data Fields

- bool **early_data**
- bool **alpn**
- bool **server_name**
- bool **max_frag_length**

5.4.1 Detailed Description

server encrypted extensions expectations/responses

5.4.2 Field Documentation

5.4.2.1 early_data

```
bool ee_status::early_data
```

true if early data accepted

5.4.2.2 alpn

```
bool ee_status::alpn
```

true if ALPN accepted

5.4.2.3 server_name

```
bool ee_status::server_name
```

true if server name accepted

5.4.2.4 max_frag_length

```
bool ee_status::max_frag_length
```

true if max frag length respected

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

- [tls1_3.h](#)

5.5 octad Struct Reference

Safe representation of an octad.

```
#include <tls_octads.h>
```

Data Fields

- int [len](#)
- int [max](#)
- char * [val](#)

5.5.1 Detailed Description

Safe representation of an octad.

5.5.2 Field Documentation

5.5.2.1 len

```
int octad::len
```

length in bytes

5.5.2.2 max

```
int octad::max
```

max length allowed - enforce truncation

5.5.2.3 val

```
char* octad::val
```

byte array

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

- [tls_octads.h](#)

5.6 pktype Struct Reference

Public key type.

```
#include <tls_x509.h>
```

Data Fields

- int [type](#)
- int [hash](#)
- int [curve](#)

5.6.1 Detailed Description

Public key type.

5.6.2 Field Documentation

5.6.2.1 type

```
int pktype::type
```

signature type (ECC or RSA)

5.6.2.2 hash

```
int pktype::hash
```

hash type

5.6.2.3 curve

```
int pktype::curve
```

elliptic curve used or RSA key length in bits

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

- [tls_x509.h](#)

5.7 ret Struct Reference

function return structure

```
#include <tls1_3.h>
```

Data Fields

- [unsign32 val](#)
- [int err](#)

5.7.1 Detailed Description

function return structure

5.7.2 Field Documentation

5.7.2.1 val

```
unsign32 ret::val
```

return value

5.7.2.2 err

```
int ret::err
```

error return

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

- [tls1_3.h](#)

5.8 Socket Class Reference

[Socket](#) instance.

```
#include <tls_sockets.h>
```

Public Member Functions

- bool **connect** (char *host, int port)
- int **getport** ()
- void **setTimeout** (int to)
- int **write** (char *buf, int len)
- int **read** (char *buf, int len)
- void **stop** ()

Static Public Member Functions

- static [Socket](#) **InetSocket** ()
- static [Socket](#) **UnixSocket** ()

5.8.1 Detailed Description

[Socket](#) instance.

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

- [tls_sockets.h](#)

5.9 ticket Struct Reference

ticket context structure

```
#include <tls1_3.h>
```

Data Fields

- bool [valid](#)
- char [tick](#) [[TLS_MAX_TICKET_SIZE](#)]
- char [nonce](#) [256]
- char [psk](#) [[TLS_MAX_HASH](#)]
- octad [TICK](#)
- octad [NONCE](#)
- octad [PSK](#)
- [unsign32](#) [age_obfuscator](#)
- [unsign32](#) [max_early_data](#)
- [unsign32](#) [birth](#)
- int [lifetime](#)
- int [cipher_suite](#)
- int [favourite_group](#)
- int [origin](#)

5.9.1 Detailed Description

ticket context structure

5.9.2 Field Documentation

5.9.2.1 [valid](#)

```
bool ticket::valid
```

Is ticket valid?

5.9.2.2 [tick](#)

```
char ticket::tick[TLS\_MAX\_TICKET\_SIZE]
```

Ticket bytes

5.9.2.3 [nonce](#)

```
char ticket::nonce[256]
```

nonce

5.9.2.4 [psk](#)

```
char ticket::psk[TLS\_MAX\_HASH]
```

pre-shared key

5.9.2.5 TICK

```
octad ticket::TICK
```

Ticket or external PSK label as octad

5.9.2.6 NONCE

```
octad ticket::NONCE
```

Nonce as octad

5.9.2.7 PSK

```
octad ticket::PSK
```

PSK as octad

5.9.2.8 age_obfuscator

```
unsign32 ticket::age_obfuscator
```

ticket age obfuscator - 0 for external PSK

5.9.2.9 max_early_data

```
unsign32 ticket::max_early_data
```

Maximum early data allowed for this ticket

5.9.2.10 birth

```
unsign32 ticket::birth
```

Birth time of this ticket

5.9.2.11 lifetime

```
int ticket::lifetime
```

ticket lifetime

5.9.2.12 cipher_suite

```
int ticket::cipher_suite
```

Cipher suite used

5.9.2.13 favourite_group

```
int ticket::favourite_group
```

the server's favourite group

5.9.2.14 origin

```
int ticket::origin
```

Origin of initial handshake - Full or PSK?

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

- [tls1_3.h](#)

5.10 TLS_session Struct Reference

TLS1.3 session state.

```
#include <tls1_3.h>
```

Data Fields

- int [status](#)
- int [max_record](#)
- [Socket](#) * [sockptr](#)
- char [id](#) [32]
- char [hostname](#) [TLS_MAX_SERVER_NAME]
- int [cipher_suite](#)
- int [favourite_group](#)
- int [server_cert_type](#)
- int [client_cert_type](#)
- [crypto](#) K_send
- [crypto](#) K_rcv
- [octad](#) HS
- char [hs](#) [TLS_MAX_HASH]
- [octad](#) RMS
- char [rms](#) [TLS_MAX_HASH]
- [octad](#) STS
- char [sts](#) [TLS_MAX_HASH]
- [octad](#) CTS
- char [cts](#) [TLS_MAX_HASH]
- [octad](#) CTX
- char [ctx](#) [TLS_MAX_HASH]
- [octad](#) IBUFF
- [octad](#) OBUFF
- char [ibuff](#) [TLS_MAX_IBUFF_SIZE]
- char [obuff](#) [TLS_MAX_OBUFF_SIZE]
- int [ptr](#)
- [unihash](#) [tlshash](#)
- [ticket](#) T
- bool [expect_heartbeats](#)
- bool [allowed_to_heartbeat](#)
- bool [heartbeat_req_in_flight](#)

5.10.1 Detailed Description

TLS1.3 session state.

5.10.2 Field Documentation

5.10.2.1 status

```
int TLS_session::status
```

Connection status

5.10.2.2 max_record

```
int TLS_session::max_record
```

max record size I should send

5.10.2.3 sockptr

```
Socket* TLS_session::sockptr
```

Pointer to socket

5.10.2.4 id

```
char TLS_session::id[32]
```

Session ID

5.10.2.5 hostname

```
char TLS_session::hostname[TLS_MAX_SERVER_NAME]
```

Server name for connection

5.10.2.6 cipher_suite

```
int TLS_session::cipher_suite
```

agreed cipher suite

5.10.2.7 favourite_group

```
int TLS_session::favourite_group
```

favourite key exchange group - may be changed on handshake retry

5.10.2.8 server_cert_type

```
int TLS_session::server_cert_type
```

server certificate type

5.10.2.9 client_cert_type

```
int TLS_session::client_cert_type
```

client certificate type

5.10.2.10 K_send

```
crypto TLS_session::K_send
```

Sending Key

5.10.2.11 K_rcv

```
crypto TLS_session::K_rcv
```

Receiving Key

5.10.2.12 HS

```
octad TLS_session::HS
```

Handshake secret

5.10.2.13 hs

```
char TLS_session::hs[TLS_MAX_HASH]
```

Handshake secret data

5.10.2.14 RMS

```
octad TLS_session::RMS
```

Resumption Master Secret

5.10.2.15 rms

```
char TLS_session::rms[TLS_MAX_HASH]
```

Resumption Master Secret data

5.10.2.16 STS

```
octad TLS_session::STS
```

Server Traffic secret

5.10.2.17 sts

```
char TLS_session::sts[TLS_MAX_HASH]
```

Server Traffic secret data

5.10.2.18 CTS

```
octad TLS_session::CTS
```

Client Traffic secret

5.10.2.19 cts

```
char TLS_session::cts[TLS_MAX_HASH]
```

Client Traffic secret data

5.10.2.20 CTX

```
octad TLS_session::CTX
```

Certificate Request Context

5.10.2.21 ctx

```
char TLS_session::ctx[TLS_MAX_HASH]
```

Certificate Request Context data

5.10.2.22 Ibuff

```
octad TLS_session::Ibuff
```

Main input buffer for this connection

5.10.2.23 OBUFF

```
octad TLS_session::OBUFF
```

output buffer for this connection

5.10.2.24 ibuff

```
char TLS_session::ibuff[TLS_MAX_IBUFF_SIZE]
```

Byte array for main input buffer for this connection

5.10.2.25 obuff

```
char TLS_session::obuff[TLS_MAX_OBUFF_SIZE]
```

output buffer for this connection

5.10.2.26 ptr

```
int TLS_session::ptr
```

pointer into IBUFF buffer

5.10.2.27 tlshash

```
unihash TLS_session::tlshash
```

Transcript hash recorder

5.10.2.28 T

```
ticket TLS_session::T
```

resumption ticket

5.10.2.29 expect_heartbeats

```
bool TLS_session::expect_heartbeats
```

Expect to receive heartbeats

5.10.2.30 allowed_to_heartbeat

```
bool TLS_session::allowed_to_heartbeat
```

Am I allowed to heartbeat?

5.10.2.31 heartbeat_req_in_flight

```
bool TLS_session::heartbeat_req_in_flight
```

timestamp on outstanding request, otherwise 0

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

- [tls1_3.h](#)

5.11 unihash Struct Reference

Universal Hash Function.

```
#include <tls1_3.h>
```

Data Fields

- char [state](#) [TLS_MAX_HASH_STATE]
- int [htype](#)

5.11.1 Detailed Description

Universal Hash Function.

5.11.2 Field Documentation

5.11.2.1 state

```
char unihash::state[TLS_MAX_HASH_STATE]
```

hash function state

5.11.2.2 htype

```
int unihash::htype
```

The hash type (typically SHA256)

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

- [tls1_3.h](#)

Chapter 6

File Documentation

6.1 tls1_3.h File Reference

Main TLS 1.3 Header File for constants and structures.

```
#include <stdint.h>
#include "tls_octads.h"
#include "tls_sockets.h"
```

Data Structures

- struct [ret](#)
function return structure
- struct [ee_status](#)
server encrypted extensions expectations/responses
- struct [crypto](#)
crypto context structure
- struct [credential](#)
client credential structure
- struct [ticket](#)
ticket context structure
- struct [unihash](#)
Universal Hash Function.
- struct [TLS_session](#)
TLS1.3 session state.

Macros

- #define [IO_NONE](#) 0
- #define [IO_APPLICATION](#) 1
- #define [IO_PROTOCOL](#) 2
- #define [IO_DEBUG](#) 3
- #define [IO_WIRE](#) 4
- #define [TINY_ECC](#) 0
- #define [TYPICAL](#) 1

- `#define POST_QUANTUM 3`
- `#define HYBRID 4`
- `#define EDDSA 2`
- `#define NO_CERT 1`
- `#define FROM_FILE 2`
- `#define FROM_ROM 3`
- `#define RSA_SS 1`
- `#define ECC_SS 2`
- `#define DLT_SS 3`
- `#define HYB_SS 6`
- `#define HW_1 4`
- `#define HW_2 5`
- `#define EDD_SS 7`
- `#define CRYPTO_SETTING TYPICAL`
- `#define VERBOSITY IO_PROTOCOL`
- `#define CLIENT_CERT FROM_ROM`
- `#define CLIENT_CERT_KIND ECC_SS`
- `#define POST_HS_AUTH`
- `#define TLS_APPLICATION_PROTOCOL (char *)("http/1.1")`
- `#define ALLOW_SELF_SIGNED`
- `#define TRY_EARLY_DATA`
- `#define MERGE_MESSAGES`
- `#define CHECK_NAME_IN_CERT`
- `#define TLS_SHA256_T 1`
- `#define TLS_SHA384_T 2`
- `#define TLS_SHA512_T 3`
- `#define TLS_MAX_HASH_STATE 768`
- `#define TLS_MAX_HASH 64`
- `#define TLS_MAX_KEY 32`
- `#define TLS_X509_MAX_FIELD 256`
- `#define TLS_MAX_EXT_LABEL 256`
- `#define TLS_MAX_FRAG 2`
- `#define TLS_MAX_IBUFF_SIZE (16384+256)`
- `#define TLS_MAX_PLAIN_FRAG 16384`
- `#define TLS_MAX_CIPHER_FRAG (16384+256)`
- `#define TLS_MAX_CERT_SIZE 2048`
- `#define TLS_MAX_CERT_B64 2800`
- `#define TLS_MAX_HELLO 1024`
- `#define TLS_MAX_SIG_PUB_KEY_SIZE 512`
- `#define TLS_MAX_SIG_SECRET_KEY_SIZE 1280`
- `#define TLS_MAX_SIGNATURE_SIZE 512`
- `#define TLS_MAX_KEX_PUB_KEY_SIZE 97`
- `#define TLS_MAX_KEX_CIPHERTEXT_SIZE 97`
- `#define TLS_MAX_KEX_SECRET_KEY_SIZE 48`
- `#define TLS_MAX_SERVER_CHAIN_LEN 2`
- `#define TLS_MAX_SERVER_CHAIN_SIZE (TLS_MAX_SERVER_CHAIN_LEN*TLS_MAX_CERT_SIZE)`
- `#define TLS_MAX_CLIENT_CHAIN_LEN 1`
- `#define TLS_MAX_CLIENT_CHAIN_SIZE (TLS_MAX_CLIENT_CHAIN_LEN*TLS_MAX_CERT_SIZE)`
- `#define TLS_MAX_SHARED_SECRET_SIZE 256`
- `#define TLS_MAX_TICKET_SIZE 4196`
- `#define TLS_MAX_EXTENSIONS 6144`
- `#define TLS_MAX_ECC_FIELD 66`
- `#define TLS_MAX_IV_SIZE 12`
- `#define TLS_MAX_TAG_SIZE 16`
- `#define TLS_MAX_COOKIE 128`

- `#define TLS_MAX_OUTPUT_RECORD_SIZE` 1024
- `#define TLS_MAX_OBUFF_SIZE` (TLS_MAX_OUTPUT_RECORD_SIZE+TLS_MAX_TAG_SIZE+6)
- `#define TLS_MAX_SERVER_NAME` 128
- `#define TLS_MAX_SUPPORTED_GROUPS` 10
- `#define TLS_MAX_SUPPORTED_SIGS` 32
- `#define TLS_MAX_PSK_MODES` 2
- `#define TLS_MAX_CIPHER_SUITES` 5
- `#define TLS_AES_128_GCM_SHA256` 0x1301
- `#define TLS_AES_256_GCM_SHA384` 0x1302
- `#define TLS_CHACHA20_POLY1305_SHA256` 0x1303
- `#define TLS_AES_128_CCM_SHA256` 0x1304
- `#define TLS_AES_128_CCM_8_SHA256` 0x1305
- `#define X25519` 0x001d
- `#define SECP256R1` 0x0017
- `#define SECP384R1` 0x0018
- `#define SECP521R1` 0x0019
- `#define X448` 0x001e
- `#define MLKEM512` 0x0200
- `#define MLKEM768` 0x0201
- `#define MLKEM1024` 0x0202
- `#define HYBRID_KX` 0x11ec
- `#define ECDSA_SECP256R1_SHA256` 0x0403
- `#define ECDSA_SECP256R1_SHA384` 0x0413
- `#define ECDSA_SECP384R1_SHA384` 0x0503
- `#define RSA_PSS_RSAE_SHA256` 0x0804
- `#define RSA_PSS_RSAE_SHA384` 0x0805
- `#define RSA_PSS_RSAE_SHA512` 0x0806
- `#define RSA_PKCS1_SHA256` 0x0401
- `#define RSA_PKCS1_SHA384` 0x0501
- `#define RSA_PKCS1_SHA512` 0x0601
- `#define ED25519` 0x0807
- `#define ED448` 0x0808
- `#define MLDSA44` 0x0904
- `#define MLDSA65` 0x0905
- `#define MLDSA87` 0x0906
- `#define MLDSA44_P256` 0xFF06
- `#define PSKOK` 0x00
- `#define PSKWECDHE` 0x01
- `#define TLS_FULL_HANDSHAKE` 1
- `#define TLS_EXTERNAL_PSK` 2
- `#define TLS1_0` 0x0301
- `#define TLS1_2` 0x0303
- `#define TLS1_3` 0x0304
- `#define TLS13_UPDATE_NOT_REQUESTED` 0
- `#define TLS13_UPDATE_REQUESTED` 1
- `#define SERVER_NAME` 0x0000
- `#define SUPPORTED_GROUPS` 0x000a
- `#define SIG_ALGS` 0x000d
- `#define POST_HANDSHAKE_AUTH` 0x0031
- `#define SIG_ALGS_CERT` 0x0032
- `#define KEY_SHARE` 0x0033
- `#define PSK_MODE` 0x002d
- `#define PRESHARED_KEY` 0x0029
- `#define TLS_VER` 0x002b
- `#define COOKIE` 0x002c

- #define [EARLY_DATA](#) 0x002a
- #define [MAX_FRAG_LENGTH](#) 0x0001
- #define [PADDING](#) 0x0015
- #define [APP_PROTOCOL](#) 0x0010
- #define [RECORD_SIZE_LIMIT](#) 0x001c
- #define [CLIENT_CERT_TYPE](#) 0x0013
- #define [SERVER_CERT_TYPE](#) 0x0014
- #define [HEARTBEAT](#) 0x000f
- #define [CERT_AUTHORITIES](#) 0x002f
- #define [HSHAKE](#) 0x16
- #define [APPLICATION](#) 0x17
- #define [ALERT](#) 0x15
- #define [CHANGE_CIPHER](#) 0x14
- #define [HEART_BEAT](#) 0x18
- #define [TIMED_OUT](#) 0x01
- #define [CLIENT_HELLO](#) 0x01
- #define [SERVER_HELLO](#) 0x02
- #define [CERTIFICATE](#) 0x0b
- #define [CERT_REQUEST](#) 0x0d
- #define [CERT_VERIFY](#) 0x0f
- #define [FINISHED](#) 0x14
- #define [ENCRYPTED_EXTENSIONS](#) 0x08
- #define [TICKET](#) 0x04
- #define [KEY_UPDATE](#) 0x18
- #define [MESSAGE_HASH](#) 0xFE
- #define [END_OF_EARLY_DATA](#) 0x05
- #define [HANDSHAKE_RETRY](#) 0x102
- #define [NOT_TLS1_3](#) -2
- #define [BAD_CERT_CHAIN](#) -3
- #define [ID_MISMATCH](#) -4
- #define [UNRECOGNIZED_EXT](#) -5
- #define [BAD_HELLO](#) -6
- #define [WRONG_MESSAGE](#) -7
- #define [MISSING_REQUEST_CONTEXT](#) -8
- #define [AUTHENTICATION_FAILURE](#) -9
- #define [BAD_RECORD](#) -10
- #define [BAD_TICKET](#) -11
- #define [NOT_EXPECTED](#) -12
- #define [CA_NOT_FOUND](#) -13
- #define [CERT_OUTOFDATE](#) -14
- #define [MEM_OVERFLOW](#) -15
- #define [FORBIDDEN_EXTENSION](#) -16
- #define [MAX_EXCEEDED](#) -17
- #define [EMPTY_CERT_CHAIN](#) -18
- #define [SELF_SIGNED_CERT](#) -20
- #define [TIME_OUT](#) -21
- #define [ERROR_ALERT_RECEIVED](#) -22
- #define [BAD_MESSAGE](#) -23
- #define [CERT_VERIFY_FAIL](#) -24
- #define [BAD_HANDSHAKE](#) -26
- #define [BAD_REQUEST_UPDATE](#) -27
- #define [CLOSURE_ALERT_RECEIVED](#) -28
- #define [MISSING_EXTENSIONS](#) -30
- #define [ILLEGAL_PARAMETER](#) 0x2F
- #define [UNEXPECTED_MESSAGE](#) 0x0A

- #define [DECRYPT_ERROR](#) 0x33
- #define [BAD_CERTIFICATE](#) 0x2A
- #define [UNSUPPORTED_EXTENSION](#) 0x6E
- #define [UNKNOWN_CA](#) 0x30
- #define [CERTIFICATE_EXPIRED](#) 0x2D
- #define [PROTOCOL_VERSION](#) 0x46
- #define [DECODE_ERROR](#) 0x32
- #define [RECORD_OVERFLOW](#) 0x16
- #define [BAD_RECORD_MAC](#) 0x14
- #define [HANDSHAKE_FAILURE](#) 0x28
- #define [CLOSE_NOTIFY](#) 0x00
- #define [MISSING_EXTENSION](#) 0x6D;
- #define [LOG_OUTPUT_TRUNCATION](#) 256
- #define [TLS13_DISCONNECTED](#) 0
- #define [TLS13_CONNECTED](#) 1
- #define [TLS13_HANDSHAKING](#) 2
- #define [TLS_FAILURE](#) 0
- #define [TLS_SUCCESS](#) 1
- #define [TLS_RESUMPTION_REQUIRED](#) 2
- #define [TLS_EARLY_DATA_ACCEPTED](#) 3
- #define [PSK_NOT](#) 0
- #define [PSK_KEY](#) 1
- #define [PSK_IBE](#) 2
- #define [X509_CERT](#) 0
- #define [RAW_PUBLIC_KEY](#) 2

Typedefs

- typedef uint8_t [byte](#)
- typedef int8_t [sign8](#)
- typedef int16_t [sign16](#)
- typedef int32_t [sign32](#)
- typedef int64_t [sign64](#)
- typedef uint16_t [unsign16](#)
- typedef uint32_t [unsign32](#)
- typedef uint64_t [unsign64](#)

6.1.1 Detailed Description

Main TLS 1.3 Header File for constants and structures.

Author

Mike Scott

6.1.2 Macro Definition Documentation

6.1.2.1 IO_NONE

```
#define IO_NONE 0
```

Run silently

6.1.2.2 IO_APPLICATION

```
#define IO_APPLICATION 1
```

just print application traffic

6.1.2.3 IO_PROTOCOL

```
#define IO_PROTOCOL 2
```

print protocol progress + application traffic

6.1.2.4 IO_DEBUG

```
#define IO_DEBUG 3
```

print lots of debug information + protocol progress + application progress

6.1.2.5 IO_WIRE

```
#define IO_WIRE 4
```

print lots of debug information + protocol progress + application progress + bytes on the wire

6.1.2.6 TINY_ECC

```
#define TINY_ECC 0
```

ECC keys only

6.1.2.7 TYPICAL

```
#define TYPICAL 1
```

Mixture of RSA and ECC - for use with most standard web servers

6.1.2.8 POST_QUANTUM

```
#define POST_QUANTUM 3
```

Post quantum (MLDSA and MLKEM?)

6.1.2.9 HYBRID

```
#define HYBRID 4
```

Hybrid, MLKEM/MLDSA + X25519

6.1.2.10 EDDSA

```
#define EDDSA 2
```

experimental EDDSA certificate chain used

6.1.2.11 NO_CERT

```
#define NO_CERT 1
```

Don't have a Client Cert

6.1.2.12 FROM_FILE

```
#define FROM_FILE 2
```

certificate and key comes from files

6.1.2.13 FROM_ROM

```
#define FROM_ROM 3
```

certificate and key comes from rom

6.1.2.14 RSA_SS

```
#define RSA_SS 1
```

self signed RSA cert

6.1.2.15 ECC_SS

```
#define ECC_SS 2
```

self signed ECC cert

6.1.2.16 DLT_SS

```
#define DLT_SS 3
```

self signed MLDSA cert

6.1.2.17 HYB_SS

```
#define HYB_SS 6
```

self signed Hybrid cert (MLDSA+ECC)

6.1.2.18 HW_1

```
#define HW_1 4
```

RP2040 1 Hardware cert

6.1.2.19 HW_2

```
#define HW_2 5
```

RP2040 2 Hardware cert

6.1.2.20 EDD_SS

```
#define EDD_SS 7
```

self signed Ed25519 cert

6.1.2.21 CRYPTO_SETTING

```
#define CRYPTO_SETTING TYPICAL
```

Cryptography Setting

6.1.2.22 VERBOSITY

```
#define VERBOSITY IO_PROTOCOL
```

Set to level of output information desired - see above

6.1.2.23 CLIENT_CERT

```
#define CLIENT_CERT FROM_ROM
```

Indicate capability of authenticating with a cert plus signing key (either built-in or from a file)

6.1.2.24 CLIENT_CERT_KIND

```
#define CLIENT_CERT_KIND ECC_SS
```

Choose a certificate - see `tls_clientcert.cpp`

6.1.2.25 POST_HS_AUTH

```
#define POST_HS_AUTH
```

Willing to do post handshake authentication

6.1.2.26 TLS_APPLICATION_PROTOCOL

```
#define TLS_APPLICATION_PROTOCOL (char *)("http/1.1")
```

Support ALPN protocol

6.1.2.27 ALLOW_SELF_SIGNED

```
#define ALLOW_SELF_SIGNED
```

allow self-signed server cert

6.1.2.28 TRY_EARLY_DATA

```
#define TRY_EARLY_DATA
```

Try to send early data on resumptions

6.1.2.29 MERGE_MESSAGES

```
#define MERGE_MESSAGES
```

allow merging of messages into single record

6.1.2.30 CHECK_NAME_IN_CERT

```
#define CHECK_NAME_IN_CERT
```

check name in cert matches hostname, and abort if it does not

6.1.2.31 TLS_SHA256_T

```
#define TLS_SHA256_T 1
```

SHA256 hash

6.1.2.32 TLS_SHA384_T

```
#define TLS_SHA384_T 2
```

SHA384 hash

6.1.2.33 TLS_SHA512_T

```
#define TLS_SHA512_T 3
```

SHA512 hash

6.1.2.34 TLS_MAX_HASH_STATE

```
#define TLS_MAX_HASH_STATE 768
```

Maximum memory required to store hash function state

6.1.2.35 TLS_MAX_HASH

```
#define TLS_MAX_HASH 64
```

Maximum hash output length in bytes

6.1.2.36 TLS_MAX_KEY

```
#define TLS_MAX_KEY 32
```

Maximum key length in bytes

6.1.2.37 TLS_X509_MAX_FIELD

```
#define TLS_X509_MAX_FIELD 256
```

Maximum X.509 field size

6.1.2.38 TLS_MAX_EXT_LABEL

```
#define TLS_MAX_EXT_LABEL 256
```

Max external psk label size

6.1.2.39 TLS_MAX_FRAG

```
#define TLS_MAX_FRAG 2
```

Max Fragment length desired - 1 for 512, 2 for 1024, 3 for 2048, 4 for 4096, 0 for 16384

6.1.2.40 TLS_MAX_IBUFF_SIZE

```
#define TLS_MAX_IBUFF_SIZE (16384+256)
```

Maximum Input/Output buffer size. We will want to reduce this as much as possible! But must be large enough to take full certificate chain

6.1.2.41 TLS_MAX_PLAIN_FRAG

```
#define TLS_MAX_PLAIN_FRAG 16384
```

Max Plaintext Fragment size

6.1.2.42 TLS_MAX_CIPHER_FRAG

```
#define TLS_MAX_CIPHER_FRAG (16384+256)
```

Max Ciphertext Fragment size

6.1.2.43 TLS_MAX_CERT_SIZE

```
#define TLS_MAX_CERT_SIZE 2048
```

I checked - current max for root CAs is 2016

6.1.2.44 TLS_MAX_CERT_B64

```
#define TLS_MAX_CERT_B64 2800
```

In base64 - current max for root CAs is 2688

6.1.2.45 TLS_MAX_HELLO

```
#define TLS_MAX_HELLO 1024
```

Max client hello size (less extensions) KEX public key is largest component

6.1.2.46 TLS_MAX_SIG_PUB_KEY_SIZE

```
#define TLS_MAX_SIG_PUB_KEY_SIZE 512
```

Max signature public key size in bytes 4096-bit RSA

6.1.2.47 TLS_MAX_SIG_SECRET_KEY_SIZE

```
#define TLS_MAX_SIG_SECRET_KEY_SIZE 1280
```

Max signature private key size in bytes 4096-bit RSA

6.1.2.48 TLS_MAX_SIGNATURE_SIZE

```
#define TLS_MAX_SIGNATURE_SIZE 512
```

Max digital signature size in bytes 4096-bit RSA

6.1.2.49 TLS_MAX_KEX_PUB_KEY_SIZE

```
#define TLS_MAX_KEX_PUB_KEY_SIZE 97
```

Max key exchange public key size in bytes 384-bit ECC

6.1.2.50 TLS_MAX_KEX_CIPHTEXT_SIZE

```
#define TLS_MAX_KEX_CIPHTEXT_SIZE 97
```

Max key exchange (KEM) ciphertext size 384-bit ECC

6.1.2.51 TLS_MAX_KEX_SECRET_KEY_SIZE

```
#define TLS_MAX_KEX_SECRET_KEY_SIZE 48
```

Max key exchange private key size in bytes 384-bit ECC

6.1.2.52 TLS_MAX_SERVER_CHAIN_LEN

```
#define TLS_MAX_SERVER_CHAIN_LEN 2
```

Maximum Server Certificate chain length - omitting root CA

6.1.2.53 TLS_MAX_SERVER_CHAIN_SIZE

```
#define TLS_MAX_SERVER_CHAIN_SIZE (TLS_MAX_SERVER_CHAIN_LEN*TLS_MAX_CERT_SIZE)
```

Maximum Server Certificate chain length in bytes

6.1.2.54 TLS_MAX_CLIENT_CHAIN_LEN

```
#define TLS_MAX_CLIENT_CHAIN_LEN 1
```

Maximum Client Certificate chain length - one self signed here

6.1.2.55 TLS_MAX_CLIENT_CHAIN_SIZE

```
#define TLS_MAX_CLIENT_CHAIN_SIZE (TLS_MAX_CLIENT_CHAIN_LEN*TLS_MAX_CERT_SIZE)
```

Maximum Client Certificate chain length in bytes

6.1.2.56 TLS_MAX_SHARED_SECRET_SIZE

```
#define TLS_MAX_SHARED_SECRET_SIZE 256
```

Max key exchange Shared secret size

6.1.2.57 TLS_MAX_TICKET_SIZE

```
#define TLS_MAX_TICKET_SIZE 4196
```

maximum resumption ticket size - beware some servers send much bigger tickets!

6.1.2.58 TLS_MAX_EXTENSIONS

```
#define TLS_MAX_EXTENSIONS 6144
```

Max extensions size

6.1.2.59 TLS_MAX_ECC_FIELD

```
#define TLS_MAX_ECC_FIELD 66
```

Max ECC field size in bytes

6.1.2.60 TLS_MAX_IV_SIZE

```
#define TLS_MAX_IV_SIZE 12
```

Max IV size in bytes

6.1.2.61 TLS_MAX_TAG_SIZE

```
#define TLS_MAX_TAG_SIZE 16
```

Max HMAC tag length in bytes

6.1.2.62 TLS_MAX_COOKIE

```
#define TLS_MAX_COOKIE 128
```

Max Cookie size

6.1.2.63 TLS_MAX_OUTPUT_RECORD_SIZE

```
#define TLS_MAX_OUTPUT_RECORD_SIZE 1024
```

Max output record size

6.1.2.64 TLS_MAX_OBUFF_SIZE

```
#define TLS_MAX_OBUFF_SIZE (TLS_MAX_OUTPUT_RECORD_SIZE+TLS_MAX_TAG_SIZE+6)
```

Max output buffer size

6.1.2.65 TLS_MAX_SERVER_NAME

```
#define TLS_MAX_SERVER_NAME 128
```

Max server name size in bytes

6.1.2.66 TLS_MAX_SUPPORTED_GROUPS

```
#define TLS_MAX_SUPPORTED_GROUPS 10
```

Max number of supported crypto groups

6.1.2.67 TLS_MAX_SUPPORTED_SIGS

```
#define TLS_MAX_SUPPORTED_SIGS 32
```

Max number of supported signature schemes

6.1.2.68 TLS_MAX_PSK_MODES

```
#define TLS_MAX_PSK_MODES 2
```

Max preshared key modes

6.1.2.69 TLS_MAX_CIPHER_SUITES

```
#define TLS_MAX_CIPHER_SUITES 5
```

Max number of supported cipher suites

6.1.2.70 TLS_AES_128_GCM_SHA256

```
#define TLS_AES_128_GCM_SHA256 0x1301
```

AES128/SHA256/GCM cipher suite - this is only one which MUST be implemented

6.1.2.71 TLS_AES_256_GCM_SHA384

```
#define TLS_AES_256_GCM_SHA384 0x1302
```

AES256/SHA384/GCM cipher suite

6.1.2.72 TLS_CHACHA20_POLY1305_SHA256

```
#define TLS_CHACHA20_POLY1305_SHA256 0x1303
```

CHACHA20/SHA256/POLY1305 cipher suite

6.1.2.73 TLS_AES_128_CCM_SHA256

```
#define TLS_AES_128_CCM_SHA256 0x1304
```

AES/SHA256/CCM cipher suite - optional

6.1.2.74 TLS_AES_128_CCM_8_SHA256

```
#define TLS_AES_128_CCM_8_SHA256 0x1305
```

AES/SHA256/CCM 8 cipher suite - optional

6.1.2.75 X25519

```
#define X25519 0x001d
```

X25519 elliptic curve key exchange

6.1.2.76 SECP256R1

```
#define SECP256R1 0x0017
```

NIST SECP256R1 elliptic curve key exchange

6.1.2.77 SECP384R1

```
#define SECP384R1 0x0018
```

NIST SECP384R1 elliptic curve key exchange

6.1.2.78 SECP521R1

```
#define SECP521R1 0x0019
```

NIST SECP521R1 elliptic curve key exchange

6.1.2.79 X448

```
#define X448 0x001e
```

X448 elliptic curve key exchange

6.1.2.80 MLKEM512

```
#define MLKEM512 0x0200
```

MLKEM PQ512 key exchange

6.1.2.81 MLKEM768

```
#define MLKEM768 0x0201
```

MLKEM PQ768 key exchange

6.1.2.82 MLKEM1024

```
#define MLKEM1024 0x0202
```

MLKEM1024 PQ key exchange

6.1.2.83 HYBRID_KX

```
#define HYBRID_KX 0x11ec
```

Hybrid key exchange, MLKEM768+X25519

6.1.2.84 ECDSA_SECP256R1_SHA256

```
#define ECDSA_SECP256R1_SHA256 0x0403
```

Supported ECDSA Signature algorithm

6.1.2.85 ECDSA_SECP256R1_SHA384

```
#define ECDSA_SECP256R1_SHA384 0x0413
```

Non-standard ECDSA Signature algorithm

6.1.2.86 ECDSA_SECP384R1_SHA384

```
#define ECDSA_SECP384R1_SHA384 0x0503
```

Supported ECDSA Signature algorithm

6.1.2.87 RSA_PSS_RSAE_SHA256

```
#define RSA_PSS_RSAE_SHA256 0x0804
```

Supported RSA Signature algorithm

6.1.2.88 RSA_PSS_RSAE_SHA384

```
#define RSA_PSS_RSAE_SHA384 0x0805
```

Supported RSA Signature algorithm

6.1.2.89 RSA_PSS_RSAE_SHA512

```
#define RSA_PSS_RSAE_SHA512 0x0806
```

Supported RSA Signature algorithm

6.1.2.90 RSA_PKCS1_SHA256

```
#define RSA_PKCS1_SHA256 0x0401
```

Supported RSA Signature algorithm

6.1.2.91 RSA_PKCS1_SHA384

```
#define RSA_PKCS1_SHA384 0x0501
```

Supported RSA Signature algorithm

6.1.2.92 RSA_PKCS1_SHA512

```
#define RSA_PKCS1_SHA512 0x0601
```

Supported RSA Signature algorithm

6.1.2.93 ED25519

```
#define ED25519 0x0807
```

Ed25519 EdDSA Signature algorithm

6.1.2.94 ED448

```
#define ED448 0x0808
```

Ed448 EdDSA Signature algorithm

6.1.2.95 MLDSA44

```
#define MLDSA44 0x0904
```

MLDSA44 Signature algorithm

6.1.2.96 MLDSA65

```
#define MLDSA65 0x0905
```

MLDSA65 Signature algorithm

6.1.2.97 MLDSA87

```
#define MLDSA87 0x0906
```

MLDSA87 Signature algorithm

6.1.2.98 PSKOK

```
#define PSKOK 0x00
```

Preshared Key only mode

6.1.2.99 PSKWECDHE

```
#define PSKWECDHE 0x01
```

Preshared Key with Diffie-Hellman key exchange mode

6.1.2.100 TLS_FULL_HANDSHAKE

```
#define TLS_FULL_HANDSHAKE 1
```

Came from Full Handshake

6.1.2.101 TLS_EXTERNAL_PSK

```
#define TLS_EXTERNAL_PSK 2
```

External Pre-Shared Key

6.1.2.102 TLS1_0

```
#define TLS1_0 0x0301
```

TLS 1.0 version

6.1.2.103 TLS1_2

```
#define TLS1_2 0x0303
```

TLS 1.2 version

6.1.2.104 TLS1_3

```
#define TLS1_3 0x0304
```

TLS 1.3 version

6.1.2.105 TLS13_UPDATE_NOT_REQUESTED

```
#define TLS13_UPDATE_NOT_REQUESTED 0
```

Updating my keys

6.1.2.106 TLS13_UPDATE_REQUESTED

```
#define TLS13_UPDATE_REQUESTED 1
```

Updating my keys and telling you to update yours

6.1.2.107 SERVER_NAME

```
#define SERVER_NAME 0x0000
```

Server Name extension

6.1.2.108 SUPPORTED_GROUPS

```
#define SUPPORTED_GROUPS 0x000a
```

Supported Group extension

6.1.2.109 SIG_ALGS

```
#define SIG_ALGS 0x000d
```

Signature algorithms extension

6.1.2.110 POST_HANDSHAKE_AUTH

```
#define POST_HANDSHAKE_AUTH 0x0031
```

Post Handshake Authentication

6.1.2.111 SIG_ALGS_CERT

```
#define SIG_ALGS_CERT 0x0032
```

Signature algorithms Certificate extension

6.1.2.112 KEY_SHARE

```
#define KEY_SHARE 0x0033
```

Key Share extension

6.1.2.113 PSK_MODE

```
#define PSK_MODE 0x002d
```

Preshared key mode extension

6.1.2.114 PRESHARED_KEY

```
#define PRESHARED_KEY 0x0029
```

Preshared key extension

6.1.2.115 TLS_VER

```
#define TLS_VER 0x002b
```

TLS version extension

6.1.2.116 COOKIE

```
#define COOKIE 0x002c
```

Cookie extension

6.1.2.117 EARLY_DATA

```
#define EARLY_DATA 0x002a
```

Early Data extension

6.1.2.118 MAX_FRAG_LENGTH

```
#define MAX_FRAG_LENGTH 0x0001
```

max fragmentation length extension

6.1.2.119 PADDING

```
#define PADDING 0x0015
```

Padding extension

6.1.2.120 APP_PROTOCOL

```
#define APP_PROTOCOL 0x0010
```

Application Layer Protocol Negotiation (ALPN)

6.1.2.121 RECORD_SIZE_LIMIT

```
#define RECORD_SIZE_LIMIT 0x001c
```

Record Size Limit

6.1.2.122 CLIENT_CERT_TYPE

```
#define CLIENT_CERT_TYPE 0x0013
```

Client Certificate type

6.1.2.123 SERVER_CERT_TYPE

```
#define SERVER_CERT_TYPE 0x0014
```

Server Certificate type

6.1.2.124 HEARTBEAT

```
#define HEARTBEAT 0x000f
```

Heartbeat

6.1.2.125 CERT_AUTHORITIES

```
#define CERT_AUTHORITIES 0x002f
```

Certificate Authorities

6.1.2.126 HSHAKE

```
#define HSHAKE 0x16
```

Handshake record

6.1.2.127 APPLICATION

```
#define APPLICATION 0x17
```

Application record

6.1.2.128 ALERT

```
#define ALERT 0x15
```

Alert record

6.1.2.129 CHANGE_CIPHER

```
#define CHANGE_CIPHER 0x14
```

Change Cipher record

6.1.2.130 HEART_BEAT

```
#define HEART_BEAT 0x18
```

Heart Beat Record

6.1.2.131 TIMED_OUT

```
#define TIMED_OUT 0x01
```

Time-out

6.1.2.132 CLIENT_HELLO

```
#define CLIENT_HELLO 0x01
```

Client Hello message

6.1.2.133 SERVER_HELLO

```
#define SERVER_HELLO 0x02
```

Server Hello message

6.1.2.134 CERTIFICATE

```
#define CERTIFICATE 0x0b
```

Certificate message

6.1.2.135 CERT_REQUEST

```
#define CERT_REQUEST 0x0d
```

Certificate Request

6.1.2.136 CERT_VERIFY

```
#define CERT_VERIFY 0x0f
```

Certificate Verify message

6.1.2.137 FINISHED

```
#define FINISHED 0x14
```

Handshake Finished message

6.1.2.138 ENCRYPTED_EXTENSIONS

```
#define ENCRYPTED_EXTENSIONS 0x08
```

Encrypted Extensions message

6.1.2.139 TICKET

```
#define TICKET 0x04
```

Ticket message

6.1.2.140 KEY_UPDATE

```
#define KEY_UPDATE 0x18
```

Key Update message

6.1.2.141 MESSAGE_HASH

```
#define MESSAGE_HASH 0xFE
```

Special synthetic message hash message

6.1.2.142 END_OF_EARLY_DATA

```
#define END_OF_EARLY_DATA 0x05
```

End of Early Data message

6.1.2.143 HANDSHAKE_RETRY

```
#define HANDSHAKE_RETRY 0x102
```

Handshake retry

6.1.2.144 NOT_TLS1_3

```
#define NOT_TLS1_3 -2
```

Wrong version error, not TLS1.3

6.1.2.145 BAD_CERT_CHAIN

```
#define BAD_CERT_CHAIN -3
```

Bad Certificate Chain error

6.1.2.146 ID_MISMATCH

```
#define ID_MISMATCH -4
```

Session ID mismatch error

6.1.2.147 UNRECOGNIZED_EXT

```
#define UNRECOGNIZED_EXT -5
```

Unrecognised extension error

6.1.2.148 BAD_HELLO

```
#define BAD_HELLO -6
```

badly formed Hello message error

6.1.2.149 WRONG_MESSAGE

```
#define WRONG_MESSAGE -7
```

Message out-of-order error

6.1.2.150 MISSING_REQUEST_CONTEXT

```
#define MISSING_REQUEST_CONTEXT -8
```

Request context missing error

6.1.2.151 AUTHENTICATION_FAILURE

```
#define AUTHENTICATION_FAILURE -9
```

Authentication error - AEAD Tag incorrect

6.1.2.152 BAD_RECORD

```
#define BAD_RECORD -10
```

Badly formed Record received

6.1.2.153 BAD_TICKET

```
#define BAD_TICKET -11
```

Badly formed Ticket received

6.1.2.154 NOT_EXPECTED

```
#define NOT_EXPECTED -12
```

Received ack for something not requested

6.1.2.155 CA_NOT_FOUND

```
#define CA_NOT_FOUND -13
```

Certificate Authority not found

6.1.2.156 CERT_OUTOFDATE

```
#define CERT_OUTOFDATE -14
```

Certificate Expired

6.1.2.157 MEM_OVERFLOW

```
#define MEM_OVERFLOW -15
```

Memory Overflow

6.1.2.158 FORBIDDEN_EXTENSION

```
#define FORBIDDEN_EXTENSION -16
```

Forbidden Encrypted Extension

6.1.2.159 MAX_EXCEEDED

```
#define MAX_EXCEEDED -17
```

Maximum record size exceeded

6.1.2.160 EMPTY_CERT_CHAIN

```
#define EMPTY_CERT_CHAIN -18
```

Empty Certificate Message

6.1.2.161 SELF_SIGNED_CERT

```
#define SELF_SIGNED_CERT -20
```

Self signed certificate

6.1.2.162 TIME_OUT

```
#define TIME_OUT -21
```

Time out

6.1.2.163 ERROR_ALERT_RECEIVED

```
#define ERROR_ALERT_RECEIVED -22
```

Alert has been received

6.1.2.164 BAD_MESSAGE

```
#define BAD_MESSAGE -23
```

Badly formed message

6.1.2.165 CERT_VERIFY_FAIL

```
#define CERT_VERIFY_FAIL -24
```

Certificate Verification failure

6.1.2.166 BAD_HANDSHAKE

```
#define BAD_HANDSHAKE -26
```

Could not agree

6.1.2.167 BAD_REQUEST_UPDATE

```
#define BAD_REQUEST_UPDATE -27
```

Bad Request Update value

6.1.2.168 CLOSURE_ALERT_RECEIVED

```
#define CLOSURE_ALERT_RECEIVED -28
```

Alert has been received

6.1.2.169 MISSING_EXTENSIONS

```
#define MISSING_EXTENSIONS -30
```

Some mandatory extensions are missing

6.1.2.170 ILLEGAL_PARAMETER

```
#define ILLEGAL_PARAMETER 0x2F
```

Illegal parameter alert

6.1.2.171 UNEXPECTED_MESSAGE

```
#define UNEXPECTED_MESSAGE 0x0A
```

Unexpected message alert

6.1.2.172 DECRYPT_ERROR

```
#define DECRYPT_ERROR 0x33
```

Decryption error alert

6.1.2.173 BAD_CERTIFICATE

```
#define BAD_CERTIFICATE 0x2A
```

Bad certificate alert

6.1.2.174 UNSUPPORTED_EXTENSION

```
#define UNSUPPORTED_EXTENSION 0x6E
```

Unsupported extension alert

6.1.2.175 UNKNOWN_CA

```
#define UNKNOWN_CA 0x30
```

Unrecognised Certificate Authority

6.1.2.176 CERTIFICATE_EXPIRED

```
#define CERTIFICATE_EXPIRED 0x2D
```

Certificate Expired

6.1.2.177 PROTOCOL_VERSION

```
#define PROTOCOL_VERSION 0x46
```

Wrong TLS version

6.1.2.178 DECODE_ERROR

```
#define DECODE_ERROR 0x32
```

Decode error alert

6.1.2.179 RECORD_OVERFLOW

```
#define RECORD_OVERFLOW 0x16
```

Record Overflow

6.1.2.180 BAD_RECORD_MAC

```
#define BAD_RECORD_MAC 0x14
```

Bad Record Mac

6.1.2.181 HANDSHAKE_FAILURE

```
#define HANDSHAKE_FAILURE 0x28
```

Could not agree

6.1.2.182 CLOSE_NOTIFY

```
#define CLOSE_NOTIFY 0x00
```

Orderly shut down of connection

6.1.2.183 MISSING_EXTENSION

```
#define MISSING_EXTENSION 0x6D;
```

Missing extension

6.1.2.184 LOG_OUTPUT_TRUNCATION

```
#define LOG_OUTPUT_TRUNCATION 256
```

Output Hex digits before truncation

6.1.2.185 TLS13_DISCONNECTED

```
#define TLS13_DISCONNECTED 0
```

TLS1.3 Connection is broken

6.1.2.186 TLS13_CONNECTED

```
#define TLS13_CONNECTED 1
```

TLS1.3 Connection is made

6.1.2.187 TLS13_HANDSHAKING

```
#define TLS13_HANDSHAKING 2
```

TLS1.3 is handshaking

6.1.2.188 TLS_FAILURE

```
#define TLS_FAILURE 0
```

Failed to cmake TLS1.3 connection

6.1.2.189 TLS_SUCCESS

```
#define TLS_SUCCESS 1
```

Succeeded in making TLS1.3 connection

6.1.2.190 TLS_RESUMPTION_REQUIRED

```
#define TLS_RESUMPTION_REQUIRED 2
```

Connection succeeded, but handshake retry was needed

6.1.2.191 TLS_EARLY_DATA_ACCEPTED

```
#define TLS_EARLY_DATA_ACCEPTED 3
```

Connection succeeded, and early data was accepted

6.1.2.192 PSK_NOT

```
#define PSK_NOT 0
```

No PSK

6.1.2.193 PSK_KEY

```
#define PSK_KEY 1
```

Using PSK from database

6.1.2.194 PSK_IBE

```
#define PSK_IBE 2
```

Using IBE based PSK

6.1.2.195 X509_CERT

```
#define X509_CERT 0
```

X509 Certificate-based authentication

6.1.2.196 RAW_PUBLIC_KEY

```
#define RAW_PUBLIC_KEY 2
```

Raw Public Key based authentication

6.1.3 Typedef Documentation

6.1.3.1 byte

```
typedef uint8_t byte
```

8-bit unsigned integer

6.1.3.2 sign8

```
typedef int8_t sign8
```

8-bit signed integer

6.1.3.3 sign16

```
typedef int16_t sign16
```

16-bit signed integer

6.1.3.4 sign32

```
typedef int32_t sign32
```

32-bit signed integer

6.1.3.5 sign64

```
typedef int64_t sign64
```

64-bit signed integer

6.1.3.6 unsign16

```
typedef uint16_t unsign16
```

16-bit unsigned integer

6.1.3.7 unsign32

```
typedef uint32_t unsign32
```

32-bit unsigned integer

6.1.3.8 unsign64

```
typedef uint64_t unsign64
```

64-bit unsigned integer

6.2 tls1_3.h

[Go to the documentation of this file.](#)

```
00001
00008 #ifndef TLS1_3_H
00009 #define TLS1_3_H
00010
00011 #include <stdint.h>
00012 #include "tls_octads.h"
00013 #include "tls_sockets.h"
00014
00015 typedef uint8_t byte;
00016 typedef int8_t sign8 ;
00017 typedef int16_t sign16;
00018 typedef int32_t sign32;
00019 typedef int64_t sign64;
00020 typedef uint16_t unsign16;
00021 typedef uint32_t unsign32;
00022 typedef uint64_t unsign64;
00024 // Terminal Output
00025 #define IO_NONE 0
00026 #define IO_APPLICATION 1
00027 #define IO_PROTOCOL 2
00028 #define IO_DEBUG 3
00029 #define IO_WIRE 4
00031 // // Supported CRYPTO_SETTINGS
00032 #define TINY_ECC 0
00033 #define TYPICAL 1
00034 #define POST_QUANTUM 3
00035 #define HYBRID 4
00036 #define EDDSA 2
00038 // Client Certificate Chain + Key
00039 #define NO_CERT 1
00040 #define FROM_FILE 2
00041 #define FROM_ROM 3
00043 // Client certificate kinds
00044 #define RSA_SS 1
00045 #define ECC_SS 2
00046 #define DLT_SS 3
00047 #define HYB_SS 6
00048 #define HW_1 4
00049 #define HW_2 5
00050 #define EDD_SS 7
00052 // THESE ARE IMPORTANT USER DEFINED SETTINGS *****
00053
00054 // Note that favourite group (as used in client hello) is determined by the SAL ordering - see
    tls_sal.cpp
00055 // If server does not support it, an expensive Handshake Retry will be required
00056 // So best to place a popular group (such as X25519) at top of list in SAL
00057
00058 #define CRYPTO_SETTING TYPICAL
00060 #define VERBOSITY IO_PROTOCOL
00061 // #define THIS_YEAR 2025          /**< Set to this year - was crudely used to deprecate old
    certificates - no longer used */
00062
00063 // Client side authentication
00064 #define CLIENT_CERT FROM_ROM
00065 #define CLIENT_CERT_KIND ECC_SS
00066 #define POST_HS_AUTH
00068 // Supported protocols
00069 #define TLS_APPLICATION_PROTOCOL (char *)("http/1.1")
00071 // self-signed certificate not be allowed in real deployments
00072 #define ALLOW_SELF_SIGNED
00073 // #define NO_CERT_CHECKS          /**< Don't do any checks on server certs - useful for Anvil testing
    */
00074 #define TRY_EARLY_DATA
00076 // #define ENABLE_HEARTBEATS          /**< Enable heartbeats */
00077 // #define PEER_CAN_HEARTBEAT        /**< Allow peer to heartbeat */
00078
00079 // Note that BUFF, Certificates and crypto keys can be quite large, and therefore maybe better taken
    from the heap
00080 // on systems with a shallow stack. Define this to use the heap.
```



```

00081
00082 // #define SHALLOW_STACK          /**< Get large arrays from heap, else stack */
00083
00084 // comment out if no max record size. In practise TLS1.3 doesn't seem to support this
00085 // #define MAX_RECORD 1024          /**< Maximum record size client is willing to receive - should be
00086 // Note that if this is not used, max_fragment_size extension is tried instead, see TLS_MAX_FRAG below
00087
00088 // define this so that all encrypted records are padded with 0s to full length
00089 // #define PAD_SHORT_RECORDS        /**< Pad short output records */
00090
00091 // #define PREFER_RAW_SERVER_PUBLIC_KEY /**< Would be happy with raw public key from server */
00092 // #define PREFER_RAW_CLIENT_PUBLIC_KEY /**< Would prefer server to accept raw public key from client
00093 */
00094 // may need to undefine this for fuzzing
00095 #define MERGE_MESSAGES
00097 // may need to comment out for testing
00098 #define CHECK_NAME_IN_CERT
00101 // *****
00102
00103
00104 // Standard Hash Types
00105
00106 #define TLS_SHA256_T 1
00107 #define TLS_SHA384_T 2
00108 #define TLS_SHA512_T 3
00110 // Some maximum sizes for stack allocated memory. Handshake will fail if these sizes are exceeded!
00111
00112 #define TLS_MAX_HASH_STATE 768
00113 #define TLS_MAX_HASH 64
00114 #define TLS_MAX_KEY 32
00115 #define TLS_X509_MAX_FIELD 256
00116 #define TLS_MAX_EXT_LABEL 256
00118 // Max Frag length must be less than TLS_MAX_IBUFF_SIZE
00119 #define TLS_MAX_FRAG 2
00121 #if CRYPTO_SETTING==TINY_ECC
00122 #define TLS_MAX_IBUFF_SIZE (4096+256)
00123 #define TLS_MAX_PLAIN_FRAG 4096
00124 #define TLS_MAX_CIPHER_FRAG (4096+256)
00126 #define TLS_MAX_CERT_SIZE 2048
00127 #define TLS_MAX_CERT_B64 2800
00128 #define TLS_MAX_HELLO 1024
00130 #define TLS_MAX_SIG_PUB_KEY_SIZE 133
00131 #define TLS_MAX_SIG_SECRET_KEY_SIZE 66
00132 #define TLS_MAX_SIGNATURE_SIZE 132
00133 #define TLS_MAX_KEX_PUB_KEY_SIZE 97
00134 #define TLS_MAX_KEX_CIPHERTEXT_SIZE 97
00135 #define TLS_MAX_KEX_SECRET_KEY_SIZE 48
00136 #endif
00137
00138 #if CRYPTO_SETTING==TYPICAL || CRYPTO_SETTING==EDDSA
00139 #define TLS_MAX_IBUFF_SIZE (16384+256)
00140 #define TLS_MAX_PLAIN_FRAG 16384
00141 #define TLS_MAX_CIPHER_FRAG (16384+256)
00143 #define TLS_MAX_CERT_SIZE 2048
00144 #define TLS_MAX_CERT_B64 2800
00145 #define TLS_MAX_HELLO 1024
00147 #define TLS_MAX_SIG_PUB_KEY_SIZE 512
00148 #define TLS_MAX_SIG_SECRET_KEY_SIZE 1280
00149 #define TLS_MAX_SIGNATURE_SIZE 512
00150 #define TLS_MAX_KEX_PUB_KEY_SIZE 97
00151 #define TLS_MAX_KEX_CIPHERTEXT_SIZE 97
00152 #define TLS_MAX_KEX_SECRET_KEY_SIZE 48
00153 #endif
00154
00155 #if CRYPTO_SETTING == POST_QUANTUM
00156
00157 #define TLS_MAX_IBUFF_SIZE (16384+256)
00158 #define TLS_MAX_PLAIN_FRAG 16384
00159 #define TLS_MAX_CIPHER_FRAG (16384+256)
00161 #define TLS_MAX_CERT_SIZE 6144
00162 #define TLS_MAX_CERT_B64 8192
00163 #define TLS_MAX_HELLO 2048
00165 // These all blow up post quantum
00166 #define TLS_MAX_SIG_PUB_KEY_SIZE 1952
00167 #define TLS_MAX_SIG_SECRET_KEY_SIZE 4000
00168 #define TLS_MAX_SIGNATURE_SIZE 3309
00169 #define TLS_MAX_KEX_PUB_KEY_SIZE 1184
00170 #define TLS_MAX_KEX_CIPHERTEXT_SIZE 1088
00171 #define TLS_MAX_KEX_SECRET_KEY_SIZE 2400
00172 #endif
00173
00174 #if CRYPTO_SETTING == HYBRID
00175
00176 #define TLS_MAX_IBUFF_SIZE (16384+256)

```

```

00177 #define TLS_MAX_PLAIN_FRAG 16384
00178 #define TLS_MAX_CIPHER_FRAG (16384+256)
00180 #define TLS_MAX_CERT_SIZE 6144
00181 #define TLS_MAX_CERT_B64 8192
00182 #define TLS_MAX_HELLO 2048
00184 // These all blow up post quantum
00185 #define TLS_MAX_SIG_PUB_KEY_SIZE 1312+65
00186 #define TLS_MAX_SIG_SECRET_KEY_SIZE 2528+200
00187 #define TLS_MAX_SIGNATURE_SIZE 2420+100
00188 #define TLS_MAX_KEX_PUB_KEY_SIZE 1184+32
00189 #define TLS_MAX_KEX_CIPHERTEXT_SIZE 1088+32
00190 #define TLS_MAX_KEX_SECRET_KEY_SIZE 2400+32
00191 #endif
00192
00193
00194 // Certificate size limits
00195 #define TLS_MAX_SERVER_CHAIN_LEN 2
00196 #define TLS_MAX_SERVER_CHAIN_SIZE (TLS_MAX_SERVER_CHAIN_LEN*TLS_MAX_CERT_SIZE)
00197 #define TLS_MAX_CLIENT_CHAIN_LEN 1
00198 #define TLS_MAX_CLIENT_CHAIN_SIZE (TLS_MAX_CLIENT_CHAIN_LEN*TLS_MAX_CERT_SIZE)
00200 #define TLS_MAX_SHARED_SECRET_SIZE 256
00202 // Both of these are bumped up by PQ IBE and Hybrid
00203 #define TLS_MAX_TICKET_SIZE 4196
00204 #define TLS_MAX_EXTENSIONS 6144
00206 #define TLS_MAX_ECC_FIELD 66
00207 #define TLS_MAX_IV_SIZE 12
00208 #define TLS_MAX_TAG_SIZE 16
00209 #define TLS_MAX_COOKIE 128
00211 #define TLS_MAX_OUTPUT_RECORD_SIZE 1024
00212 #define TLS_MAX_OBUFF_SIZE (TLS_MAX_OUTPUT_RECORD_SIZE+TLS_MAX_TAG_SIZE+6)
00214 #define TLS_MAX_SERVER_NAME 128
00215 #define TLS_MAX_SUPPORTED_GROUPS 10
00216 #define TLS_MAX_SUPPORTED_SIGS 32
00217 #define TLS_MAX_PSK_MODES 2
00218 #define TLS_MAX_CIPHER_SUITES 5
00220 // Cipher Suites
00221 #define TLS_AES_128_GCM_SHA256 0x1301
00222 #define TLS_AES_256_GCM_SHA384 0x1302
00223 #define TLS_CHACHA20_POLY1305_SHA256 0x1303
00224 #define TLS_AES_128_CCM_SHA256 0x1304
00225 #define TLS_AES_128_CCM_8_SHA256 0x1305
00227 // Key exchange groups
00228 #define X25519 0x001d
00229 #define SECP256R1 0x0017
00230 #define SECP384R1 0x0018
00231 #define SECP521R1 0x0019
00232 #define X448 0x001e
00233 #define MLKEM512 0x0200
00234 #define MLKEM768 0x0201
00235 #define MLKEM1024 0x0202
00236 #define HYBRID_KX 0x11ec
00238 // Signature algorithms for TLS1.3 and Certs that we can handle
00239 #define ECDSA_SECP256R1_SHA256 0x0403
00240 #define ECDSA_SECP256R1_SHA384 0x0413
00241 #define ECDSA_SECP384R1_SHA384 0x0503
00242 #define RSA_PSS_RSAE_SHA256 0x0804
00243 #define RSA_PSS_RSAE_SHA384 0x0805
00244 #define RSA_PSS_RSAE_SHA512 0x0806
00245 #define RSA_PKCS1_SHA256 0x0401
00246 #define RSA_PKCS1_SHA384 0x0501
00247 #define RSA_PKCS1_SHA512 0x0601
00248 #define ED25519 0x0807
00249 #define ED448 0x0808
00250 #define MLDSA44 0x0904
00251 #define MLDSA65 0x0905
00252 #define MLDSA87 0x0906
00253 #define MLDSA44_P256 0xFF06 //0x0907
00254
00255 // pre-shared Key (PSK) modes
00256 #define PSKOK 0x00
00257 #define PSKWECDHE 0x01
00259 // ticket origin
00260 #define TLS_FULL_HANDSHAKE 1
00261 #define TLS_EXTERNAL_PSK 2
00263 // TLS versions
00264 #define TLS1_0 0x0301
00265 #define TLS1_2 0x0303
00266 #define TLS1_3 0x0304
00268 #define TLS13_UPDATE_NOT_REQUESTED 0
00269 #define TLS13_UPDATE_REQUESTED 1
00271 // Extensions
00272 #define SERVER_NAME 0x0000
00273 #define SUPPORTED_GROUPS 0x000a
00274 #define SIG_ALGS 0x000d
00275 #define POST_HANDSHAKE_AUTH 0x0031
00276 #define SIG_ALGS_CERT 0x0032
00277 #define KEY_SHARE 0x0033

```

```
00278 #define PSK_MODE 0x002d
00279 #define PRESHARED_KEY 0x0029
00280 #define TLS_VER 0x002b
00281 #define COOKIE 0x002c
00282 #define EARLY_DATA 0x002a
00283 #define MAX_FRAG_LENGTH 0x0001
00284 #define PADDING 0x0015
00285 #define APP_PROTOCOL 0x0010
00286 #define RECORD_SIZE_LIMIT 0x001c
00287 #define CLIENT_CERT_TYPE 0x0013
00288 #define SERVER_CERT_TYPE 0x0014
00289 #define HEARTBEAT 0x000f
00290 #define CERT_AUTHORITIES 0x002f
00292 // record types
00293 #define HSHAKE 0x16
00294 #define APPLICATION 0x17
00295 #define ALERT 0x15
00296 #define CHANGE_CIPHER 0x14
00297 #define HEART_BEAT 0x18
00298 // pseudo record types
00299 #define TIMED_OUT 0x01
00301 // message types
00302 #define CLIENT_HELLO 0x01
00303 #define SERVER_HELLO 0x02
00304 #define CERTIFICATE 0x0b
00305 #define CERT_REQUEST 0x0d
00306 #define CERT_VERIFY 0x0f
00307 #define FINISHED 0x14
00308 #define ENCRYPTED_EXTENSIONS 0x08
00309 #define TICKET 0x04
00310 #define KEY_UPDATE 0x18
00311 #define MESSAGE_HASH 0xfe
00312 #define END_OF_EARLY_DATA 0x05
00313 // pseudo message types
00314 #define HANDSHAKE_RETRY 0x102
00316 // Causes of server error - which should generate a client alert
00317 #define NOT_TLS1_3 -2
00318 #define BAD_CERT_CHAIN -3
00319 #define ID_MISMATCH -4
00320 #define UNRECOGNIZED_EXT -5
00321 #define BAD_HELLO -6
00322 #define WRONG_MESSAGE -7
00323 #define MISSING_REQUEST_CONTEXT -8
00324 #define AUTHENTICATION_FAILURE -9
00325 #define BAD_RECORD -10
00326 #define BAD_TICKET -11
00327 #define NOT_EXPECTED -12
00328 #define CA_NOT_FOUND -13
00329 #define CERT_OUTOFDATE -14
00330 #define MEM_OVERFLOW -15
00331 #define FORBIDDEN_EXTENSION -16
00332 #define MAX_EXCEEDED -17
00333 #define EMPTY_CERT_CHAIN -18
00334 #define SELF_SIGNED_CERT -20
00335 #define TIME_OUT -21
00336 #define ERROR_ALERT_RECEIVED -22
00337 #define BAD_MESSAGE -23
00338 #define CERT_VERIFY_FAIL -24
00339 #define BAD_HANDSHAKE -26
00340 #define BAD_REQUEST_UPDATE -27
00341 #define CLOSURE_ALERT_RECEIVED -28
00342 #define MISSING_EXTENSIONS -30
00343 // client alerts
00344 #define ILLEGAL_PARAMETER 0x2f
00345 #define UNEXPECTED_MESSAGE 0x0a
00346 #define DECRYPT_ERROR 0x33
00347 #define BAD_CERTIFICATE 0x2a
00348 #define UNSUPPORTED_EXTENSION 0x6e
00349 #define UNKNOWN_CA 0x30
00350 #define CERTIFICATE_EXPIRED 0x2d
00351 #define PROTOCOL_VERSION 0x46
00352 #define DECODE_ERROR 0x32
00353 #define RECORD_OVERFLOW 0x16
00354 #define BAD_RECORD_MAC 0x14
00355 #define HANDSHAKE_FAILURE 0x28
00356 #define CLOSE_NOTIFY 0x00
00357 #define MISSING_EXTENSION 0x6d;
00359 #define LOG_OUTPUT_TRUNCATION 256
00361 #define TLS13_DISCONNECTED 0
00362 #define TLS13_CONNECTED 1
00363 #define TLS13_HANDSHAKING 2
00365 // protocol returns..
00366 #define TLS_FAILURE 0
00367 #define TLS_SUCCESS 1
00368 #define TLS_RESUMPTION_REQUIRED 2
00369 #define TLS_EARLY_DATA_ACCEPTED 3
00371 // PSK modes
```

```

00372 #define PSK_NOT 0
00373 #define PSK_KEY 1
00374 #define PSK_IKE 2
00376 // Certificate types
00377 #define X509_CERT 0
00378 #define RAW_PUBLIC_KEY 2
00382 typedef struct
00383 {
00384     unsigned val;
00385     int err;
00386 } ret;
00387
00390 typedef struct
00391 {
00392     bool early_data;
00393     bool alpn;
00394     bool server_name;
00395     bool max_frag_length;
00396 } ee_status;
00397
00400 typedef struct
00401 {
00402     bool active;
00403     char k[TLS_MAX_KEY];
00404     char iv[12];
00405     octad K;
00406     octad IV;
00407     unsigned record;
00408     int suite;
00409     int taglen;
00410 } crypto;
00411
00414 typedef struct
00415 {
00416     char certchain[TLS_MAX_CLIENT_CHAIN_SIZE];
00417     char publickey[TLS_MAX_SIG_PUB_KEY_SIZE];
00418     char secretkey[TLS_MAX_SIG_SECRET_KEY_SIZE];
00419     octad CERTCHAIN;
00420     octad PUBLICKEY;
00421     octad SECRETKEY;
00422     unsigned requirements[16];
00423     int nregs;
00424     int nregsraw;
00425     int sigalg;
00426 } credential;
00427
00430 typedef struct
00431 {
00432     bool valid;
00433     char tick[TLS_MAX_TICKET_SIZE];
00434     char nonce[256];
00435     char psk[TLS_MAX_HASH];
00436     octad TICK;
00437     octad NONCE;
00438     octad PSK;
00439     unsigned age_obfuscator;
00440     unsigned max_early_data;
00441     unsigned birth;
00442     int lifetime;
00443     int cipher_suite;
00444     int favourite_group;
00445     int origin;
00446 } ticket;
00447
00450 typedef struct
00451 {
00452     char state[TLS_MAX_HASH_STATE];
00453     int htype;
00454 } unihash;
00455
00458 typedef struct
00459 {
00460     int status;
00461     int max_record;
00462     Socket *sockptr;
00463     char id[32];
00464     char hostname[TLS_MAX_SERVER_NAME];
00465     int cipher_suite;
00466     int favourite_group;
00467     int server_cert_type;
00468     int client_cert_type;
00469     crypto K_send;
00470     crypto K_rcv;
00471     octad HS;
00472     char hs[TLS_MAX_HASH];
00473     octad RMS;
00474     char rms[TLS_MAX_HASH];

```

```
00475     octad STS;
00476     char sts[TLS_MAX_HASH];
00477     octad CTS;
00478     char cts[TLS_MAX_HASH];
00479     octad CTX;
00480     char ctx[TLS_MAX_HASH];
00481     octad IBUFF;
00482     octad OBUFF;
00483 #ifndef SHALLOW_STACK
00484     char ibuff[TLS_MAX_IBUFF_SIZE];
00485     char obuff[TLS_MAX_OBUFF_SIZE];
00486 #endif
00487     int ptr;
00488     unihash tlshash;
00489     ticket T;
00490     bool expect_heartbeats;
00491     bool allowed_to_heartbeat;
00492     bool heartbeat_req_in_flight;
00493 } TLS_session;
00494
00495 // IBUFF buffer
00496 // xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
00497 // -----ptr----->-----IBUFF.len----->
00498 //
00499 // when ptr becomes equal to IBUFF.len, pull in another record (and maybe decrypt it)
00500
00501 #endif
```

6.3 tls_bfibe.h File Reference

Boneh and Franklin IBE.

```
#include "core.h"
```

Functions

- bool **BFIBE_CCA_ENCRYPT** (char *ID, octet *R32, octet *SSK, octet *CT)
Create key SSK encapsulated in ciphertext CT to be sent to ID.
- bool **BFIBE_CCA_DECRYPT** (octet *SK, octet *CT, octet *SSK)
Create key SSK encapsulated in ciphertext CT to be sent to ID.

6.3.1 Detailed Description

Boneh and Franklin IBE.

Author

Mike Scott

6.3.2 Function Documentation

6.3.2.1 BFIBE_CCA_ENCRYPT()

```
bool BFIBE_CCA_ENCRYPT (
    char * ID,
    octet * R32,
    octet * SSK,
    octet * CT )
```

Create key SSK encapsulated in ciphertext CT to be sent to ID.

Parameters

<i>ID</i>	the entity to receive encapsulated key
<i>R32</i>	32 random bytes
<i>SSK</i>	is the encapsulated key
<i>CT</i>	is the ciphertext

Returns

true if OK

6.3.2.2 BFIBE_CCA_DECRYPT()

```
bool BFIBE_CCA_DECRYPT (
    octet * SK,
    octet * CT,
    octet * SSK )
```

Create key SSK encapsulated in ciphertext CT to be sent to ID.

Parameters

<i>SK</i>	the secret key of ID
<i>CT</i>	is the ciphertext
<i>SSK</i>	is the decapsulated key

Returns

true if OK

6.4 tls_bfibe.h

[Go to the documentation of this file.](#)

```
00001
00007 //
00008 // Pairing-based B&F IBE 128-bit API Functions
00009 // Uses MIRACL
00010 //
00011 #ifndef BFIBE_BLS381_H
00012 #define BFIBE_BLS381_H
00013
00014 #include "core.h"
00015 using namespace core;
00016
00017 /* IBE primitives */
00018
00027 bool BFIBE_CCA_ENCRYPT(char *ID, octet *R32, octet *SSK, octet *CT);
00028
00036 bool BFIBE_CCA_DECRYPT(octet *SK, octet *CT, octet *SSK);
00037
00038 #endif
00039
```

6.5 `tls_cert_chain.h` File Reference

Process Certificate Chain.

```
#include "tls1_3.h"
#include "tls_x509.h"
#include "tls_sal.h"
#include "tls_client_recv.h"
#include "tls_logger.h"
#include "tls_certs.h"
```

Functions

- int `checkServerCertChain` (`octad` *CERTCHAIN, char *hostname, int cert_type, `octad` *PUBKEY, `octad` *SIG)
Check Certificate Chain for hostname, and extract public key.

6.5.1 Detailed Description

Process Certificate Chain.

Author

Mike Scott

6.5.2 Function Documentation

6.5.2.1 `checkServerCertChain()`

```
int checkServerCertChain (
    octad * CERTCHAIN,
    char * hostname,
    int cert_type,
    octad * PUBKEY,
    octad * SIG ) [extern]
```

Check Certificate Chain for hostname, and extract public key.

Parameters

<i>CERTCHAIN</i>	the input certificate chain
<i>hostname</i>	the input Server name associated with the Certificate chain
<i>cert_type</i>	the certificate type (a certificate or a raw key)
<i>PUBKEY</i>	the Server's public key extracted from the Certificate chain
<i>SIG</i>	signature (supplied as workspace)

Returns

0 if certificate chain is OK, else returns negative failure reason

6.6 `tls_cert_chain.h`

[Go to the documentation of this file.](#)

```

00001
00008 // TLS1.3 Server Certificate Chain Code
00009
00010 #ifndef TLS_CERT_CHAIN_H
00011 #define TLS_CERT_CHAIN_H
00012 #include "tls1_3.h"
00013 #include "tls_x509.h"
00014 #include "tls_sal.h"
00015 #include "tls_client_recv.h"
00016 #include "tls_logger.h"
00017 #include "tls_certs.h"
00018
00019 using namespace std;
00020
00030 extern int checkServerCertChain(octad *CERTCHAIN, char *hostname, int cert_type, octad *PUBKEY, octad
    *SIG);
00031
00032 #endif

```

6.7 `tls_certs.h` File Reference

Certificate Authority root certificate store.

```
#include "tls1_3.h"
```

Functions

- bool [setCredential](#) ([credential](#) *Credential)
Extract certificate chain and secret key from client credentials (either stored or from file).

Variables

- const char * [myprivate](#)
- const char * [mycert](#)
- const char * [cacerts](#)

6.7.1 Detailed Description

Certificate Authority root certificate store.

Author

Mike Scott

6.7.2 Function Documentation

6.7.2.1 `setCredential()`

```
bool setCredential (
    credential * Credential ) [extern]
```

Extract certificate chain and secret key from client credentials (either stored or from file).

Parameters

<i>Credential</i>	the client credential structure to be filled
-------------------	--

Returns

false if client not equipped (via SAL) to implement signature

6.7.3 Variable Documentation

6.7.3.1 myprivate

```
const char* myprivate [extern]
```

Client private key

6.7.3.2 mycert

```
const char* mycert [extern]
```

Client certificate

6.7.3.3 cacerts

```
const char* cacerts [extern]
```

The Root Certificate store

6.8 tls_certs.h

[Go to the documentation of this file.](#)

```
00001
00008 #ifndef TLS_CA_CERTS_H
00009 #define TLS_CA_CERTS_H
00010
00011 #include "tls1_3.h"
00012
00013 //extern const char *mysupportedca; /**< Supported root CA */
00014 extern const char *myprivate;
00015 extern const char *mycert;
00016 extern const char *cacerts;
00018 using namespace std;
00019
00025 extern bool setCredential(Credential *Credential);
00026
00027 #endif
```

6.9 tls_client_recv.h File Reference

Process Input received from the Server.

```
#include "tls_sal.h"
#include "tls1_3.h"
#include "tls_sockets.h"
#include "tls_keys_calc.h"
#include "tls_client_send.h"
```

Functions

- **ret parseoctad** (octad *E, int len, octad *M, int &ptr)
Parse out an octad from a pointer into an octad.
- **ret parsebytes** (char *e, int len, octad *M, int &ptr)
Parse out byte array from a pointer into an octad.
- **ret parseInt** (octad *M, int len, int &ptr)
Parse out an unsigned integer from a pointer into an octad.
- **ret parseoctadptr** (octad *E, int len, octad *M, int &ptr)
Return a pointer to an octad from a pointer into an octad.
- **int getServerRecord** (TLS_session *session)
Read a record from the Server, a fragment of a full protocol message.
- **ret parseIntorPull** (TLS_session *session, int len)
Parse out an unsigned integer from a pointer into an octad, if necessary pulling in a new fragment.
- **ret parseoctadorPull** (TLS_session *session, octad *O, int len)
Parse out an octad from a pointer into an octad, if necessary pulling in a new fragment.
- **ret parsebytesorPull** (TLS_session *session, char *o, int len)
Parse out a byte array from a pointer into an octad, if necessary pulling in a new fragment.
- **ret parseoctadorPullptrX** (TLS_session *session, octad *O, int len)
Return a pointer to an octad from a pointer into an octad, if necessary pulling in a new fragment.
- **bool badResponse** (TLS_session *session, ret r)
Process response from server input.
- **ret seeWhatsNext** (TLS_session *session)
Identify type of incoming message.
- **ret getServerEncryptedExtensions** (TLS_session *session, ee_status *enc_ext_expt, ee_status *enc_ext_resp)
Receive and parse Server Encrypted Extensions.
- **ret getServerCertVerify** (TLS_session *session, octad *SCVsig, int &sigalg)
Get Server proof that he owns the Certificate, by receiving and verifying its signature on transcript hash.
- **ret getServerFinished** (TLS_session *session, octad *HFIN)
Get final handshake message from Server, a HMAC on the transcript hash.
- **ret getServerHello** (TLS_session *session, int &kex, octad *CK, octad *PK, int &pskid)
Receive and parse initial Server Hello.
- **ret getCheckServerCertificateChain** (TLS_session *session, octad *PUBKEY, octad *SIG)
Receive and check certificate chain.
- **ret getCertificateRequest** (TLS_session *session, bool context, credential *Credential)
process a Certificate Request

6.9.1 Detailed Description

Process Input received from the Server.

Author

Mike Scott

6.9.2 Function Documentation

6.9.2.1 parseoctad()

```
ret parseoctad (  
    octad * E,  
    int len,  
    octad * M,  
    int & ptr ) [extern]
```

Parse out an octad from a pointer into an octad.

Parameters

<i>E</i>	the output octad copied out from the octad M
<i>len</i>	the expected length of the output octad E
<i>M</i>	the input octad
<i>ptr</i>	a pointer into M, which advances after use

Returns

the actual length of E extracted, and an error flag

6.9.2.2 parsebytes()

```
ret parsebytes (  
    char * e,  
    int len,  
    octad * M,  
    int & ptr ) [extern]
```

Parse out byte array from a pointer into an octad.

Parameters

<i>e</i>	the output byte array copied out from the octad M
<i>len</i>	the expected length of e
<i>M</i>	the input octad
<i>ptr</i>	a pointer into M, which advances after use

Returns

the actual length of *e* extracted, and an error flag

6.9.2.3 parseInt()

```
ret parseInt (
    octad * M,
    int len,
    int & ptr ) [extern]
```

Parse out an unsigned integer from a pointer into an octad.

Parameters

<i>M</i>	the input octad
<i>len</i>	the number of bytes in integer
<i>ptr</i>	a pointer into <i>M</i> , which advances after use

Returns

the integer value, and an error flag

6.9.2.4 parseoctadptr()

```
ret parseoctadptr (
    octad * E,
    int len,
    octad * M,
    int & ptr ) [extern]
```

Return a pointer to an octad from a pointer into an octad.

Parameters

<i>E</i>	a pointer to an octad contained within an octad <i>M</i>
<i>len</i>	the expected length of the octad <i>E</i>
<i>M</i>	the input octad
<i>ptr</i>	a pointer into <i>M</i> , which advances after use

Returns

the actual length of *E*, and an error flag

6.9.2.5 getServerRecord()

```
int getServerRecord (
    TLS_session * session ) [extern]
```

Read a record from the Server, a fragment of a full protocol message.

Parameters

<i>session</i>	the TLS session structure
----------------	---------------------------

Returns

a positive indication of the record type, or a negative error return

6.9.2.6 parseIntorPull()

```
ret parseIntorPull (
    TLS_session * session,
    int len ) [extern]
```

Parse out an unsigned integer from a pointer into an octad, if necessary pulling in a new fragment.

Parameters

<i>session</i>	the TLS session structure
<i>len</i>	the number of bytes in integer

Returns

the unsigned integer, and an error flag

6.9.2.7 parseoctadorPull()

```
ret parseoctadorPull (
    TLS_session * session,
    octad * O,
    int len ) [extern]
```

Parse out an octad from a pointer into an octad, if necessary pulling in a new fragment.

Parameters

<i>session</i>	the TLS session structure
<i>O</i>	the output octad
<i>len</i>	the expected length of the output octad O

Returns

the actual length of O extracted, and an error flag

6.9.2.8 parsebytesorPull()

```
ret parsebytesorPull (
    TLS_session * session,
```

```
char * o,
int len ) [extern]
```

Parse out a byte array from a pointer into an octad, if necessary pulling in a new fragment.

Parameters

<i>session</i>	the TLS session structure
<i>o</i>	the output bytes
<i>len</i>	the expected length of the output

Returns

the actual length of o extracted, and an error flag

6.9.2.9 parseoctadorPullptrX()

```
ret parseoctadorPullptrX (
    TLS_session * session,
    octad * O,
    int len ) [extern]
```

Return a pointer to an octad from a pointer into an octad, if necessary pulling in a new fragment.

Parameters

<i>session</i>	the TLS session structure
<i>O</i>	a pointer to an octad contained within an octad IO
<i>len</i>	the expected length of the octad O

Returns

the actual length of O extracted, and an error flag

6.9.2.10 badResponse()

```
bool badResponse (
    TLS_session * session,
    ret r ) [extern]
```

Process response from server input.

Parameters

<i>session</i>	the TLS1.3 session structure
<i>r</i>	return value to be processed

Returns

true, if its a bad response requiring an abort

6.9.2.11 seeWhatsNext()

```
ret seeWhatsNext (
    TLS_session * session ) [extern]
```

Identify type of incoming message.

Parameters

<i>session</i>	the TLS session structure
----------------	---------------------------

Returns

negative error, zero for OK, or positive for message type

6.9.2.12 getServerEncryptedExtensions()

```
ret getServerEncryptedExtensions (
    TLS_session * session,
    ee_status * enc_ext_expt,
    ee_status * enc_ext_resp ) [extern]
```

Receive and parse Server Encrypted Extensions.

Parameters

<i>session</i>	the TLS session structure
<i>enc_ext_expt</i>	ext structure containing server expectations
<i>enc_ext_resp</i>	ext structure containing server responses

Returns

response structure

6.9.2.13 getServerCertVerify()

```
ret getServerCertVerify (
    TLS_session * session,
    octad * SCVSIg,
    int & sigalg ) [extern]
```

Get Server proof that he owns the Certificate, by receiving and verifying its signature on transcript hash.

Parameters

<i>session</i>	the TLS session structure
<i>SCVSIG</i>	the received signature on the transcript hash
<i>sigalg</i>	the type of the received signature

Returns

response structure

6.9.2.14 getServerFinished()

```
ret getServerFinished (
    TLS_session * session,
    octad * HFIN ) [extern]
```

Get final handshake message from Server, a HMAC on the transcript hash.

Parameters

<i>session</i>	the TLS session structure
<i>HFIN</i>	an octad containing HMAC on transcript as calculated by Server

Returns

response structure

6.9.2.15 getServerHello()

```
ret getServerHello (
    TLS_session * session,
    int & kex,
    octad * CK,
    octad * PK,
    int & pskid ) [extern]
```

Receive and parse initial Server Hello.

Parameters

<i>session</i>	the TLS session structure
<i>kex</i>	key exchange data
<i>CK</i>	an output Cookie
<i>PK</i>	the key exchange public value supplied by the Server
<i>pskid</i>	indicates if a pre-shared key was accepted, otherwise -1

Returns

response structure

6.9.2.16 getCheckServerCertificateChain()

```
ret getCheckServerCertificateChain (
    TLS_session * session,
    octad * PUBKEY,
    octad * SIG ) [extern]
```

Receive and check certificate chain.

Parameters

<i>session</i>	the TLS session structure
<i>PUBKEY</i>	the public key extracted from the Server certificate
<i>SIG</i>	signature (supplied as workspace)

Returns

response structure

6.9.2.17 getCertificateRequest()

```
ret getCertificateRequest (
    TLS_session * session,
    bool context,
    credential * Credential ) [extern]
```

process a Certificate Request

Parameters

<i>session</i>	the TLS session structure
<i>context</i>	true if expecting a context
<i>Credential</i>	client credential

Returns

response structure

6.10 tls_client_recv.h

[Go to the documentation of this file.](#)

```
00001
00007 // Process input received from Server
00008
00009 #ifndef TLS_CLIENT_RECV_H
```

```

00010 #define TLS_CLIENT_RECV_H
00011
00012 #include "tls_sal.h"
00013 #include "tls1_3.h"
00014 #include "tls_sockets.h"
00015 #include "tls_keys_calc.h"
00016 #include "tls_client_send.h"
00017
00026 extern ret parseoctad(octad *E,int len,octad *M,int &ptr);
00027
00036 extern ret parsebytes(char *e,int len,octad *M,int &ptr);
00037
00045 extern ret parseInt(octad *M,int len,int &ptr);
00046
00055 extern ret parseoctadptr(octad *E,int len,octad *M,int &ptr);
00056
00062 extern int getServerRecord(TLS_session *session);
00063
00070 extern ret parseIntorPull(TLS_session *session,int len);
00071
00079 extern ret parseoctadorPull(TLS_session *session,octad *O,int len);
00080
00081
00089 extern ret parsebytesorPull(TLS_session *session,char *o,int len);
00090
00098 extern ret parseoctadorPullptrX(TLS_session *session,octad *O,int len);
00099
00106 extern bool badResponse(TLS_session *session,ret r);
00107
00113 extern ret seeWhatsNext(TLS_session *session);
00114
00122 extern ret getServerEncryptedExtensions(TLS_session *session,ee_status *enc_ext_expt,ee_status
    *enc_ext_resp);
00123
00131 extern ret getServerCertVerify(TLS_session *session,octad *SCVSIG,int &sigalg);
00132
00139 extern ret getServerFinished(TLS_session *session,octad *HFIN);
00140
00150 extern ret getServerHello(TLS_session *session,/*int &cipher,*/int &kex,octad *CK,octad *PK,int
    &pskid);
00151
00159 extern ret getCheckServerCertificateChain(TLS_session *session,octad *PUBKEY,octad *SIG);
00160
00168 extern ret getCertificateRequest(TLS_session *session,bool context,credential *Credential);
00169
00170
00171
00172 #endif

```

6.11 tls_client_send.h File Reference

Process Output to be sent to the Server.

```

#include "tls_sal.h"
#include "tls1_3.h"
#include "tls_sockets.h"
#include "tls_keys_calc.h"

```

Functions

- void [sendCCCS](#) (TLS_session *session)
Send Change Cipher Suite message.
- void [addHeartbeat](#) (octad *EXT)
Add Heartbeat extension.
- void [addCertAuthorities](#) (octad *EXT)
Add Certificate Authorities extension.
- int [addPreSharedKeyExt](#) (octad *EXT, [unsign32](#) age, octad *IDS, int sha)
Add PreShared Key extension to under-construction Extensions Octet (omitting binder)

- void [addServerNameExt](#) ([octad](#) *EXT, char *servername)
Add Server name extension to under-construction Extensions Octet.
- void [addSupportedGroupsExt](#) ([octad](#) *EXT, int nsg, int *supportedGroups)
Add Supported Groups extension to under-construction Extensions Octet.
- void [addServerRawPublicKey](#) ([octad](#) *EXT)
indicate acceptance of raw server public key
- void [addClientRawPublicKey](#) ([octad](#) *EXT)
indicate acceptance of raw client public key
- void [addSigAlgsExt](#) ([octad](#) *EXT, int nsa, int *sigAlgs)
Add Supported TLS1.3 Signature algorithms to under-construction Extensions Octet.
- void [addSigAlgsCertExt](#) ([octad](#) *EXT, int nsac, int *sigAlgsCert)
Add Supported X.509 Certificate Signature algorithms to under-construction Extensions Octet.
- void [addKeyShareExt](#) ([octad](#) *EXT, int alg, [octad](#) *PK)
Add Key Share extension to under-construction Extensions Octet.
- void [addALPNExt](#) ([octad](#) *EXT, [octad](#) *AP)
Add Application Layer Protocol Negotiation (ALPN) extension to under-construction Extensions Octet.
- void [addMFLExt](#) ([octad](#) *EXT, int mode)
Add Maximum Fragment Length extension to under-construction Extensions Octet.
- void [addRSLExt](#) ([octad](#) *EXT, int size)
Add Record Size Limit extension to under-construction Extensions Octet.
- void [addPSKModesExt](#) ([octad](#) *EXT, int mode)
Add Preshared Key exchange modes extension to under-construction Extensions Octet.
- void [addVersionExt](#) ([octad](#) *EXT, int version)
Add Version extension to under-construction Extensions Octet.
- void [addPadding](#) ([octad](#) *EXT, int n)
Add padding extension to under-construction Extensions Octet.
- void [addCookieExt](#) ([octad](#) *EXT, [octad](#) *CK)
Add Cookie extension to under-construction Extensions Octet.
- void [addEarlyDataExt](#) ([octad](#) *EXT)
Indicate desire to send Early Data in under-construction Extensions Octet.
- void [addPostHSAuth](#) ([octad](#) *EXT)
indicate willingness to do post handshake authentication
- int [clientRandom](#) ([octad](#) *RN)
Generate 32-byte random octad.
- int [cipherSuites](#) ([octad](#) *CS, int ncs, int *ciphers)
Build a cipher-suites octad from supported ciphers.
- void [sendRecord](#) ([TLS_session](#) *session, int rectype, int version, [octad](#) *DATA, bool flush)
Send a record to server.
- void [sendClientMessage](#) ([TLS_session](#) *session, int rectype, int version, [octad](#) *CM, [octad](#) *EXT, bool flush)
Send a generic client message (as a single record) to the Server.
- void [sendHeartbeatRequest](#) ([TLS_session](#) *session)
Send a Heartbeat request record.
- void [sendZeroRecord](#) ([TLS_session](#) *session)
Send zero length application fragment - to bewilder the enemy.
- void [sendBinder](#) ([TLS_session](#) *session, [octad](#) *BND)
Send a preshared key binder message to the Server.
- void [sendClientHello](#) ([TLS_session](#) *session, int version, [octad](#) *CH, [octad](#) *CRN, bool already_agreed, [octad](#) *EXTENSIONS, int extra, bool resume, bool flush)
Prepare and send Client Hello message to the Server, appending prepared extensions.
- void [sendAlert](#) ([TLS_session](#) *session, int type)
Prepare and send an Alert message to the Server.

- void `sendKeyUpdate` (`TLS_session` *session, int type)
Prepare and send a key update message to the Server.
- void `sendClientFinish` (`TLS_session` *session, `octad` *CHF)
Prepare and send a final handshake Verification message to the Server.
- void `sendClientCertificateChain` (`TLS_session` *session, `octad` *CERTCHAIN)
Prepare and send client certificate message to the Server.
- void `sendClientCertVerify` (`TLS_session` *session, int sigAlg, `octad` *CCVSIG)
Send client Certificate Verify message to the Server.
- void `sendEndOfEarlyData` (`TLS_session` *session)
Indicate End of Early Data in message to the Server.
- int `alert_from_cause` (int rtn)
Maps problem cause to Alert.

6.11.1 Detailed Description

Process Output to be sent to the Server.

Author

Mike Scott

6.11.2 Function Documentation

6.11.2.1 `sendCCCS()`

```
void sendCCCS (
    TLS_session * session ) [extern]
```

Send Change Cipher Suite message.

Parameters

<code>session</code>	the TLS session structure
----------------------	---------------------------

6.11.2.2 `addHeartbeat()`

```
void addHeartbeat (
    octad * EXT ) [extern]
```

Add Heartbeat extension.

Parameters

<code>EXT</code>	the extensions octad which is being built
------------------	---

6.11.2.3 addCertAuthorities()

```
void addCertAuthorities (
    octad * EXT ) [extern]
```

Add Certificate Authorities extension.

Parameters

<i>EXT</i>	the extensions octad which is being built
------------	---

6.11.2.4 addPreSharedKeyExt()

```
int addPreSharedKeyExt (
    octad * EXT,
    unsigned age,
    octad * IDS,
    int sha ) [extern]
```

Add PreShared Key extension to under-construction Extensions Octet (omitting binder)

Parameters

<i>EXT</i>	the extensions octad which is being built
<i>age</i>	the obfuscated age of the preshared key
<i>IDS</i>	the proposed preshared key identity
<i>sha</i>	the hash algorithm used to calculate the HMAC binder

Returns

length of binder to be sent later

6.11.2.5 addServerNameExt()

```
void addServerNameExt (
    octad * EXT,
    char * servername ) [extern]
```

Add Server name extension to under-construction Extensions Octet.

Parameters

<i>EXT</i>	the extensions octad which is being built
<i>servername</i>	the Host name (URL) of the Server

6.11.2.6 addSupportedGroupsExt()

```
void addSupportedGroupsExt (
```

```

    octad * EXT,
    int nsg,
    int * supportedGroups ) [extern]

```

Add Supported Groups extension to under-construction Extensions Octet.

Parameters

<i>EXT</i>	the extensions octad which is being built
<i>nsg</i>	Number of supported groups
<i>supportedGroups</i>	an array of supported groups

6.11.2.7 addServerRawPublicKey()

```

void addServerRawPublicKey (
    octad * EXT ) [extern]

```

indicate acceptance of raw server public key

Parameters

<i>EXT</i>	the extensions octad which is being built
------------	---

6.11.2.8 addClientRawPublicKey()

```

void addClientRawPublicKey (
    octad * EXT ) [extern]

```

indicate acceptance of raw client public key

Parameters

<i>EXT</i>	the extensions octad which is being built
------------	---

6.11.2.9 addSigAlgsExt()

```

void addSigAlgsExt (
    octad * EXT,
    int nsa,
    int * sigAlgs ) [extern]

```

Add Supported TLS1.3 Signature algorithms to under-construction Extensions Octet.

Parameters

<i>EXT</i>	the extensions octad which is being built
<i>nsa</i>	Number of supported signature algorithms
<i>sigAlgs</i>	an array of supported signature algorithms

6.11.2.10 addSigAlgsCertExt()

```
void addSigAlgsCertExt (
    octad * EXT,
    int nsac,
    int * sigAlgsCert ) [extern]
```

Add Supported X.509 Certificate Signature algorithms to under-construction Extensions Octet.

Parameters

<i>EXT</i>	the extensions octad which is being built
<i>nsac</i>	Number of supported signature algorithms
<i>sigAlgsCert</i>	an array of supported signature algorithms

6.11.2.11 addKeyShareExt()

```
void addKeyShareExt (
    octad * EXT,
    int alg,
    octad * PK ) [extern]
```

Add Key Share extension to under-construction Extensions Octet.

Parameters

<i>EXT</i>	the extensions octad which is being built
<i>alg</i>	the suggested key exchange algorithm
<i>PK</i>	the key exchange public value to be sent to the Server

6.11.2.12 addALPNExt()

```
void addALPNExt (
    octad * EXT,
    octad * AP ) [extern]
```

Add Application Layer Protocol Negotiation (ALPN) extension to under-construction Extensions Octet.

Parameters

<i>EXT</i>	the extensions octad which is being built
<i>AP</i>	the IANA sequence associated with the expected protocol

6.11.2.13 addMFLExt()

```
void addMFLExt (
    octad * EXT,
    int mode ) [extern]
```

Add Maximum Fragment Length extension to under-construction Extensions Octet.

Parameters

<i>EXT</i>	the extensions octad which is being built
<i>mode</i>	the proposed maximum fragment size

6.11.2.14 addRSLExt()

```
void addRSLExt (
    octad * EXT,
    int size ) [extern]
```

Add Record Size Limit extension to under-construction Extensions Octet.

Parameters

<i>EXT</i>	the extensions octad which is being built
<i>size</i>	the demanded maximum fragment size

6.11.2.15 addPSKModesExt()

```
void addPSKModesExt (
    octad * EXT,
    int mode ) [extern]
```

Add Preshared Key exchange modes extension to under-construction Extensions Octet.

Parameters

<i>EXT</i>	the extensions octad which is being built
<i>mode</i>	the proposed preshared key mode

6.11.2.16 addVersionExt()

```
void addVersionExt (
    octad * EXT,
    int version ) [extern]
```

Add Version extension to under-construction Extensions Octet.

Parameters

<i>EXT</i>	the extensions octad which is being built
<i>version</i>	the supported TLS version

6.11.2.17 addPadding()

```
void addPadding (
    octad * EXT,
    int n ) [extern]
```

Add padding extension to under-construction Extensions Octet.

Parameters

<i>EXT</i>	the extensions octad which is being built
<i>n</i>	the zero padding length

6.11.2.18 addCookieExt()

```
void addCookieExt (
    octad * EXT,
    octad * CK ) [extern]
```

Add Cookie extension to under-construction Extensions Octet.

Parameters

<i>EXT</i>	the extensions octad which is being built
<i>CK</i>	the cookie octad to be added

6.11.2.19 addEarlyDataExt()

```
void addEarlyDataExt (
    octad * EXT ) [extern]
```

Indicate desire to send Early Data in under-construction Extensions Octet.

Parameters

<i>EXT</i>	the extensions octad which is being built
------------	---

6.11.2.20 addPostHSAuth()

```
void addPostHSAuth (
    octad * EXT ) [extern]
```

indicate willingness to do post handshake authentication

Parameters

<i>EXT</i>	the extensions octad which is being built
------------	---

6.11.2.21 clientRandom()

```
int clientRandom (
    octad * RN ) [extern]
```

Generate 32-byte random octad.

Parameters

<i>RN</i>	the output 32-byte octad
-----------	--------------------------

Returns

length of output octad

6.11.2.22 cipherSuites()

```
int cipherSuites (
    octad * CS,
    int ncs,
    int * ciphers ) [extern]
```

Build a cipher-suites octad from supported ciphers.

Parameters

<i>CS</i>	the output cipher-suite octad
<i>ncs</i>	the number of supported cipher-suites
<i>ciphers</i>	an array of supported cipher-suites

Returns

length of the output octad

6.11.2.23 sendRecord()

```
void sendRecord (
    TLS_session * session,
    int rectype,
    int version,
    octad * DATA,
    bool flush ) [extern]
```

Send a record to server.

Parameters

<i>session</i>	the TLS session structure
<i>rectype</i>	the record type
<i>version</i>	TLS version indication
<i>DATA</i>	the record (message fragment) to be transmitted
<i>flush</i>	is true if record is to be sent immediately

6.11.2.24 sendClientMessage()

```
void sendClientMessage (
    TLS_session * session,
    int rectype,
    int version,
    octad * CM,
    octad * EXT,
    bool flush ) [extern]
```

Send a generic client message (as a single record) to the Server.

Parameters

<i>session</i>	the TLS session structure
<i>rectype</i>	the record type
<i>version</i>	TLS version indication
<i>CM</i>	the client message to be sent
<i>EXT</i>	extensions to be added (or NULL if there are none)
<i>flush</i>	transmit immediately if true

6.11.2.25 sendHeartbeatRequest()

```
void sendHeartbeatRequest (
    TLS_session * session ) [extern]
```

Send a Heartbeat request record.

Parameters

<i>session</i>	the TLS session structure
----------------	---------------------------

6.11.2.26 sendZeroRecord()

```
void sendZeroRecord (
    TLS_session * session ) [extern]
```

Send zero length application fragment - to bewilder the enemy.

Parameters

<i>session</i>	the TLS session structure
----------------	---------------------------

6.11.2.27 sendBinder()

```
void sendBinder (
    TLS_session * session,
    octad * BND ) [extern]
```

Send a preshared key binder message to the Server.

Parameters

<i>session</i>	the TLS session structure
<i>BND</i>	binding HMAC of truncated transcript hash

6.11.2.28 sendClientHello()

```
void sendClientHello (
    TLS_session * session,
    int version,
    octad * CH,
    octad * CRN,
    bool already_agreed,
    octad * EXTENSIONS,
    int extra,
    bool resume,
    bool flush ) [extern]
```

Prepare and send Client Hello message to the Server, appending prepared extensions.

Parameters

<i>session</i>	the TLS session structure
<i>version</i>	TLS version indication
<i>CH</i>	workspace octad in which to build client Hello
<i>CRN</i>	Random bytes
<i>already_agreed</i>	true if cipher suite previously negotiated, else false
<i>EXTENSIONS</i>	pre-prepared extensions
<i>extra</i>	length of preshared key binder to be sent later
<i>resume</i>	true if this hello is for handshae resumption
<i>flush</i>	transmit immediately

6.11.2.29 sendAlert()

```
void sendAlert (
    TLS_session * session,
    int type ) [extern]
```

Prepare and send an Alert message to the Server.

Parameters

<i>session</i>	the TLS session structure
<i>type</i>	the type of the Alert

6.11.2.30 sendKeyUpdate()

```
void sendKeyUpdate (
    TLS_session * session,
    int type ) [extern]
```

Prepare and send a key update message to the Server.

Parameters

<i>session</i>	the TLS session structure
<i>type</i>	the type of the update

6.11.2.31 sendClientFinish()

```
void sendClientFinish (
    TLS_session * session,
    octad * CHF ) [extern]
```

Prepare and send a final handshake Verification message to the Server.

Parameters

<i>session</i>	the TLS session structure
<i>CHF</i>	the client verify data HMAC

6.11.2.32 sendClientCertificateChain()

```
void sendClientCertificateChain (
    TLS_session * session,
    octad * CERTCHAIN ) [extern]
```

Prepare and send client certificate message to the Server.

Parameters

<i>session</i>	the TLS session structure
<i>CERTCHAIN</i>	the client certificate chain

6.11.2.33 sendClientCertVerify()

```
void sendClientCertVerify (
    TLS_session * session,
    int sigAlg,
    octad * CCVSIG ) [extern]
```

Send client Certificate Verify message to the Server.

Parameters

<i>session</i>	the TLS session structure
<i>sigAlg</i>	the client's digital signature algorithm
<i>CCVSIG</i>	the client's signature

6.11.2.34 sendEndOfEarlyData()

```
void sendEndOfEarlyData (
    TLS_session * session ) [extern]
```

Indicate End of Early Data in message to the Server.

Parameters

<i>session</i>	the TLS session structure
----------------	---------------------------

6.11.2.35 alert_from_cause()

```
int alert_from_cause (
    int rtn ) [extern]
```

Maps problem cause to Alert.

Parameters

<i>rtn</i>	the cause of a problem (a function error return)
------------	--

Returns

type of Alert that should be sent to Server

6.12 tls_client_send.h

[Go to the documentation of this file.](#)

```
00001
00008 // Process output sent to Server
00009 #ifndef TLS_CLIENT_SEND_H
00010 #define TLS_CLIENT_SEND_H
00011
00012 #include "tls_sal.h"
00013 #include "tls1_3.h"
00014 #include "tls_sockets.h"
00015 #include "tls_keys_calc.h"
00016
00021 extern void sendCCCS(TLS_session *session);
00022
00023
00028 extern void addHeartbeat(octad *EXT);
00029
00034 extern void addCertAuthorities(octad *EXT);
00035
00036
00045 extern int addPreSharedKeyExt(octad *EXT, unsigned age, octad *IDS, int sha);
```

```

00046
00052 extern void addServerNameExt(octad *EXT, char *servername);
00053
00060 extern void addSupportedGroupsExt(octad *EXT, int nsg, int *supportedGroups);
00061
00066 extern void addServerRawPublicKey(octad *EXT);
00071 extern void addClientRawPublicKey(octad *EXT);
00072
00073
00080 extern void addSigAlgsExt(octad *EXT, int nsa, int *sigAlgs);
00081
00088 extern void addSigAlgsCertExt(octad *EXT, int nsac, int *sigAlgsCert);
00089
00090
00097 extern void addKeyShareExt(octad *EXT, int alg, octad *PK);
00098
00099
00105 extern void addALPNExt(octad *EXT, octad *AP);
00106
00107
00113 extern void addMFLExt(octad *EXT, int mode);
00114
00120 extern void addRSLEExt(octad *EXT, int size);
00121
00127 extern void addPSKModesExt(octad *EXT, int mode);
00128
00134 extern void addVersionExt(octad *EXT, int version);
00135
00141 extern void addPadding(octad *EXT, int n);
00142
00148 extern void addCookieExt(octad *EXT, octad *CK);
00149
00154 extern void addEarlyDataExt(octad *EXT);
00155
00160 extern void addPostHSAuth(octad *EXT);
00161
00167 extern int clientRandom(octad *RN);
00168
00176 extern int cipherSuites(octad *CS, int ncs, int *ciphers);
00177
00186 extern void sendRecord(TLS_session *session, int rectype, int version, octad *DATA, bool flush);
00187
00197 extern void sendClientMessage(TLS_session *session, int rectype, int version, octad *CM, octad *EXT, bool
flush);
00198
00203 extern void sendHeartbeatRequest(TLS_session *session);
00204
00209 extern void sendZeroRecord(TLS_session *session);
00210
00211
00217 extern void sendBinder(TLS_session *session, octad *BND);
00218
00231 extern void sendClientHello(TLS_session *session, int version, octad *CH, octad *CRN, bool
already_agreed, octad *EXTENSIONS, int extra, bool resume, bool flush);
00232
00238 extern void sendAlert(TLS_session *session, int type);
00239
00240
00246 extern void sendKeyUpdate(TLS_session *session, int type);
00247
00253 extern void sendClientFinish(TLS_session *session, octad *CHF);
00254
00260 extern void sendClientCertificateChain(TLS_session *session, octad *CERTCHAIN);
00261
00268 extern void sendClientCertVerify(TLS_session *session, int sigAlg, octad *CCVSIG);
00269
00270
00275 extern void sendEndOfEarlyData(TLS_session *session);
00276
00282 extern int alert_from_cause(int rtn);
00283 #endif

```

6.13 tls_keys_calc.h File Reference

TLS 1.3 crypto support functions.

```

#include "tls1_3.h"
#include "tls_sal.h"
#include "tls_client_recv.h"

```

Functions

- void `initTranscriptHash` (`TLS_session *session`)
Initialise Transcript hash.
- void `runningHash` (`TLS_session *session`, `octad *O`)
Accumulate octad into ongoing hashing.
- void `runningHashIO` (`TLS_session *session`)
Accumulate transcript hash from IO buffer.
- void `rewindIO` (`TLS_session *session`)
rewind the IO buffer
- void `runningHashIOrewind` (`TLS_session *session`)
Accumulate transcript hash and from IO buffer, and rewind IO buffer.
- void `transcriptHash` (`TLS_session *session`, `octad *O`)
Output current hash value.
- void `runningSyntheticHash` (`TLS_session *session`, `octad *O`, `octad *E`)
Calculate special synthetic hash calculation for first clientHello after retry request (RFC 8446 section 4.4.1)
- void `initCryptoContext` (`crypto *C`)
Initiate a Crypto Context.
- void `updateCryptoContext` (`crypto *C`, `octad *K`, `octad *IV`)
Build a Crypto Context.
- void `incrementCryptoContext` (`crypto *C`)
Increment a Crypto Context for the next record, updating IV.
- void `createCryptoContext` (int cipher, `octad *TS`, `crypto *context`)
Create a crypto context from an input raw Secret and an agreed cipher_suite.
- void `createSendCryptoContext` (`TLS_session *session`, `octad *TS`)
Build a crypto context for transmission from an input raw Secret and an agreed cipher_suite.
- void `createRecvCryptoContext` (`TLS_session *session`, `octad *TS`)
Build a crypto context for reception from an input raw Secret and an agreed cipher_suite.
- void `recoverPSK` (`TLS_session *session`)
Recover pre-shared key from the Resumption Master Secret and store with ticket.
- void `deriveEarlySecrets` (int htype, `octad *PSK`, `octad *ES`, `octad *BKE`, `octad *BKR`)
Extract Early Secret Key and Binder Key from Preshared Key (External or Resumption)
- void `deriveLaterSecrets` (int htype, `octad *H`, `octad *ES`, `octad *CETS`, `octad *EEMS`)
Extract more secrets from Early Secret.
- void `deriveHandshakeSecrets` (`TLS_session *session`, `octad *SS`, `octad *ES`, `octad *H`)
Extract Handshake Secret from Shared Secret and Early Secret. Use Handshake Secret to extract Client and Server Handshake Traffic secrets.
- void `deriveApplicationSecrets` (`TLS_session *session`, `octad *SFH`, `octad *CFH`, `octad *EMS`)
Extract Application Secret from Handshake Secret and Early Secret. Use Handshake Secret to extract Client and Server Application Traffic secrets.
- void `deriveUpdatedKeys` (`crypto *context`, `octad *TS`)
Perform a Key Update on a crypto context.
- bool `checkVerifierData` (int htype, `octad *SF`, `octad *STS`, `octad *H`)
Test if data from Server is verified using server traffic secret and a transcript hash.
- void `deriveVerifierData` (int htype, `octad *SF`, `octad *CTS`, `octad *H`)
Create handshake verification data for Client to send to Server from client traffic secret and a transcript hash.
- bool `checkServerCertVerifier` (int sigalg, `octad *SCVSIG`, `octad *H`, `octad *CERTPK`)
verify Server's signature on protocol transcript
- void `createClientCertVerifier` (int sigAlg, `octad *H`, `octad *KEY`, `octad *CCVSIG`)
Create Cert Verify message, as a digital signature on some TLS1.3 specific message+transcript hash.

6.13.1 Detailed Description

TLS 1.3 crypto support functions.

Author

Mike Scott

6.13.2 Function Documentation

6.13.2.1 initTranscriptHash()

```
void initTranscriptHash (
    TLS_session * session ) [extern]
```

Initialise Transcript hash.

Parameters

<i>session</i>	the TLS session structure
----------------	---------------------------

6.13.2.2 runningHash()

```
void runningHash (
    TLS_session * session,
    octad * O ) [extern]
```

Accumulate octad into ongoing hashing.

Parameters

<i>session</i>	the TLS session structure
<i>O</i>	an octad to be included in hash

6.13.2.3 runningHashIO()

```
void runningHashIO (
    TLS_session * session ) [extern]
```

Accumulate transcript hash from IO buffer.

Parameters

<i>session</i>	the TLS session structure
----------------	---------------------------

6.13.2.4 rewindIO()

```
void rewindIO (
    TLS_session * session ) [extern]
```

rewind the IO buffer

Parameters

<i>session</i>	the TLS session structure
----------------	---------------------------

6.13.2.5 runningHashIOrewind()

```
void runningHashIOrewind (
    TLS_session * session ) [extern]
```

Accumulate transcript hash and from IO buffer, and rewind IO buffer.

Parameters

<i>session</i>	the TLS session structure
----------------	---------------------------

6.13.2.6 transcriptHash()

```
void transcriptHash (
    TLS_session * session,
    octad * O ) [extern]
```

Output current hash value.

Parameters

<i>session</i>	the TLS session structure
<i>O</i>	an output octad containing current hash

6.13.2.7 runningSyntheticHash()

```
void runningSyntheticHash (
    TLS_session * session,
    octad * O,
    octad * E ) [extern]
```

Calculate special synthetic hash calculation for first clientHello after retry request (RFC 8446 section 4.4.1)

Parameters

<i>session</i>	the TLS session structure
<i>O</i>	an octad containing clientHello
<i>E</i>	an octad containing clientHello extensions

6.13.2.8 initCryptoContext()

```
void initCryptoContext (
    crypto * C ) [extern]
```

Initiate a Crypto Context.

Parameters

<i>C</i>	an AEAD encryption context
----------	----------------------------

6.13.2.9 updateCryptoContext()

```
void updateCryptoContext (
    crypto * C,
    octad * K,
    octad * IV ) [extern]
```

Build a Crypto Context.

Parameters

<i>C</i>	an AEAD encryption context
<i>K</i>	an encryption key
<i>IV</i>	an encryption Initialisation Vector

6.13.2.10 incrementCryptoContext()

```
void incrementCryptoContext (
    crypto * C ) [extern]
```

Increment a Crypto Context for the next record, updating IV.

Parameters

<i>C</i>	an AEAD encryption context
----------	----------------------------

6.13.2.11 createCryptoContext()

```
void createCryptoContext (
    int cipher,
    octad * TS,
    crypto * context ) [extern]
```

Create a crypto context from an input raw Secret and an agreed cipher_suite.

Parameters

<i>cipher</i>	the chosen cipher site
<i>TS</i>	the input raw secret
<i>context</i>	the output crypto context

6.13.2.12 createSendCryptoContext()

```
void createSendCryptoContext (
    TLS_session * session,
    octad * TS ) [extern]
```

Build a crypto context for transmission from an input raw Secret and an agreed cipher_suite.

Parameters

<i>session</i>	TLS session structure
<i>TS</i>	the input raw secret

6.13.2.13 createRecvCryptoContext()

```
void createRecvCryptoContext (
    TLS_session * session,
    octad * TS ) [extern]
```

Build a crypto context for reception from an input raw Secret and an agreed cipher_suite.

Parameters

<i>session</i>	TLS session structure
<i>TS</i>	the input raw secret

6.13.2.14 recoverPSK()

```
void recoverPSK (
    TLS_session * session ) [extern]
```

Recover pre-shared key from the Resumption Master Secret and store with ticket.

Parameters

<i>session</i>	the TLS session structure
----------------	---------------------------

6.13.2.15 deriveEarlySecrets()

```
void deriveEarlySecrets (
```

```

int htype,
octad * PSK,
octad * ES,
octad * BKE,
octad * BKR ) [extern]

```

Extract Early Secret Key and Binder Key from Preshared Key (External or Resumption)

Parameters

<i>htype</i>	hash algorithm
<i>PSK</i>	the input pre-shared key, or NULL if not available
<i>ES</i>	the output early secret key
<i>BKE</i>	the output external binder key (or NULL if not required)
<i>BKR</i>	the output resumption binder key (or NULL if not required)

6.13.2.16 deriveLaterSecrets()

```

void deriveLaterSecrets (
    int htype,
    octad * H,
    octad * ES,
    octad * CETS,
    octad * EEMS ) [extern]

```

Extract more secrets from Early Secret.

Parameters

<i>htype</i>	hash algorithm
<i>H</i>	a partial transcript hash
<i>ES</i>	the input early secret key
<i>CETS</i>	the output Client Early Traffic Secret (or NULL if not required)
<i>EEMS</i>	the output Early Exporter Master Secret (or NULL if not required)

6.13.2.17 deriveHandshakeSecrets()

```

void deriveHandshakeSecrets (
    TLS_session * session,
    octad * SS,
    octad * ES,
    octad * H ) [extern]

```

Extract Handshake Secret from Shared Secret and Early Secret. Use Handshake Secret to extract Client and Server Handshake Traffic secrets.

Parameters

<i>session</i>	the TLS session structure
<i>SS</i>	input Shared Secret
<i>ES</i>	the input early secret key
<i>H</i>	a partial transcript hash

6.13.2.18 deriveApplicationSecrets()

```
void deriveApplicationSecrets (
    TLS_session * session,
    octad * SFH,
    octad * CFH,
    octad * EMS ) [extern]
```

Extract Application Secret from Handshake Secret and Early Secret. Use Handshake Secret to extract Client and Server Application Traffic secrets.

Parameters

<i>session</i>	the TLS session structure
<i>SFH</i>	an input partial transcript hash
<i>CFH</i>	an input partial transcript hash
<i>EMS</i>	the output External Master Secret (or NULL if not required)

6.13.2.19 deriveUpdatedKeys()

```
void deriveUpdatedKeys (
    crypto * context,
    octad * TS ) [extern]
```

Perform a Key Update on a crypto context.

Parameters

<i>context</i>	an AEAD encryption context
<i>TS</i>	the updated Traffic secret

6.13.2.20 checkVeriferData()

```
bool checkVeriferData (
    int htype,
    octad * SF,
    octad * STS,
    octad * H ) [extern]
```

Test if data from Server is verified using server traffic secret and a transcript hash.

Parameters

<i>htype</i>	hash algorithm
<i>SF</i>	the input verification data from Server
<i>STS</i>	the input Server Traffic Secret
<i>H</i>	the input partial transcript hash

Returns

true is data is verified, else false

6.13.2.21 deriveVerifierData()

```
void deriveVerifierData (
    int htype,
    octad * SF,
    octad * CTS,
    octad * H ) [extern]
```

Create handshake verification data for Client to send to Server from client traffic secret and a transcript hash.

Parameters

<i>htype</i>	hash algorithm
<i>SF</i>	the output verification data
<i>CTS</i>	the input Client Traffic Secret
<i>H</i>	the input partial transcript hash

6.13.2.22 checkServerCertVerifier()

```
bool checkServerCertVerifier (
    int sigalg,
    octad * SCVSIG,
    octad * H,
    octad * CERTPK ) [extern]
```

verify Server's signature on protocol transcript

Parameters

<i>sigalg</i>	the algorithm used for digital signature
<i>SCVSIG</i>	the input signature on the transcript
<i>H</i>	the transcript hash
<i>CERTPK</i>	the Server's public key

Returns

true if signature is verified, else returns false

6.13.2.23 createClientCertVerifier()

```
void createClientCertVerifier (
    int sigAlg,
    octad * H,
```

```
octad * KEY,  
octad * CCVSIG ) [extern]
```

Create Cert Verify message, as a digital signature on some TLS1.3 specific message+transcript hash.

Parameters

<i>sigAlg</i>	the signature algorithm
<i>H</i>	a transcript hash to be signed
<i>KEY</i>	the Client's private key
<i>CCVSIG</i>	the output digital signature

6.14 tls_keys_calc.h

[Go to the documentation of this file.](#)

```

00001
00008 // TLS1.3 crypto support functions
00009 #ifndef TLS_KEYS_CALC_H
00010 #define TLS_KEYS_CALC_H
00011
00012 #include "tls1_3.h"
00013 #include "tls_sal.h"
00014 #include "tls_client_recv.h"
00015
00016 // transcript hash support
00021 extern void initTranscriptHash(TLS_session *session);
00022
00029 extern void runningHash(TLS_session *session, octad *O);
00030
00036 extern void runningHashIO(TLS_session *session);
00037
00043 extern void rewindIO(TLS_session *session);
00044
00045
00051 extern void runningHashIOrewind(TLS_session *session);
00052
00053
00059 extern void transcriptHash(TLS_session *session, octad *O);
00060
00068 extern void runningSyntheticHash(TLS_session *session, octad *O, octad *E);
00069
00074 extern void initCryptoContext(crypto *C);
00075
00082 extern void updateCryptoContext(crypto *C, octad *K, octad *IV);
00083
00088 extern void incrementCryptoContext(crypto *C);
00089
00090
00097 extern void createCryptoContext(int cipher, octad *TS, crypto *context);
00098
00104 extern void createSendCryptoContext(TLS_session *session, octad *TS);
00105
00111 extern void createRecvCryptoContext(TLS_session *session, octad *TS);
00112
00117 extern void recoverPSK(TLS_session *session);
00118
00127 extern void deriveEarlySecrets(int htype, octad *PSK, octad *ES, octad *BKE, octad *BKR);
00128
00137 extern void deriveLaterSecrets(int htype, octad *H, octad *ES, octad *CETS, octad *EEMS);
00138
00146 extern void deriveHandshakeSecrets(TLS_session *session, octad *SS, octad *ES, octad *H);
00147
00155 extern void deriveApplicationSecrets(TLS_session *session, octad *SFH, octad *CFH, octad *EMS);
00156
00162 extern void deriveUpdatedKeys(crypto *context, octad *TS);
00163
00172 extern bool checkVerifierData(int htype, octad *SF, octad *STS, octad *H);
00173
00181 extern void deriveVerifierData(int htype, octad *SF, octad *CTS, octad *H);
00182
00191 extern bool checkServerCertVerifier(int sigalg, octad *SCVSIG, octad *H, octad *CERTPK);
00192
00200 extern void createClientCertVerifier(int sigAlg, octad *H, octad *KEY, octad *CCVSIG);
00201
00202 #endif

```

6.15 tls_logger.h File Reference

TLS 1.3 logging.

```
#include <string.h>
#include "tls1_3.h"
#include "tls_x509.h"
```

Functions

- void [myprintf](#) (char *s)
internal printf function - all output funnels through this function
- void [log](#) (int logit, char *preamble, char *string, [unsign32](#) info, [octad](#) *O)
basic logging function
- void [logServerHello](#) (int cipher_suite, int pskid, [octad](#) *PK, [octad](#) *CK)
logging the Server hello
- void [logTicket](#) ([ticket](#) *T)
logging a resumption ticket
- void [logEncExt](#) ([ee_status](#) *e, [ee_status](#) *r)
logging server extended extensions responses vs expectations
- void [logCert](#) ([octad](#) *CERT)
logging a Certificate in standard base 64 format
- void [logCertDetails](#) ([octad](#) *PUBKEY, [pktype](#) pk, [octad](#) *SIG, [pktype](#) sg, [octad](#) *ISSUER, [octad](#) *SUBJECT)
logging Certificate details
- void [logServerResponse](#) ([ret](#) r)
log client processing of a Server response
- void [logAlert](#) (int detail)
log Server Alert
- void [logCipherSuite](#) (int cipher_suite)
log Cipher Suite
- void [logKeyExchange](#) (int kex)
log Key Exchange Group
- void [logSigAlg](#) (int sigAlg)
log Signature Algorithm

6.15.1 Detailed Description

TLS 1.3 logging.

Author

Mike Scott

6.15.2 Function Documentation

6.15.2.1 myprintf()

```
void myprintf (
    char * s )    [extern]
```

internal printf function - all output funnels through this function

Parameters

<i>s</i>	a string to be output
----------	-----------------------

6.15.2.2 log()

```
void log (
    int logit,
    char * preamble,
    char * string,
    unsigned int info,
    octad * O ) [extern]
```

basic logging function

Parameters

<i>logit</i>	logging level
<i>preamble</i>	a string to be output
<i>string</i>	another string, or a format specifier for info, or NULL
<i>info</i>	an integer to be output
<i>O</i>	an octad to be output (or NULL)

6.15.2.3 logServerHello()

```
void logServerHello (
    int cipher_suite,
    int pskid,
    octad * PK,
    octad * CK ) [extern]
```

logging the Server hello

Parameters

<i>cipher_suite</i>	the chosen cipher suite
<i>pskid</i>	the chosen preshared key (or -1 if none)
<i>PK</i>	the Server Public Key
<i>CK</i>	a Cookie (if any)

6.15.2.4 logTicket()

```
void logTicket (
    ticket * T ) [extern]
```

logging a resumption ticket

Parameters

<i>T</i>	a resumption ticket
----------	---------------------

6.15.2.5 logEncExt()

```
void logEncExt (
    ee_status * e,
    ee_status * r ) [extern]
```

logging server extended extensions responses vs expectations

Parameters

<i>e</i>	structure containing server expectations
<i>r</i>	structure containing server responses

6.15.2.6 logCert()

```
void logCert (
    octad * CERT ) [extern]
```

logging a Certificate in standard base 64 format

Parameters

<i>CERT</i>	the certificate to be logged
-------------	------------------------------

6.15.2.7 logCertDetails()

```
void logCertDetails (
    octad * PUBKEY,
    pktype pk,
    octad * SIG,
    pktype sg,
    octad * ISSUER,
    octad * SUBJECT ) [extern]
```

logging Certificate details

Parameters

<i>PUBKEY</i>	the certificate public key octad
<i>pk</i>	the public key type
<i>SIG</i>	the signature on the certificate
<i>sg</i>	the signature type
<i>ISSUER</i>	the (composite) certificate issuer
<i>SUBJECT</i>	the (composite) certificate subject

6.15.2.8 logServerResponse()

```
void logServerResponse (
    ret r ) [extern]
```

log client processing of a Server response

Parameters

<i>r</i>	the Server response
----------	---------------------

6.15.2.9 logAlert()

```
void logAlert (
    int detail ) [extern]
```

log Server Alert

Parameters

<i>detail</i>	the server's alert code
---------------	-------------------------

6.15.2.10 logCipherSuite()

```
void logCipherSuite (
    int cipher_suite ) [extern]
```

log Cipher Suite

Parameters

<i>cipher_suite</i>	the Cipher Suite to be logged
---------------------	-------------------------------

6.15.2.11 logKeyExchange()

```
void logKeyExchange (
    int kex ) [extern]
```

log Key Exchange Group

Parameters

<i>kex</i>	the Key Exchange Group to be logged
------------	-------------------------------------

6.15.2.12 logSigAlg()

```
void logSigAlg (
    int sigAlg ) [extern]
```

log Signature Algorithm

Parameters

<i>sigAlg</i>	the Signature Algorithm to be logged
---------------	--------------------------------------

6.16 tls_logger.h

[Go to the documentation of this file.](#)

```
00001
00007 // Log protocol progress
00008 #ifndef TLS_LOGGER_H
00009 #define TLS_LOGGER_H
00010
00011 #include <string.h>
00012 #include "tls1_3.h"
00013 #include "tls_x509.h"
00014
00019 extern void myprintf(char *s);
00020
00029 extern void log(int logit, char *preamble, char *string, unsigned info, octad *O);
00030
00038 extern void logServerHello(int cipher_suite, int pskid, octad *PK, octad *CK);
00039
00044 extern void logTicket(ticket *T);
00045
00051 extern void logEncExt(ee_status *e, ee_status *r);
00052
00057 extern void logCert(octad *CERT);
00058
00068 extern void logCertDetails(octad *PUBKEY, pktype pk, octad *SIG, pktype sg, octad *ISSUER, octad *SUBJECT);
00069
00074 extern void logServerResponse(ret r);
00075
00080 extern void logAlert(int detail);
00081
00086 extern void logCipherSuite(int cipher_suite);
00087
00092 extern void logKeyExchange(int kex);
00093
00098 extern void logSigAlg(int sigAlg);
00099
00100 #endif
```

6.17 tls_octads.h File Reference

octad handling routines - octads don't overflow, they truncate

```
#include <stddef.h>
```

Data Structures

- struct [octad](#)

Safe representation of an octad.

Functions

- unsigned long `millis` ()
read milliseconds from a stop-watch
- void `OCT_append_int` (`octad *O`, unsigned int `x`, int `len`)
Join len bytes of integer x to end of octad O (big endian)
- void `OCT_append_octad` (`octad *O`, `octad *P`)
Join one octad to the end of another.
- bool `OCT_compare` (`octad *O`, `octad *P`)
Compare two octads.
- void `OCT_shift_left` (`octad *O`, int `n`)
Shifts octad left by n bytes.
- void `OCT_kill` (`octad *O`)
Wipe clean an octad.
- void `OCT_from_hex` (`octad *O`, char `*src`)
Convert a hex number to an octad.
- void `OCT_append_string` (`octad *O`, char `*s`)
Join from a C string to end of an octad.
- void `OCT_append_byte` (`octad *O`, int `b`, int `n`)
Join single byte to end of an octad, repeated n times.
- void `OCT_append_bytes` (`octad *O`, char `*s`, int `n`)
Join bytes to end of an octad.
- void `OCT_from_base64` (`octad *O`, char `*b`)
Create an octad from a base64 number.
- void `OCT_reverse` (`octad *O`)
Reverse bytes in an octad.
- void `OCT_truncate` (`octad *O`, int `n`)
Reverse bytes in an octad.
- void `OCT_copy` (`octad *O`, `octad *P`)
Copy one octad into another.
- bool `OCT_output_hex` (`octad *O`, int `max`, char `*s`)
Output octad as hex string.
- bool `OCT_output_string` (`octad *O`, int `max`, char `*s`)
Output octad as C ascii string.
- void `OCT_output_base64` (`octad *O`, int `max`, char `*s`)
Output octad as base64 string.

6.17.1 Detailed Description

octad handling routines - octads don't overflow, they truncate

Author

Mike Scott

6.17.2 Function Documentation

6.17.2.1 millis()

```
unsigned long millis ( ) [extern]
```

read milliseconds from a stop-watch

Returns

milliseconds read from stop-watch

6.17.2.2 OCT_append_int()

```
void OCT_append_int (
    octad * O,
    unsigned int x,
    int len ) [extern]
```

Join len bytes of integer x to end of octad O (big endian)

Parameters

<i>O</i>	octad to be appended to
<i>x</i>	integer to be appended to O
<i>len</i>	number of bytes in m

6.17.2.3 OCT_append_octad()

```
void OCT_append_octad (
    octad * O,
    octad * P ) [extern]
```

Join one octad to the end of another.

Parameters

<i>O</i>	octad to be appended to
<i>P</i>	octad to be joined to the end of O

6.17.2.4 OCT_compare()

```
bool OCT_compare (
    octad * O,
    octad * P ) [extern]
```

Compare two octads.

Parameters

<i>O</i>	first octad to be compared
<i>P</i>	second octad to be compared

Returns

true if equal, else false

6.17.2.5 `OCT_shift_left()`

```
void OCT_shift_left (
    octad * O,
    int n ) [extern]
```

Shifts octad left by n bytes.

Leftmost bytes disappear

Parameters

<i>O</i>	octad to be shifted
<i>n</i>	number of bytes to shift

6.17.2.6 `OCT_kill()`

```
void OCT_kill (
    octad * O ) [extern]
```

Wipe clean an octad.

Parameters

<i>O</i>	octad to be cleared
----------	---------------------

6.17.2.7 `OCT_from_hex()`

```
void OCT_from_hex (
    octad * O,
    char * src ) [extern]
```

Convert a hex number to an octad.

Parameters

<i>O</i>	octad
<i>src</i>	Hex string to be converted

6.17.2.8 OCT_append_string()

```
void OCT_append_string (
    octad * O,
    char * s ) [extern]
```

Join from a C string to end of an octad.

Parameters

<i>O</i>	octad to be written to
<i>s</i>	zero terminated string to be joined to octad

6.17.2.9 OCT_append_byte()

```
void OCT_append_byte (
    octad * O,
    int b,
    int n ) [extern]
```

Join single byte to end of an octad, repeated n times.

Parameters

<i>O</i>	octad to be written to
<i>b</i>	byte to be joined to end of octad
<i>n</i>	number of times b is to be joined

6.17.2.10 OCT_append_bytes()

```
void OCT_append_bytes (
    octad * O,
    char * s,
    int n ) [extern]
```

Join bytes to end of an octad.

Parameters

<i>O</i>	octad to be written to
<i>s</i>	byte array to be joined to end of octad
<i>n</i>	number of bytes to join

6.17.2.11 OCT_from_base64()

```
void OCT_from_base64 (
    octad * O,
    char * b ) [extern]
```

Create an octad from a base64 number.

Parameters

<i>O</i>	octad to be populated
<i>b</i>	zero terminated base64 string

6.17.2.12 OCT_reverse()

```
void OCT_reverse (
    octad * O ) [extern]
```

Reverse bytes in an octad.

Parameters

<i>O</i>	octad to be reversed
----------	----------------------

6.17.2.13 OCT_truncate()

```
void OCT_truncate (
    octad * O,
    int n ) [extern]
```

Reverse bytes in an octad.

Parameters

<i>O</i>	octad to be truncated
<i>n</i>	the new shorter length

6.17.2.14 OCT_copy()

```
void OCT_copy (
    octad * O,
    octad * P ) [extern]
```

Copy one octad into another.

Parameters

<i>O</i>	octad to be copied to
<i>P</i>	octad to be copied from

6.17.2.15 OCT_output_hex()

```
bool OCT_output_hex (
    octad * O,
    int max,
    char * s ) [extern]
```

Output octad as hex string.

Parameters

<i>O</i>	octad to be output
<i>max</i>	the maximum output length
<i>s</i>	the char array to receive output

6.17.2.16 OCT_output_string()

```
bool OCT_output_string (
    octad * O,
    int max,
    char * s ) [extern]
```

Output octad as C ascii string.

Parameters

<i>O</i>	octad to be output
<i>max</i>	the maximum output length
<i>s</i>	the char array to receive output

6.17.2.17 OCT_output_base64()

```
void OCT_output_base64 (
    octad * O,
    int max,
    char * s ) [extern]
```

Output octad as base64 string.

Parameters

<i>O</i>	octad to be output
<i>max</i>	the maximum output length
<i>s</i>	the char array to receive output

6.18 tls_octads.h

[Go to the documentation of this file.](#)

```

00001
00008 #ifndef TLS_OCTADS_H
00009 #define TLS_OCTADS_H
00010
00011 // An octad - "a group or set of eight" - Oxford dictionary
00012
00013 // #define TLS_ARDUINO          /**< Define for Arduino-based implementation */
00014
00015 #include <stddef.h>
00016
00021 #ifndef TLS_ARDUINO
00022 extern unsigned long millis();
00023 #endif
00027 typedef struct
00028 {
00029     int len;
00030     int max;
00031     char *val;
00032 } octad;
00033
00041 extern void OCT_append_int(octad *O, unsigned int x, int len);
00042
00048 extern void OCT_append_octad(octad *O, octad *P);
00049
00056 extern bool OCT_compare(octad *O, octad *P);
00057
00065 extern void OCT_shift_left(octad *O, int n);
00066
00071 extern void OCT_kill(octad *O);
00072
00078 extern void OCT_from_hex(octad *O, char *src);
00079
00085 extern void OCT_append_string(octad *O, char *s);
00086
00093 extern void OCT_append_byte(octad *O, int b, int n);
00094
00101 extern void OCT_append_bytes(octad *O, char *s, int n);
00102
00109 extern void OCT_from_base64(octad *O, char *b);
00110
00115 extern void OCT_reverse(octad *O);
00116
00122 extern void OCT_truncate(octad *O, int n);
00123
00130 extern void OCT_copy(octad *O, octad *P);
00131
00139 extern bool OCT_output_hex(octad *O, int max, char *s);
00140
00148 extern bool OCT_output_string(octad *O, int max, char *s);
00149
00157 extern void OCT_output_base64(octad *O, int max, char *s);
00158 #endif

```

6.19 tls_pqibe.h File Reference

Ducas et al. IBE.

```
#include "core.h"
```

Functions

- void **PQIBE_CCA_ENCRYPT** (char *ID, octet *R32, octet *KEY, octet *CT)
IBE KEM CCA encrypt.
- void **PQIBE_CCA_DECRYPT** (char *ID, const int16_t *csk, octet *CT, octet *KEY)
IBE KEM CCA decrypt.

6.19.1 Detailed Description

Ducas et al. IBE.

Author

Mike Scott

6.19.2 Function Documentation

6.19.2.1 PQIBE_CCA_ENCRYPT()

```
void PQIBE_CCA_ENCRYPT (
    char * ID,
    octet * R32,
    octet * KEY,
    octet * CT ) [extern]
```

IBE KEM CCA encrypt.

Parameters

<i>ID</i>	Identity
<i>R32</i>	32 random bytes
<i>KEY</i>	random session key generated
<i>CT</i>	encapsulating ciphertext

6.19.2.2 PQIBE_CCA_DECRYPT()

```
void PQIBE_CCA_DECRYPT (
    char * ID,
    const int16_t * csk,
    octet * CT,
    octet * KEY ) [extern]
```

IBE KEM CCA decrypt.

Parameters

<i>ID</i>	Identity
<i>csk</i>	secret key
<i>CT</i>	ciphertext
<i>KEY</i>	output session key

6.20 tls_pqibe.h

[Go to the documentation of this file.](#)

```

00001
00007 //
00008 // Lattice-based B&F IBE 128-bit API Functions
00009 // Ducas et al. Method
00010 // Implementation by M.Scott
00011 //
00012
00013 #ifndef PQIBE_H
00014 #define PQIBE_H
00015
00016 #include "core.h"
00017 using namespace core;
00018
00026 extern void PQIBE_CCA_ENCRYPT(char *ID,octet *R32,octet *KEY,octet *CT);
00027
00035 extern void PQIBE_CCA_DECRYPT(char *ID,const int16_t *csk,octet *CT,octet *KEY);
00036
00037 #endif

```

6.21 tls_protocol.h File Reference

TLS 1.3 main client-side protocol functions.

```
#include "tls1_3.h"
```

Functions

- [TLS_session TLS13_start](#) ([Socket](#) *client, char *hostname)
initialise a TLS 1.3 session structure
- void [TLS13_end](#) ([TLS_session](#) *session)
terminate a session structure
- void [TLS13_stop](#) ([TLS_session](#) *session)
stop sending - send CLOSE_NOTIFY and DISCONNECT
- bool [TLS13_connect](#) ([TLS_session](#) *session, [octad](#) *EARLY, [credential](#) *Credential)
TLS 1.3 forge connection.
- void [TLS13_send](#) ([TLS_session](#) *session, [octad](#) *DATA)
TLS 1.3 send data.
- int [TLS13_recv](#) ([TLS_session](#) *session, [octad](#) *DATA, [credential](#) *Credential)
TLS 1.3 receive data.
- int [TLS13_recv_and_check](#) ([TLS_session](#) *session, [octad](#) *DATA)
TLS 1.3 receive data and check for liveness of connection.
- void [TLS13_clean](#) ([TLS_session](#) *session)
TLS 1.3 end session, delete keys, clean up buffers.

6.21.1 Detailed Description

TLS 1.3 main client-side protocol functions.

Author

Mike Scott

6.21.2 Function Documentation

6.21.2.1 TLS13_start()

```
TLS_session TLS13_start (  
    Socket * client,  
    char * hostname ) [extern]
```

initialise a TLS 1.3 session structure

Parameters

<i>client</i>	the socket connection to the Server
<i>hostname</i>	the host name (URL) of the server

Returns

an initialised TLS1.3 session structure

6.21.2.2 TLS13_end()

```
void TLS13_end (
    TLS_session * session ) [extern]
```

terminate a session structure

Parameters

<i>session</i>	the session structure
----------------	-----------------------

6.21.2.3 TLS13_stop()

```
void TLS13_stop (
    TLS_session * session ) [extern]
```

stop sending - send CLOSE_NOTIFY and DISCONNECT

Parameters

<i>session</i>	the session structure
----------------	-----------------------

6.21.2.4 TLS13_connect()

```
bool TLS13_connect (
    TLS_session * session,
    octad * EARLY,
    credential * Credential ) [extern]
```

TLS 1.3 forge connection.

Parameters

<i>session</i>	an initialised TLS session structure
<i>EARLY</i>	some early data to be transmitted
<i>Credential</i>	client credential

Returns

false for failure, true for success

6.21.2.5 TLS13_send()

```
void TLS13_send (
    TLS_session * session,
    octad * DATA ) [extern]
```

TLS 1.3 send data.

Parameters

<i>session</i>	an initialised TLS session structure
<i>DATA</i>	some data to be transmitted

6.21.2.6 TLS13_rcv()

```
int TLS13_rcv (
    TLS_session * session,
    octad * DATA,
    credential * Credential ) [extern]
```

TLS 1.3 receive data.

Parameters

<i>session</i>	an initialised TLS session structure
<i>DATA</i>	that has been received
<i>Credential</i>	client credential

Returns

0 for time-out, negative for error, or length of data successfully received

6.21.2.7 TLS13_rcv_and_check()

```
int TLS13_rcv_and_check (
    TLS_session * session,
    octad * DATA ) [extern]
```

TLS 1.3 receive data and check for liveness of connection.

Parameters

<i>session</i>	an initialised TLS session structure
<i>DATA</i>	that has been received

Returns

0 for time-out, negative for error, or length of data successfully received

6.21.2.8 TLS13_clean()

```
void TLS13_clean (
    TLS_session * session ) [extern]
```

TLS 1.3 end session, delete keys, clean up buffers.

Parameters

<i>session</i>	an initialised TLS session structure
----------------	--------------------------------------

6.22 tls_protocol.h

[Go to the documentation of this file.](#)

```
00001
00007 // Main TLS 1.3 Protocol
00008
00009 #ifndef TLS_PROTOCOL_H
00010 #define TLS_PROTOCOL_H
00011
00012 #include "tls1_3.h" // for sockets, octads and session structure
00013
00014
00021 extern TLS_session TLS13_start(Socket *client, char *hostname);
00022
00027 extern void TLS13_end(TLS_session *session);
00028
00029
00034 extern void TLS13_stop(TLS_session *session);
00035
00036
00044 extern bool TLS13_connect(TLS_session *session, octad *EARLY, credential *Credential);
00045
00051 extern void TLS13_send(TLS_session *session, octad *DATA);
00052
00060 extern int TLS13_rcv(TLS_session *session, octad *DATA, credential *Credential);
00061
00068 extern int TLS13_rcv_and_check(TLS_session *session, octad *DATA);
00069
00070
00075 extern void TLS13_clean(TLS_session *session);
00076 #endif
```

6.23 tls_sal.h File Reference

Security Abstraction Layer for TLS.

```
#include "tls1_3.h"
```

Functions

- char * [SAL_name](#) ()
Return name of SAL provider.
- int [SAL_ciphers](#) (int *ciphers)
Return supported ciphers.
- int [SAL_groups](#) (int *groups)
Return supported groups in preferred order.
- int [SAL_sigs](#) (int *sigAlgs)
Return supported TLS signature algorithms in preferred order.
- int [SAL_sigCerts](#) (int *sigAlgsCert)
Return supported TLS signature algorithms for Certificates in preferred order.
- bool [SAL_initLib](#) ()
Initialise library for use.
- void [SAL_endLib](#) ()
finish use of library
- int [SAL_hashType](#) (int cipher_suite)
return hash type associated with a cipher suite
- int [SAL_hashLen](#) (int hash_type)
return output length of hash function associated with a hash type
- int [SAL_aeadKeylen](#) (int cipher_suite)
return key length associated with a cipher suite
- int [SAL_aeadTaglen](#) (int cipher_suite)
return authentication tag length associated with a cipher suite
- int [SAL_randomByte](#) ()
get a random byte
- void [SAL_randomOctad](#) (int len, octad *R)
get a random octad
- void [SAL_hkdfExtract](#) (int sha, octad *PRK, octad *SALT, octad *IKM)
HKDF Extract function.
- void [SAL_hkdfExpand](#) (int htype, int olen, octad *OKM, octad *PRK, octad *INFO)
Special HKDF Expand function (for TLS)
- void [SAL_hmac](#) (int htype, octad *T, octad *K, octad *M)
simple HMAC function
- void [SAL_hashNull](#) (int sha, octad *H)
simple HASH of nothing function
- void [SAL_hashInit](#) (int hlen, unihash *h)
Initiate Hashing context.
- void [SAL_hashProcessArray](#) (unihash *h, char *b, int len)
Hash process an array of bytes.
- int [SAL_hashOutput](#) (unihash *h, char *d)
Hash finish and output.
- void [SAL_aeadEncrypt](#) (crypto *send, int hdrlen, char *hdr, int ptlen, char *pt, octad *TAG)
AEAD encryption.
- bool [SAL_aeadDecrypt](#) (crypto *recv, int hdrlen, char *hdr, int ctlen, char *ct, octad *TAG)
AEAD decryption.
- void [SAL_generateKeyPair](#) (int group, octad *SK, octad *PK)
generate a public/private key pair in an approved group for a key exchange
- bool [SAL_generateSharedSecret](#) (int group, octad *SK, octad *PK, octad *SS)
generate a Diffie-Hellman shared secret
- bool [SAL_tlsSignatureVerify](#) (int sigAlg, octad *TRANS, octad *SIG, octad *PUBKEY)
Verify a generic TLS signature.
- void [SAL_tlsSignature](#) (int sigAlg, octad *KEY, octad *TRANS, octad *SIG)
Apply a generic TLS transcript signature.

6.23.1 Detailed Description

Security Abstraction Layer for TLS.

Author

Mike Scott

6.23.2 Function Documentation

6.23.2.1 SAL_name()

```
char * SAL_name ( ) [extern]
```

Return name of SAL provider.

Returns

name of SAL provider

6.23.2.2 SAL_ciphers()

```
int SAL_ciphers (
    int * ciphers ) [extern]
```

Return supported ciphers.

Parameters

<i>ciphers</i>	array of supported ciphers in preferred order
----------------	---

Returns

number of supported ciphers

6.23.2.3 SAL_groups()

```
int SAL_groups (
    int * groups ) [extern]
```

Return supported groups in preferred order.

Parameters

<i>groups</i>	array of supported groups
---------------	---------------------------

Returns

number of supported groups

6.23.2.4 SAL_sigs()

```
int SAL_sigs (
    int * sigAlgs ) [extern]
```

Return supported TLS signature algorithms in preferred order.

Parameters

<i>sigAlgs</i>	array of supported signature algorithms
----------------	---

Returns

number of supported groups

6.23.2.5 SAL_sigCerts()

```
int SAL_sigCerts (
    int * sigAlgsCert ) [extern]
```

Return supported TLS signature algorithms for Certificates in preferred order.

Parameters

<i>sigAlgsCert</i>	array of supported signature algorithms for Certificates
--------------------	--

Returns

number of supported groups

6.23.2.6 SAL_initLib()

```
bool SAL_initLib ( ) [extern]
```

Initialise library for use.

Returns

return true if successful, else false

6.23.2.7 SAL_hashType()

```
int SAL_hashType (
    int cipher_suite ) [extern]
```

return hash type associated with a cipher suite

Parameters

<i>cipher_suite</i>	a TLS cipher suite
---------------------	--------------------

Returns

hash function output length

6.23.2.8 SAL_hashLen()

```
int SAL_hashLen (
    int hash_type ) [extern]
```

return output length of hash function associated with a hash type

Parameters

<i>hash_type</i>	a TLS hash type
------------------	-----------------

Returns

hash function output length

6.23.2.9 SAL_aeadKeylen()

```
int SAL_aeadKeylen (
    int cipher_suite )
```

return key length associated with a cipher suite

Parameters

<i>cipher_suite</i>	a TLS cipher suite
---------------------	--------------------

Returns

key length

6.23.2.10 SAL_aeadTaglen()

```
int SAL_aeadTaglen (
    int cipher_suite )
```

return authentication tag length associated with a cipher suite

Parameters

<i>cipher_suite</i>	a TLS cipher suite
---------------------	--------------------

Returns

tag length

6.23.2.11 SAL_randomByte()

```
int SAL_randomByte ( ) [extern]
```

get a random byte

Returns

a random byte

6.23.2.12 SAL_randomOctad()

```
void SAL_randomOctad (
    int len,
    octad * R ) [extern]
```

get a random octad

Parameters

<i>len</i>	number of random bytes
<i>R</i>	octad to be filled with random bytes

6.23.2.13 SAL_hkdfExtract()

```
void SAL_hkdfExtract (
    int sha,
    octad * PRK,
    octad * SALT,
    octad * IKM ) [extern]
```

HKDF Extract function.

Parameters

<i>sha</i>	hash algorithm
<i>PRK</i>	an output Key
<i>SALT</i>	public input salt
<i>IKM</i>	raw secret keying material

6.23.2.14 SAL_hkdfExpand()

```
void SAL_hkdfExpand (
    int htype,
    int olen,
    octad * OKM,
    octad * PRK,
    octad * INFO ) [extern]
```

Special HKDF Expand function (for TLS)

Parameters

<i>htype</i>	hash algorithm
<i>olen</i>	is the desired length of the expanded key
<i>OKM</i>	an expanded output Key
<i>PRK</i>	is the fixed length input key
<i>INFO</i>	is public label information

6.23.2.15 SAL_hmac()

```
void SAL_hmac (
    int htype,
    octad * T,
    octad * K,
    octad * M ) [extern]
```

simple HMAC function

Parameters

<i>htype</i>	hash algorithm
<i>T</i>	an output tag
<i>K</i>	an input key, or salt
<i>M</i>	an input message

6.23.2.16 SAL_hashNull()

```
void SAL_hashNull (
    int sha,
    octad * H ) [extern]
```

simple HASH of nothing function

Parameters

<i>sha</i>	the SHA2 function output length (32,48 or 64)
<i>H</i>	the output hash

6.23.2.17 SAL_hashInit()

```
void SAL_hashInit (
    int hlen,
    unihash * h ) [extern]
```

Initiate Hashing context.

Parameters

<i>hlen</i>	length in bytes of SHA2 hashing output
<i>h</i>	a hashing context

6.23.2.18 SAL_hashProcessArray()

```
void SAL_hashProcessArray (
    unihash * h,
    char * b,
    int len ) [extern]
```

Hash process an array of bytes.

Parameters

<i>h</i>	a hashing context
<i>b</i>	the byte array to be included in hash
<i>len</i>	the array length

6.23.2.19 SAL_hashOutput()

```
int SAL_hashOutput (
    unihash * h,
    char * d ) [extern]
```

Hash finish and output.

Parameters

<i>h</i>	a hashing context
<i>d</i>	the current output digest of an ongoing hashing operation

Returns

hash output length

6.23.2.20 SAL_aeadEncrypt()

```
void SAL_aeadEncrypt (
    crypto * send,
```

```

    int hdrlen,
    char * hdr,
    int ptlen,
    char * pt,
    octad * TAG ) [extern]

```

AEAD encryption.

Parameters

<i>send</i>	the AES key and IV
<i>hdrlen</i>	the length of the header
<i>hdr</i>	the header bytes
<i>ptlen</i>	the plaintext length
<i>pt</i>	the input plaintext and output ciphertext
<i>TAG</i>	the output authentication tag

6.23.2.21 SAL_aeadDecrypt()

```

bool SAL_aeadDecrypt (
    crypto * recv,
    int hdrlen,
    char * hdr,
    int ctlen,
    char * ct,
    octad * TAG ) [extern]

```

AEAD decryption.

Parameters

<i>recv</i>	the AES key and IV
<i>hdrlen</i>	the length of the header
<i>hdr</i>	the header bytes
<i>ctlen</i>	the ciphertext length
<i>ct</i>	the input ciphertext and output plaintext
<i>TAG</i>	the expected authentication tag

Returns

false if tag is wrong, else true

6.23.2.22 SAL_generateKeyPair()

```

void SAL_generateKeyPair (
    int group,
    octad * SK,
    octad * PK ) [extern]

```

generate a public/private key pair in an approved group for a key exchange

Parameters

<i>group</i>	the cryptographic group used to generate the key pair
<i>SK</i>	the output Private Key
<i>PK</i>	the output Public Key

6.23.2.23 SAL_generateSharedSecret()

```
bool SAL_generateSharedSecret (
    int group,
    octad * SK,
    octad * PK,
    octad * SS ) [extern]
```

generate a Diffie-Hellman shared secret

Parameters

<i>group</i>	the cryptographic group used to generate the shared secret
<i>SK</i>	the input client private key
<i>PK</i>	the input server public Key
<i>SS</i>	the output shared secret

Returns

false for all zeros, else true

6.23.2.24 SAL_tlsSignatureVerify()

```
bool SAL_tlsSignatureVerify (
    int sigAlg,
    octad * TRANS,
    octad * SIG,
    octad * PUBKEY ) [extern]
```

Verify a generic TLS signature.

Parameters

<i>sigAlg</i>	the signature type
<i>TRANS</i>	the signed input transcript hash
<i>SIG</i>	the input signature
<i>PUBKEY</i>	the public key used to verify the signature

Returns

true if signature is valid, else false

6.23.2.25 SAL_tlsSignature()

```
void SAL_tlsSignature (
    int sigAlg,
    octad * KEY,
    octad * TRANS,
    octad * SIG ) [extern]
```

Apply a generic TLS transcript signature.

Parameters

<i>sigAlg</i>	the signature type
<i>KEY</i>	the private key used to form the signature
<i>TRANS</i>	the input transcript hash to be signed
<i>SIG</i>	the output signature

6.24 tls_sal.h

[Go to the documentation of this file.](#)

```
00001
00007 // Process input received from Server
00008
00009 #ifndef TLS_SAL_H
00010 #define TLS_SAL_H
00011
00012 // Use MIRACL core library
00013
00014 #include "tls1_3.h"
00015
00020 extern char *SAL_name();
00021
00027 extern int SAL_ciphers(int *ciphers);
00028
00034 extern int SAL_groups(int *groups);
00035
00041 extern int SAL_sigs(int *sigAlgs);
00042
00048 extern int SAL_sigCerts(int *sigAlgsCert);
00049
00054 extern bool SAL_initLib();
00055
00059 extern void SAL_endLib();
00060
00066 extern int SAL_hashType(int cipher_suite);
00067
00073 extern int SAL_hashLen(int hash_type);
00074
00080 int SAL_aeadKeylen(int cipher_suite);
00081
00087 int SAL_aeadTaglen(int cipher_suite);
00088
00093 extern int SAL_randomByte();
00094
00100 extern void SAL_randomOctad(int len, octad *R);
00101
00109 extern void SAL_hkdfExtract(int sha, octad *PRK, octad *SALT, octad *IKM);
00110
00119 extern void SAL_hkdfExpand(int htype, int olen, octad *OKM, octad *PRK, octad *INFO);
00120
00128 extern void SAL_hmac(int htype, octad *T, octad *K, octad *M);
00129
00135 extern void SAL_hashNull(int sha, octad *H);
00136
00137 // hash functions
00138
00144 extern void SAL_hashInit(int hlen, unihash *h);
00145
00152 extern void SAL_hashProcessArray(unihash *h, char *b, int len);
00153
```

```

00154
00161 extern int SAL_hashOutput (unihash *h, char *d);
00162
00172 extern void SAL_aeadEncrypt (crypto *send, int hdrlen, char *hdr, int ptlen, char *pt, octad *TAG);
00173
00184 extern bool SAL_aeadDecrypt (crypto *recv, int hdrlen, char *hdr, int ctlen, char *ct, octad *TAG);
00185
00192 extern void SAL_generateKeyPair (int group, octad *SK, octad *PK);
00193
00202 extern bool SAL_generateSharedSecret (int group, octad *SK, octad *PK, octad *SS);
00203
00204
00213 extern bool SAL_tlsSignatureVerify (int sigAlg, octad *TRANS, octad *SIG, octad *PUBKEY);
00214
00222 extern void SAL_tlsSignature (int sigAlg, octad *KEY, octad *TRANS, octad *SIG);
00223
00224
00225 #endif

```

6.25 tls_sockets.h File Reference

set up sockets for reading and writing

```

#include <string.h>
#include "tls_octads.h"
#include <time.h>
#include <unistd.h>
#include <stdio.h>
#include <sys/socket.h>
#include <arpa/inet.h>
#include <stdlib.h>
#include <netinet/in.h>
#include <netdb.h>
#include <sys/un.h>

```

Data Structures

- class [Socket](#)
Socket instance.

Functions

- int [setclientsock](#) (int port, char *ip, int toms)
create a client socket
- int [getIPaddress](#) (char *ip, char *hostname)
get the IP address from a URL
- void [sendOctad](#) ([Socket](#) *client, octad *B)
send an octad over a socket
- void [sendLen](#) ([Socket](#) *client, int len)
send a 16-bit integer as an octad to Server
- int [getBytes](#) ([Socket](#) *client, char *b, int expected)
receive bytes over a socket sonnection
- int [getInt16](#) ([Socket](#) *client)
receive 16-bit integer from a socket
- int [getInt24](#) ([Socket](#) *client)
receive 24-bit integer from a socket
- int [getBytes](#) ([Socket](#) *client)
receive a single byte from a socket
- int [getOctad](#) ([Socket](#) *client, octad *B, int expected)
receive an octad from a socket

6.25.1 Detailed Description

set up sockets for reading and writing

Author

Mike Scott

6.25.2 Function Documentation

6.25.2.1 setclientsock()

```
int setclientsock (  
    int port,  
    char * ip,  
    int toms ) [extern]
```

create a client socket

Parameters

<i>port</i>	the TCP/IP port on which to connect
<i>ip</i>	the IP address with which to connect
<i>toms</i>	the time-out period in milliseconds

Returns

the socket handle

6.25.2.2 getIPAddress()

```
int getIPAddress (  
    char * ip,  
    char * hostname ) [extern]
```

get the IP address from a URL

Parameters

<i>ip</i>	the IP address
<i>hostname</i>	the input Server name (URL)

Returns

1 for success, 0 for failure

6.25.2.3 sendOctad()

```
void sendOctad (  

```

```
Socket * client,  
octad * B ) [extern]
```

send an octad over a socket

Parameters

<i>client</i>	the socket connection to the Server
<i>B</i>	the octad to be transmitted

6.25.2.4 sendLen()

```
void sendLen (  
    Socket * client,  
    int len ) [extern]
```

send a 16-bit integer as an octad to Server

Parameters

<i>client</i>	the socket connection to the Server
<i>len</i>	the 16-bit integer to be encoded as octad and transmitted

6.25.2.5 getBytes()

```
int getBytes (  
    Socket * client,  
    char * b,  
    int expected ) [extern]
```

receive bytes over a socket sonnection

Parameters

<i>client</i>	the socket connection to the Server
<i>b</i>	the received bytes
<i>expected</i>	the number of bytes expected

Returns

-1 on failure, 0 on success

6.25.2.6 getInt16()

```
int getInt16 (  
    Socket * client ) [extern]
```

receive 16-bit integer from a socket

Parameters

<i>client</i>	the socket connection to the Server
---------------	-------------------------------------

Returns

a 16-bit integer

6.25.2.7 `getInt24()`

```
int getInt24 (  
    Socket * client ) [extern]
```

receive 24-bit integer from a socket

Parameters

<i>client</i>	the socket connection to the Server
---------------	-------------------------------------

Returns

a 24-bit integer

6.25.2.8 `getByte()`

```
int getByte (  
    Socket * client ) [extern]
```

receive a single byte from a socket

Parameters

<i>client</i>	the socket connection to the Server
---------------	-------------------------------------

Returns

a byte

6.25.2.9 `getOctad()`

```
int getOctad (  
    Socket * client,  
    octad * B,  
    int expected ) [extern]
```

receive an octad from a socket

Parameters

<i>client</i>	the socket connection to the Server
<i>B</i>	the output octad
<i>expected</i>	the number of bytes expected

Returns

-1 on failure, 0 on success

6.26 tls_sockets.h

[Go to the documentation of this file.](#)

```

00001
00008 // Set up and read/write sockets
00009
00010 #ifndef TLS_SOCKETS_H
00011 #define TLS_SOCKETS_H
00012
00013 #include <string.h>
00014 #include "tls_octads.h"
00015
00016 #ifdef TLS_ARDUINO
00017 #include "tls_wifi.h"
00018 #else
00019 #include <time.h>
00020 #include <unistd.h>
00021 #include <stdio.h>
00022 #include <sys/socket.h>
00023 #include <arpa/inet.h>
00024 #include <stdlib.h>
00025 #include <netinet/in.h>
00026 #include <netdb.h>
00027 #include <netinet/in.h>
00028 #include <sys/un.h>
00029 #endif
00030
00031 #ifndef TLS_ARDUINO
00032
00040 extern int setclientsock(int port, char *ip, int toms);
00041
00048 extern int getIPaddress(char *ip, char *hostname);
00049
00050 // Simple socket class, mimics Arduino
00053 class Socket
00054 {
00055     int sock;
00056     int toms;
00057     bool is_af_unix;
00059 private:
00060     Socket(bool is_af_unix) {
00061         this->sock = 0;
00062         this->toms = 5000;
00063         this->is_af_unix = is_af_unix;
00064     }
00065
00066     static int afunix_setclientsock(const char *const socket_path)
00067     {
00068         int sock;
00069         struct sockaddr_un serv_addr;
00070         if ((sock = socket(AF_UNIX, SOCK_STREAM, 0)) < 0)
00071             return -2;
00072
00073         serv_addr.sun_family = AF_UNIX;
00074         strcpy(serv_addr.sun_path, socket_path);
00075
00076         if (::connect(sock, (struct sockaddr *)&serv_addr, sizeof(serv_addr)) < 0)
00077             return -1;
00078
00079         return sock;
00080     }
00081
00082 public:
00083
00084     bool connect(char *host, int port) {

```

```

00085         if(!this->is_af_unix) {
00086             char ip[40];
00087             sock = 0;
00088             if (!getIPAddress(ip, host))
00089                 return false;
00090             sock = setclientsock(port, ip, toms);
00091             if (sock <= 0)
00092                 return false;
00093             return true;
00094         } else {
00095             bool connected = true;
00096             sock = afunix_setclientsock(host);
00097             if (sock <= 0) {
00098                 connected = false;
00099             }
00100             return connected;
00101         }
00102     }
00103
00104     int getport() {
00105         struct sockaddr_in local_address;
00106         socklen_t addr_size = sizeof(local_address);
00107         getsockname(sock, (struct sockaddr *)&local_address, &addr_size);
00108         return ntohs(local_address.sin_port);
00109     }
00110
00111     static Socket InetSocket() {
00112         return Socket(false);
00113     }
00114
00115     static Socket UnixSocket() {
00116         return Socket(true);
00117     }
00118
00119     void setTimeout(int to) {toms=to;}
00120     int write(char *buf,int len) {return ::send(sock,buf,len,0);}
00121     int read(char *buf,int len) {return ::recv(sock,buf,len,0);}
00122     void stop() {::close(sock);}
00123
00124     ~Socket() {::close(sock);}
00125 };
00126 #else
00127
00133 extern void clearsoc(Socket &client,octad *IO);
00134
00135 #endif
00136
00142 extern void sendOctad(Socket *client,octad *B);
00143
00149 extern void sendLen(Socket *client,int len);
00150
00158 extern int getBytes(Socket *client,char *b,int expected);
00159
00165 extern int getInt16(Socket *client);
00166
00172 extern int getInt24(Socket *client);
00173
00179 extern int getByte(Socket *client);
00180
00188 extern int getOctad(Socket *client,octad *B,int expected);
00189
00190 #endif

```

6.27 tls_tickets.h File Reference

TLS 1.3 process resumption tickets.

```

#include "tls1_3.h"
#include "tls_client_recv.h"

```

Functions

- int [parseTicket](#) (octad *TICK, [unsign32](#) birth, [ticket](#) *T)
parse a received ticket octad into a ticket structure

- void `initTicketContext` (`ticket *T`)
initialize a ticket structure
- void `endTicketContext` (`ticket *T`)
terminate a ticket structure
- bool `ticket_still_good` (`ticket *T`)
Check that a ticket is still good, and not out-of-date.

6.27.1 Detailed Description

TLS 1.3 process resumption tickets.

Author

Mike Scott

6.27.2 Function Documentation

6.27.2.1 `parseTicket()`

```
int parseTicket (
    octad * TICK,
    unsign32 birth,
    ticket * T ) [extern]
```

parse a received ticket octad into a ticket structure

Parameters

<i>TICK</i>	the input ticket octad
<i>T</i>	the output ticket structure
<i>birth</i>	the birth time of the ticket

Returns

bad ticket error, or 0 if ticket is good

6.27.2.2 `initTicketContext()`

```
void initTicketContext (
    ticket * T ) [extern]
```

initialize a ticket structure

Parameters

<i>T</i>	the ticket structure
----------	----------------------

6.27.2.3 endTicketContext()

```
void endTicketContext (
    ticket * T ) [extern]
```

terminate a ticket structure

Parameters

<i>T</i>	the ticket structure
----------	----------------------

6.27.2.4 ticket_still_good()

```
bool ticket_still_good (
    ticket * T ) [extern]
```

Check that a ticket is still good, and not out-of-date.

Parameters

<i>T</i>	the ticket structure
----------	----------------------

Returns

true if ticket is still good

6.28 tls_tickets.h

[Go to the documentation of this file.](#)

```
00001
00007 // Process Resumption Tickets
00008
00009 #ifndef TLS_TICKETS_H
00010 #define TLS_TICKETS_H
00011
00012 #include "tls1_3.h"
00013 #include "tls_client_rcv.h"
00014
00022 extern int parseTicket(octad *TICK,unsign32 birth,ticket *T);
00023
00028 extern void initTicketContext(ticket *T);
00029
00034 extern void endTicketContext(ticket *T);
00035
00041 extern bool ticket_still_good(ticket *T);
00042
00043 #endif
```

6.29 tls_wifi.h File Reference

define [Socket](#) structure depending on processor context

```
#include "tls1_3.h"
```

6.29.1 Detailed Description

define [Socket](#) structure depending on processor context

Author

Mike Scott

6.30 `tls_wifi.h`

[Go to the documentation of this file.](#)

```
00001
00007 // Set up WiFi environment for Arduino boards
00008
00009 #ifndef TLS_WIFI_H
00010 #define TLS_WIFI_H
00011
00012 #include "tls1_3.h"
00013
00014 #ifdef TLS_ARDUINO
00015
00016 #ifdef FISHINO_PIRANHA
00017 // Fishino Piranha board
00018 #define PARTICULAR_BOARD
00019 #include <Fishino.h>
00020 #include <SPI.h>
00021 typedef FishinoClient Socket;
00022 #define WiFi Fishino
00023 #define FISHINO
00024
00025 #endif
00026
00027 #ifdef ESP32
00028 // ESP32 board
00029 #define PARTICULAR_BOARD
00030 #include <WiFi.h>
00031 typedef WiFiClient Socket;
00032
00033 #endif
00034
00035 #ifndef PARTICULAR_BOARD
00036 // any other board
00037 #include <WiFi.h> // or maybe WifinINA?
00038 typedef WiFiClient Socket;
00039
00040 #endif
00041 #endif
00042
00043 #endif
```

6.31 `tls_x509.h` File Reference

X509 function Header File.

Data Structures

- struct [pktype](#)
Public key type.

Macros

- #define [X509_ECC](#) 1
- #define [X509_RSA](#) 2
- #define [X509_ECD](#) 3
- #define [X509_PQ](#) 4
- #define [X509_HY](#) 5
- #define [X509_H256](#) 2
- #define [X509_H384](#) 3
- #define [X509_H512](#) 4
- #define [X509_SHAKE256](#) 5
- #define [USE_NIST256](#) 0
- #define [USE_ED25519](#) 1
- #define [USE_ED448](#) 4
- #define [USE_NIST384](#) 10
- #define [USE_NIST521](#) 12

Functions

- void [ecdsa_sig_encode](#) ([octad](#) *c)
in-place ECDSA signature encoding
- int [ecdsa_sig_decode](#) ([octad](#) *c)
in-place ECDSA signature decoding
- [pktype](#) [X509_extract_private_key](#) ([octad](#) *c, [octad](#) *pk)
Extract private key.
- [pktype](#) [X509_extract_cert_sig](#) ([octad](#) *c, [octad](#) *s)
Extract certificate signature.
- int [X509_extract_cert](#) ([octad](#) *sc, [octad](#) *c)
- int [X509_find_public_key](#) ([octad](#) *c, int *ptr)
- [pktype](#) [X509_get_public_key](#) ([octad](#) *c, [octad](#) *key)
- [pktype](#) [X509_extract_public_key](#) ([octad](#) *c, [octad](#) *k)
- int [X509_find_issuer](#) ([octad](#) *c, int *len)
- int [X509_find_validity](#) ([octad](#) *c)
- int [X509_find_subject](#) ([octad](#) *c, int *len)
- int [X509_self_signed](#) ([octad](#) *c)
- int [X509_find_entity_property](#) ([octad](#) *c, [octad](#) *S, int s, int *f)
- int [X509_find_start_date](#) ([octad](#) *c, int s)
- int [X509_find_expiry_date](#) ([octad](#) *c, int s)
- int [X509_find_extensions](#) ([octad](#) *c)
- int [X509_find_extension](#) ([octad](#) *c, [octad](#) *S, int s, int *f)
- int [X509_find_alt_name](#) ([octad](#) *c, int s, char *name)

Variables

- [octad](#) [X509_CN](#)
- [octad](#) [X509_ON](#)
- [octad](#) [X509_EN](#)
- [octad](#) [X509_LN](#)
- [octad](#) [X509_UN](#)
- [octad](#) [X509_MN](#)
- [octad](#) [X509_SN](#)
- [octad](#) [X509_AN](#)
- [octad](#) [X509_KU](#)
- [octad](#) [X509_BC](#)

6.31.1 Detailed Description

X509 function Header File.

Author

Mike Scott

defines structures declares functions

6.31.2 Macro Definition Documentation

6.31.2.1 X509_ECC

```
#define X509_ECC 1
```

Elliptic Curve data type detected

6.31.2.2 X509_RSA

```
#define X509_RSA 2
```

RSA data type detected

6.31.2.3 X509_ECD

```
#define X509_ECD 3
```

Elliptic Curve (Ed25519) detected

6.31.2.4 X509_PQ

```
#define X509_PQ 4
```

Post Quantum method

6.31.2.5 X509_HY

```
#define X509_HY 5
```

Hybrid Post_Quantum

6.31.2.6 X509_H256

```
#define X509_H256 2
```

SHA256 hash algorithm used

6.31.2.7 X509_H384

```
#define X509_H384 3
```

SHA384 hash algorithm used

6.31.2.8 X509_H512

```
#define X509_H512 4
```

SHA512 hash algorithm used

6.31.2.9 X509_SHAKE256

```
#define X509_SHAKE256 5
```

SHA3 SHAKE256 algorithm used

6.31.2.10 USE_NIST256

```
#define USE_NIST256 0
```

For the NIST 256-bit standard curve - WEIERSTRASS only

6.31.2.11 USE_ED25519

```
#define USE_ED25519 1
```

Bernstein's Modulus $2^{255}-19$ - EDWARDS

6.31.2.12 USE_ED448

```
#define USE_ED448 4
```

Goldilocks Modulus Ed448 - EDWARDS

6.31.2.13 USE_NIST384

```
#define USE_NIST384 10
```

For the NIST 384-bit standard curve - WEIERSTRASS only

6.31.2.14 USE_NIST521

```
#define USE_NIST521 12
```

For the NIST 521-bit standard curve - WEIERSTRASS only

6.31.3 Function Documentation

6.31.3.1 ecdsa_sig_encode()

```
void ecdsa_sig_encode (  
    octad * c ) [extern]
```

in-place ECDSA signature encoding

Parameters

<i>c</i>	an ecdsa signature to be converted from r s form to ASN.1
----------	---

6.31.3.2 ecdsa_sig_decode()

```
int ecdsa_sig_decode (
    octad * c ) [extern]
```

in-place ECDSA signature decoding

Parameters

<i>c</i>	an ecdsa signature to be converted from ASN.1 to simple r s form
----------	--

Returns

index into c where conversion ended

6.31.3.3 X509_extract_private_key()

```
pktype X509_extract_private_key (
    octad * c,
    octad * pk ) [extern]
```

Extract private key.

Parameters

<i>c</i>	an X.509 private key
<i>pk</i>	the extracted private key - for RSA octad = p q dp dq c, for ECC octad = k

Returns

0 on failure, or indicator of private key type (ECC or RSA)

6.31.3.4 X509_extract_cert_sig()

```
pktype X509_extract_cert_sig (
    octad * c,
    octad * s ) [extern]
```

Extract certificate signature.

Parameters

<i>c</i>	an X.509 certificate
<i>s</i>	the extracted signature

Returns

0 on failure, or indicator of signature type (ECC or RSA)

6.31.3.5 X509_extract_cert()

```
int X509_extract_cert (
    octad * sc,
    octad * c ) [extern]
```

Parameters

<i>sc</i>	a signed certificate
<i>c</i>	the extracted certificate

Returns

0 on failure

6.31.3.6 X509_find_public_key()

```
int X509_find_public_key (
    octad * c,
    int * ptr ) [extern]
```

Parameters

<i>c</i>	an X.509 certificate
<i>ptr</i>	pointer to ASN.1 raw public key

Returns

length of raw public key

6.31.3.7 X509_get_public_key()

```
pktype X509_get_public_key (
    octad * c,
    octad * key ) [extern]
```

Parameters

<i>c</i>	an ASN.1 encoded public key
<i>key</i>	the extracted public key

Returns

indicator of public key type (ECC or RSA)

6.31.3.8 X509_extract_public_key()

```
pktype X509_extract_public_key (
    octad * c,
    octad * k ) [extern]
```

Parameters

<i>c</i>	an X.509 certificate
<i>k</i>	the extracted key

Returns

0 on failure, or indicator of public key type (ECC or RSA)

6.31.3.9 X509_find_issuer()

```
int X509_find_issuer (
    octad * c,
    int * len ) [extern]
```

Parameters

<i>c</i>	an X.509 certificate
<i>len</i>	length of issuer field

Returns

0 on failure, or pointer to issuer field in cert

6.31.3.10 X509_find_validity()

```
int X509_find_validity (
    octad * c ) [extern]
```

Parameters

<i>c</i>	an X.509 certificate
----------	----------------------

Returns

0 on failure, or pointer to validity field in cert

6.31.3.11 X509_find_subject()

```
int X509_find_subject (
    octad * c,
    int * len ) [extern]
```

Parameters

<i>c</i>	an X.509 certificate
<i>len</i>	length of subject field

Returns

0 on failure, or pointer to subject field in cert

6.31.3.12 X509_self_signed()

```
int X509_self_signed (
    octad * c ) [extern]
```

Parameters

<i>c</i>	an X.509 certificate
----------	----------------------

Returns

true if self-signed, else false

6.31.3.13 X509_find_entity_property()

```
int X509_find_entity_property (
    octad * c,
    octad * S,
    int s,
    int * f ) [extern]
```

Parameters

<i>c</i>	an X.509 certificate
<i>S</i>	is OID of property we are looking for
<i>s</i>	is a pointer to the section of interest in the cert
<i>f</i>	is pointer to the length of the property

Returns

0 on failure, or pointer to the property

6.31.3.14 X509_find_start_date()

```
int X509_find_start_date (
    octad * c,
    int s ) [extern]
```

Parameters

<i>c</i>	an X.509 certificate
<i>s</i>	is a pointer to the start of the validity field

Returns

0 on failure, or pointer to the start date

6.31.3.15 X509_find_expiry_date()

```
int X509_find_expiry_date (
    octad * c,
    int s ) [extern]
```

Parameters

<i>c</i>	an X.509 certificate
<i>s</i>	is a pointer to the start of the validity field

Returns

0 on failure, or pointer to the expiry date

6.31.3.16 X509_find_extensions()

```
int X509_find_extensions (
    octad * c ) [extern]
```

Parameters

<i>c</i>	an X.509 certificate
----------	----------------------

Returns

0 on failure (or no extensions), or pointer to extensions field in cert

6.31.3.17 X509_find_extension()

```
int X509_find_extension (
    octad * c,
```

```
    octad * S,  
    int s,  
    int * f ) [extern]
```

Parameters

<i>c</i>	an X.509 certificate
<i>S</i>	is OID of particular extension we are looking for
<i>s</i>	is a pointer to the section of interest in the cert
<i>f</i>	is pointer to the length of the extension

Returns

0 on failure, or pointer to the extension

6.31.3.18 X509_find_alt_name()

```
int X509_find_alt_name (  
    octad * c,  
    int s,  
    char * name ) [extern]
```

Parameters

<i>c</i>	an X.509 certificate
<i>s</i>	is a pointer to certificate extension SubjectAltNames
<i>name</i>	is a URL

Returns

0 on failure, 1 if URL is in list of alt names

6.31.4 Variable Documentation**6.31.4.1 X509_CN**

```
octad X509_CN [extern]
```

Country Name

6.31.4.2 X509_ON

```
octad X509_ON [extern]
```

organisation Name

6.31.4.3 X509_EN

`octad X509_EN [extern]`

email

6.31.4.4 X509_LN

`octad X509_LN [extern]`

local name

6.31.4.5 X509_UN

`octad X509_UN [extern]`

Unit name (aka Organisation Unit OU)

6.31.4.6 X509_MN

`octad X509_MN [extern]`

My Name (aka Common Name)

6.31.4.7 X509_SN

`octad X509_SN [extern]`

State Name

6.31.4.8 X509_AN

`octad X509_AN [extern]`

Alternate Name

6.31.4.9 X509_KU

`octad X509_KU [extern]`

Key Usage

6.31.4.10 X509_BC

`octad X509_BC [extern]`

Basic Constraints

6.32 tls_x509.h

[Go to the documentation of this file.](#)

```

00001
00011 #ifndef TLS_X509_H
00012 #define TLS_X509_H
00013
00014 // Supported Encryption Methods
00015
00016 #define X509_ECC 1
00017 #define X509_RSA 2
00018 #define X509_ECD 3
00019 #define X509_PQ 4
00020 #define X509_HY 5
00022 // Supported Hash functions
00023
00024 #define X509_H256 2
00025 #define X509_H384 3
00026 #define X509_H512 4
00027 #define X509_SHAKE256 5
00028 // Supported Curves
00029
00030 #define USE_NIST256 0
00031 #define USE_ED25519 1
00032 #define USE_ED448 4
00033 // #define USE_BRAINPOOL 2 /**< For Brainpool 256-bit curve - WEIERSTRASS only */
00034 // #define USE_ANSSI 3 /**< For French 256-bit standard curve - WEIERSTRASS only */
00035 #define USE_NIST384 10
00036 #define USE_NIST521 12
00038 extern octad X509_CN;
00039 extern octad X509_ON;
00040 extern octad X509_EN;
00041 extern octad X509_LN;
00042 extern octad X509_UN;
00043 extern octad X509_MN;
00044 extern octad X509_SN;
00046 extern octad X509_AN;
00047 extern octad X509_KU;
00048 extern octad X509_BC;
00053 typedef struct
00054 {
00055     int type;
00056     int hash;
00057     int curve;
00058 } pktype;
00059
00060
00065 extern void ecdsa_sig_encode(octad *c);
00066
00072 extern int ecdsa_sig_decode(octad *c);
00073
00074 /* X.509 functions */
00075
00082 extern pktype X509_extract_private_key(octad *c, octad *pk);
00083
00091 extern pktype X509_extract_cert_sig(octad *c, octad *s);
00098 extern int X509_extract_cert(octad *sc, octad *c);
00099
00106 extern int X509_find_public_key(octad *c, int *ptr);
00107
00114 extern pktype X509_get_public_key(octad *c, octad *key);
00115
00116
00123 extern pktype X509_extract_public_key(octad *c, octad *k);
00130 extern int X509_find_issuer(octad *c, int *len);
00136 extern int X509_find_validity(octad *c);
00143 extern int X509_find_subject(octad *c, int *len);
00144
00150 extern int X509_self_signed(octad *c);
00151
00160 extern int X509_find_entity_property(octad *c, octad *S, int s, int *f);
00167 extern int X509_find_start_date(octad *c, int s);
00174 extern int X509_find_expiry_date(octad *c, int s);
00175
00181 extern int X509_find_extensions(octad *c);
00190 extern int X509_find_extension(octad *c, octad *S, int s, int *f);
00191
00199 extern int X509_find_alt_name(octad *c, int s, char *name);
00200
00201 #endif

```

6.33 ECCX08.h

```

00001 /*
00002  This file is part of the ArduinoECCX08 library.
00003  Copyright (c) 2018 Arduino SA. All rights reserved.
00004
00005  This library is free software; you can redistribute it and/or
00006  modify it under the terms of the GNU Lesser General Public
00007  License as published by the Free Software Foundation; either
00008  version 2.1 of the License, or (at your option) any later version.
00009
00010  This library is distributed in the hope that it will be useful,
00011  but WITHOUT ANY WARRANTY; without even the implied warranty of
00012  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU
00013  Lesser General Public License for more details.
00014
00015  You should have received a copy of the GNU Lesser General Public
00016  License along with this library; if not, write to the Free Software
00017  Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA
00018 */
00019
00020 #ifndef _ECCX08_H_
00021 #define _ECCX08_H_
00022
00023 #include <Arduino.h>
00024 #include <Wire.h>
00025
00026 class ECCX08Class
00027 {
00028 public:
00029   ECCX08Class(TwoWire& wire, uint8_t address);
00030   virtual ~ECCX08Class();
00031
00032   int begin();
00033   void end();
00034
00035   int serialNumber(byte sn[]);
00036   String serialNumber();
00037
00038   long random(long max);
00039   long random(long min, long max);
00040   int random(byte data[], size_t length);
00041
00042   int generatePrivateKey(int slot, byte publicKey[]);
00043   int generatePublicKey(int slot, byte publicKey[]);
00044   int generateSharedKey(int slot, byte publicKey[], byte sharedKey[]); // M.Scott 12/7/2021
00045
00046   int ecdsaVerify(const byte message[], const byte signature[], const byte pubkey[]);
00047   int ecSign(int slot, const byte message[], byte signature[]);
00048
00049   int challenge(const byte message[]);
00050   int aesEncrypt(byte block[]);
00051   int aesGFM(byte state[], byte H[]);
00052
00053   int beginSHA256();
00054   int beginHMAC(int slot);
00055   int updateSHA256(const byte data[], int len); // variable
00056   int endSHA256(byte result[]);
00057   int endSHA256(const byte data[], int length, byte result[]);
00058   int readSHA256(byte context[]);
00059   int writeSHA256(byte context[], int length);
00060
00061   int readSlot(int slot, byte data[], int length);
00062   int writeSlot(int slot, const byte data[], int length);
00063
00064   int locked();
00065   int writeConfiguration(const byte data[]);
00066   int readConfiguration(byte data[]);
00067   int lock();
00068
00069 private:
00070   int wakeup();
00071   int sleep();
00072   int idle();
00073
00074   long version();
00075   int verify(const byte signature[], const byte pubkey[]);
00076   int sign(int slot, byte signature[]);
00077
00078   int read(int zone, int address, byte buffer[], int length);
00079   int write(int zone, int address, const byte buffer[], int length);
00080   int lock(int zone);
00081
00082   int addressForSlotOffset(int slot, int offset);
00083
00084   int sendCommand(uint8_t opcode, uint8_t param1, uint16_t param2, const byte data[] = NULL, size_t
dataLength = 0);

```

```
00085     int receiveResponse(void* response, size_t length);
00086     int receiveResponse(void* response);
00087     uint16_t crc16(const byte data[], size_t length);
00088
00089 private:
00090     TwoWire* _wire;
00091     uint8_t _address;
00092
00093     static const uint32_t _wakeupFrequency;
00094     static const uint32_t _normalFrequency;
00095 };
00096
00097 extern ECCX08Class ECCX08;
00098
00099 #endif
```


Index

active
 crypto, [15](#)
addALPNExt
 tls_client_send.h, [85](#)
addCertAuthorities
 tls_client_send.h, [82](#)
addClientRawPublicKey
 tls_client_send.h, [84](#)
addCookieExt
 tls_client_send.h, [87](#)
addEarlyDataExt
 tls_client_send.h, [87](#)
addHeartbeat
 tls_client_send.h, [82](#)
addKeyShareExt
 tls_client_send.h, [85](#)
addMFLExt
 tls_client_send.h, [85](#)
addPadding
 tls_client_send.h, [86](#)
addPostHSAuth
 tls_client_send.h, [87](#)
addPreSharedKeyExt
 tls_client_send.h, [83](#)
addPSKModesExt
 tls_client_send.h, [86](#)
addRSLExt
 tls_client_send.h, [86](#)
addServerNameExt
 tls_client_send.h, [83](#)
addServerRawPublicKey
 tls_client_send.h, [84](#)
addSigAlgsCertExt
 tls_client_send.h, [85](#)
addSigAlgsExt
 tls_client_send.h, [84](#)
addSupportedGroupsExt
 tls_client_send.h, [83](#)
addVersionExt
 tls_client_send.h, [86](#)
age_obfuscator
 ticket, [23](#)
ALERT
 tls1_3.h, [52](#)
alert_from_cause
 tls_client_send.h, [92](#)
ALLOW_SELF_SIGNED
 tls1_3.h, [39](#)
allowed_to_heartbeat
 TLS_session, [28](#)
alpn
 ee_status, [18](#)
APP_PROTOCOL
 tls1_3.h, [51](#)
APPLICATION
 tls1_3.h, [51](#)
AUTHENTICATION_FAILURE
 tls1_3.h, [55](#)

BAD_CERT_CHAIN
 tls1_3.h, [54](#)
BAD_CERTIFICATE
 tls1_3.h, [57](#)
BAD_HANDSHAKE
 tls1_3.h, [56](#)
BAD_HELLO
 tls1_3.h, [54](#)
BAD_MESSAGE
 tls1_3.h, [56](#)
BAD_RECORD
 tls1_3.h, [55](#)
BAD_RECORD_MAC
 tls1_3.h, [58](#)
BAD_REQUEST_UPDATE
 tls1_3.h, [57](#)
BAD_TICKET
 tls1_3.h, [55](#)
badResponse
 tls_client_recv.h, [76](#)
BFIBE_CCA_DECRYPT
 tls_bfibe.h, [68](#)
BFIBE_CCA_ENCRYPT
 tls_bfibe.h, [67](#)
birth
 ticket, [23](#)
byte
 tls1_3.h, [61](#)

CA_NOT_FOUND
 tls1_3.h, [55](#)
cacerts
 tls_certs.h, [71](#)
CERT_AUTHORITIES
 tls1_3.h, [51](#)
CERT_OUTOFDATE
 tls1_3.h, [55](#)
CERT_REQUEST
 tls1_3.h, [53](#)
CERT_VERIFY

- tls1_3.h, 53
- CERT_VERIFY_FAIL
 - tls1_3.h, 56
- CERTCHAIN
 - credential, 14
- certchain
 - credential, 13
- CERTIFICATE
 - tls1_3.h, 52
- CERTIFICATE_EXPIRED
 - tls1_3.h, 58
- CHANGE_CIPHER
 - tls1_3.h, 52
- CHECK_NAME_IN_CERT
 - tls1_3.h, 39
- checkServerCertChain
 - tls_cert_chain.h, 69
- checkServerCertVerifier
 - tls_keys_calc.h, 101
- checkVerifierData
 - tls_keys_calc.h, 100
- cipher_suite
 - ticket, 23
 - TLS_session, 25
- cipherSuites
 - tls_client_send.h, 88
- CLIENT_CERT
 - tls1_3.h, 38
- CLIENT_CERT_KIND
 - tls1_3.h, 38
- CLIENT_CERT_TYPE
 - tls1_3.h, 51
- client_cert_type
 - TLS_session, 26
- CLIENT_HELLO
 - tls1_3.h, 52
- clientRandom
 - tls_client_send.h, 88
- CLOSE_NOTIFY
 - tls1_3.h, 58
- CLOSURE_ALERT_RECEIVED
 - tls1_3.h, 57
- Configure the Arduino Nano RP2040, 7
- COOKIE
 - tls1_3.h, 50
- createClientCertVerifier
 - tls_keys_calc.h, 101
- createCryptoContext
 - tls_keys_calc.h, 97
- createRecvCryptoContext
 - tls_keys_calc.h, 98
- createSendCryptoContext
 - tls_keys_calc.h, 98
- credential, 13
 - CERTCHAIN, 14
 - certchain, 13
 - nreqs, 14
 - nreqsraw, 14
 - PUBLICKEY, 14
 - publickey, 13
 - requirements, 14
 - SECRETKEY, 14
 - secretkey, 14
 - sigalg, 14
- crypto, 15
 - active, 15
 - IV, 16
 - iv, 15
 - K, 16
 - k, 15
 - record, 16
 - suite, 16
 - taglen, 16
- CRYPTO_SETTING
 - tls1_3.h, 38
- CTS
 - TLS_session, 27
- cts
 - TLS_session, 27
- CTX
 - TLS_session, 27
- ctx
 - TLS_session, 27
- curve
 - pktype, 20
- DECODE_ERROR
 - tls1_3.h, 58
- DECRYPT_ERROR
 - tls1_3.h, 57
- deriveApplicationSecrets
 - tls_keys_calc.h, 100
- deriveEarlySecrets
 - tls_keys_calc.h, 98
- deriveHandshakeSecrets
 - tls_keys_calc.h, 99
- deriveLaterSecrets
 - tls_keys_calc.h, 99
- deriveUpdatedKeys
 - tls_keys_calc.h, 100
- deriveVerifierData
 - tls_keys_calc.h, 101
- DLT_SS
 - tls1_3.h, 37
- EARLY_DATA
 - tls1_3.h, 50
- early_data
 - ee_status, 18
- ECC_SS
 - tls1_3.h, 37
- ECCX08.h, 152
- ECCX08Class, 17
- ECDSA_SECP256R1_SHA256
 - tls1_3.h, 46
- ECDSA_SECP256R1_SHA384
 - tls1_3.h, 46

ECDSA_SECP384R1_SHA384
 tls1_3.h, 46
ecdsa_sig_decode
 tls_x509.h, 144
ecdsa_sig_encode
 tls_x509.h, 143
ED25519
 tls1_3.h, 47
ED448
 tls1_3.h, 47
EDD_SS
 tls1_3.h, 38
EDDSA
 tls1_3.h, 36
ee_status, 17
 alpn, 18
 early_data, 18
 max_frag_length, 18
 server_name, 18
EMPTY_CERT_CHAIN
 tls1_3.h, 56
ENCRYPTED_EXTENSIONS
 tls1_3.h, 53
END_OF_EARLY_DATA
 tls1_3.h, 53
endTicketContext
 tls_tickets.h, 138
err
 ret, 20
ERROR_ALERT_RECEIVED
 tls1_3.h, 56
expect_heartbeats
 TLS_session, 28

favourite_group
 ticket, 23
 TLS_session, 25
FINISHED
 tls1_3.h, 53
FORBIDDEN_EXTENSION
 tls1_3.h, 55
FROM_FILE
 tls1_3.h, 37
FROM_ROM
 tls1_3.h, 37

getByte
 tls_sockets.h, 135
getBytes
 tls_sockets.h, 134
getCertificateRequest
 tls_client_rcv.h, 79
getCheckServerCertificateChain
 tls_client_rcv.h, 79
getInt16
 tls_sockets.h, 134
getInt24
 tls_sockets.h, 135
getIPAddress
 tls_sockets.h, 133
getOctad
 tls_sockets.h, 135
getServerCertVerify
 tls_client_rcv.h, 77
getServerEncryptedExtensions
 tls_client_rcv.h, 77
getServerFinished
 tls_client_rcv.h, 78
getServerHello
 tls_client_rcv.h, 78
getServerRecord
 tls_client_rcv.h, 74

HANDSHAKE_FAILURE
 tls1_3.h, 58
HANDSHAKE_RETRY
 tls1_3.h, 54
hash
 pktype, 20
HEART_BEAT
 tls1_3.h, 52
HEARTBEAT
 tls1_3.h, 51
heartbeat_req_in_flight
 TLS_session, 28
hostname
 TLS_session, 25
HS
 TLS_session, 26
hs
 TLS_session, 26
HSHAKE
 tls1_3.h, 51
htype
 unihash, 29
HW_1
 tls1_3.h, 37
HW_2
 tls1_3.h, 38
HYB_SS
 tls1_3.h, 37
HYBRID
 tls1_3.h, 36
HYBRID_KX
 tls1_3.h, 46

IBUFF
 TLS_session, 27
ibuff
 TLS_session, 28
id
 TLS_session, 25
ID_MISMATCH
 tls1_3.h, 54
ILLEGAL_PARAMETER
 tls1_3.h, 57
incrementCryptoContext
 tls_keys_calc.h, 97

- initCryptoContext
 - tls_keys_calc.h, 97
- initTicketContext
 - tls_tickets.h, 138
- initTranscriptHash
 - tls_keys_calc.h, 95
- IO_APPLICATION
 - tls1_3.h, 35
- IO_DEBUG
 - tls1_3.h, 36
- IO_NONE
 - tls1_3.h, 35
- IO_PROTOCOL
 - tls1_3.h, 36
- IO_WIRE
 - tls1_3.h, 36
- IV
 - crypto, 16
- iv
 - crypto, 15
- K
 - crypto, 16
- k
 - crypto, 15
- K_recv
 - TLS_session, 26
- K_send
 - TLS_session, 26
- KEY_SHARE
 - tls1_3.h, 50
- KEY_UPDATE
 - tls1_3.h, 53
- len
 - octad, 19
- lifetime
 - ticket, 23
- log
 - tls_logger.h, 105
- LOG_OUTPUT_TRUNCATION
 - tls1_3.h, 59
- logAlert
 - tls_logger.h, 107
- logCert
 - tls_logger.h, 106
- logCertDetails
 - tls_logger.h, 106
- logCipherSuite
 - tls_logger.h, 107
- logEncExt
 - tls_logger.h, 106
- logKeyExchange
 - tls_logger.h, 107
- logServerHello
 - tls_logger.h, 105
- logServerResponse
 - tls_logger.h, 107
- logSigAlg
 - tls_logger.h, 107
- logTicket
 - tls_logger.h, 105
- max
 - octad, 19
- max_early_data
 - ticket, 23
- MAX_EXCEEDED
 - tls1_3.h, 56
- MAX_FRAG_LENGTH
 - tls1_3.h, 50
- max_frag_length
 - ee_status, 18
- max_record
 - TLS_session, 25
- MEM_OVERFLOW
 - tls1_3.h, 55
- MERGE_MESSAGES
 - tls1_3.h, 39
- MESSAGE_HASH
 - tls1_3.h, 53
- millis
 - tls_octads.h, 110
- MISSING_EXTENSION
 - tls1_3.h, 59
- MISSING_EXTENSIONS
 - tls1_3.h, 57
- MISSING_REQUEST_CONTEXT
 - tls1_3.h, 54
- MLDSA44
 - tls1_3.h, 47
- MLDSA65
 - tls1_3.h, 48
- MLDSA87
 - tls1_3.h, 48
- MLKEM1024
 - tls1_3.h, 46
- MLKEM512
 - tls1_3.h, 46
- MLKEM768
 - tls1_3.h, 46
- mycert
 - tls_certs.h, 71
- myprintf
 - tls_logger.h, 104
- myprivate
 - tls_certs.h, 71
- NO_CERT
 - tls1_3.h, 37
- NONCE
 - ticket, 23
- nonce
 - ticket, 22
- NOT_EXPECTED
 - tls1_3.h, 55
- NOT_TLS1_3
 - tls1_3.h, 54

- nreqs
 - credential, [14](#)
- nreqsraw
 - credential, [14](#)
- OBUFF
 - TLS_session, [27](#)
- obuf
 - TLS_session, [28](#)
- OCT_append_byte
 - tls_octads.h, [112](#)
- OCT_append_bytes
 - tls_octads.h, [112](#)
- OCT_append_int
 - tls_octads.h, [110](#)
- OCT_append_octad
 - tls_octads.h, [110](#)
- OCT_append_string
 - tls_octads.h, [112](#)
- OCT_compare
 - tls_octads.h, [110](#)
- OCT_copy
 - tls_octads.h, [113](#)
- OCT_from_base64
 - tls_octads.h, [112](#)
- OCT_from_hex
 - tls_octads.h, [111](#)
- OCT_kill
 - tls_octads.h, [111](#)
- OCT_output_base64
 - tls_octads.h, [114](#)
- OCT_output_hex
 - tls_octads.h, [113](#)
- OCT_output_string
 - tls_octads.h, [114](#)
- OCT_reverse
 - tls_octads.h, [113](#)
- OCT_shift_left
 - tls_octads.h, [111](#)
- OCT_truncate
 - tls_octads.h, [113](#)
- octad, [18](#)
 - len, [19](#)
 - max, [19](#)
 - val, [19](#)
- origin
 - ticket, [24](#)
- PADDING
 - tls1_3.h, [50](#)
- parsebytes
 - tls_client_rcv.h, [73](#)
- parsebytesorPull
 - tls_client_rcv.h, [75](#)
- parseInt
 - tls_client_rcv.h, [74](#)
- parseIntorPull
 - tls_client_rcv.h, [75](#)
- parseoctad
 - tls_client_rcv.h, [73](#)
- parseoctadorPull
 - tls_client_rcv.h, [75](#)
- parseoctadorPullptrX
 - tls_client_rcv.h, [76](#)
- parseoctadptr
 - tls_client_rcv.h, [74](#)
- parseTicket
 - tls_tickets.h, [138](#)
- pktype, [19](#)
 - curve, [20](#)
 - hash, [20](#)
 - type, [20](#)
- POST_HANDSHAKE_AUTH
 - tls1_3.h, [49](#)
- POST_HS_AUTH
 - tls1_3.h, [38](#)
- POST_QUANTUM
 - tls1_3.h, [36](#)
- PQIBE_CCA_DECRYPT
 - tls_pqibe.h, [116](#)
- PQIBE_CCA_ENCRYPT
 - tls_pqibe.h, [116](#)
- PRESHARED_KEY
 - tls1_3.h, [50](#)
- PROTOCOL_VERSION
 - tls1_3.h, [58](#)
- PSK
 - ticket, [23](#)
- psk
 - ticket, [22](#)
- PSK_IBE
 - tls1_3.h, [60](#)
- PSK_KEY
 - tls1_3.h, [60](#)
- PSK_MODE
 - tls1_3.h, [50](#)
- PSK_NOT
 - tls1_3.h, [60](#)
- PSKOK
 - tls1_3.h, [48](#)
- PSKWECDHE
 - tls1_3.h, [48](#)
- ptr
 - TLS_session, [28](#)
- PUBLICKEY
 - credential, [14](#)
- publickey
 - credential, [13](#)
- RAW_PUBLIC_KEY
 - tls1_3.h, [60](#)
- record
 - crypto, [16](#)
- RECORD_OVERFLOW
 - tls1_3.h, [58](#)
- RECORD_SIZE_LIMIT
 - tls1_3.h, [51](#)
- recoverPSK

- tls_keys_calc.h, 98
- requirements
 - credential, 14
- ret, 20
 - err, 20
 - val, 20
- rewindIO
 - tls_keys_calc.h, 95
- RMS
 - TLS_session, 26
- rms
 - TLS_session, 26
- RSA_PKCS1_SHA256
 - tls1_3.h, 47
- RSA_PKCS1_SHA384
 - tls1_3.h, 47
- RSA_PKCS1_SHA512
 - tls1_3.h, 47
- RSA_PSS_RSAE_SHA256
 - tls1_3.h, 46
- RSA_PSS_RSAE_SHA384
 - tls1_3.h, 47
- RSA_PSS_RSAE_SHA512
 - tls1_3.h, 47
- RSA_SS
 - tls1_3.h, 37
- runningHash
 - tls_keys_calc.h, 95
- runningHashIO
 - tls_keys_calc.h, 95
- runningHashIOrewind
 - tls_keys_calc.h, 96
- runningSyntheticHash
 - tls_keys_calc.h, 96
- SAL_aeadDecrypt
 - tls_sal.h, 129
- SAL_aeadEncrypt
 - tls_sal.h, 128
- SAL_aeadKeylen
 - tls_sal.h, 125
- SAL_aeadTaglen
 - tls_sal.h, 125
- SAL_ciphers
 - tls_sal.h, 123
- SAL_generateKeyPair
 - tls_sal.h, 129
- SAL_generateSharedSecret
 - tls_sal.h, 130
- SAL_groups
 - tls_sal.h, 123
- SAL_hashInit
 - tls_sal.h, 127
- SAL_hashLen
 - tls_sal.h, 125
- SAL_hashNull
 - tls_sal.h, 127
- SAL_hashOutput
 - tls_sal.h, 128
- SAL_hashProcessArray
 - tls_sal.h, 128
- SAL_hashType
 - tls_sal.h, 124
- SAL_hkdfExpand
 - tls_sal.h, 127
- SAL_hkdfExtract
 - tls_sal.h, 126
- SAL_hmac
 - tls_sal.h, 127
- SAL_initLib
 - tls_sal.h, 124
- SAL_name
 - tls_sal.h, 123
- SAL_randomByte
 - tls_sal.h, 126
- SAL_randomOctad
 - tls_sal.h, 126
- SAL_sigCerts
 - tls_sal.h, 124
- SAL_sigs
 - tls_sal.h, 124
- SAL_tlsSignature
 - tls_sal.h, 130
- SAL_tlsSignatureVerify
 - tls_sal.h, 130
- SECP256R1
 - tls1_3.h, 45
- SECP384R1
 - tls1_3.h, 45
- SECP521R1
 - tls1_3.h, 45
- SECRETKEY
 - credential, 14
- secretkey
 - credential, 14
- seeWhatsNext
 - tls_client_rcv.h, 77
- SELF_SIGNED_CERT
 - tls1_3.h, 56
- sendAlert
 - tls_client_send.h, 90
- sendBinder
 - tls_client_send.h, 89
- sendCCCS
 - tls_client_send.h, 82
- sendClientCertificateChain
 - tls_client_send.h, 91
- sendClientCertVerify
 - tls_client_send.h, 91
- sendClientFinish
 - tls_client_send.h, 91
- sendClientHello
 - tls_client_send.h, 90
- sendClientMessage
 - tls_client_send.h, 89
- sendEndOfEarlyData
 - tls_client_send.h, 92

- sendHeartbeatRequest
 - tls_client_send.h, [89](#)
- sendKeyUpdate
 - tls_client_send.h, [90](#)
- sendLen
 - tls_sockets.h, [134](#)
- sendOctad
 - tls_sockets.h, [133](#)
- sendRecord
 - tls_client_send.h, [88](#)
- sendZeroRecord
 - tls_client_send.h, [89](#)
- SERVER_CERT_TYPE
 - tls1_3.h, [51](#)
- server_cert_type
 - TLS_session, [26](#)
- SERVER_HELLO
 - tls1_3.h, [52](#)
- SERVER_NAME
 - tls1_3.h, [49](#)
- server_name
 - ee_status, [18](#)
- setclientsock
 - tls_sockets.h, [133](#)
- setCredential
 - tls_certs.h, [70](#)
- SIG_ALGS
 - tls1_3.h, [49](#)
- SIG_ALGS_CERT
 - tls1_3.h, [49](#)
- sigalg
 - credential, [14](#)
- sign16
 - tls1_3.h, [61](#)
- sign32
 - tls1_3.h, [61](#)
- sign64
 - tls1_3.h, [61](#)
- sign8
 - tls1_3.h, [61](#)
- Socket, [21](#)
- sockptr
 - TLS_session, [25](#)
- state
 - unihash, [29](#)
- status
 - TLS_session, [25](#)
- STS
 - TLS_session, [27](#)
- sts
 - TLS_session, [27](#)
- suite
 - crypto, [16](#)
- SUPPORTED_GROUPS
 - tls1_3.h, [49](#)
- T
 - TLS_session, [28](#)
- taglen
 - crypto, [16](#)
- TICK
 - ticket, [22](#)
- tick
 - ticket, [22](#)
- TICKET
 - tls1_3.h, [53](#)
- ticket, [21](#)
 - age_obfuscator, [23](#)
 - birth, [23](#)
 - cipher_suite, [23](#)
 - favourite_group, [23](#)
 - lifetime, [23](#)
 - max_early_data, [23](#)
 - NONCE, [23](#)
 - nonce, [22](#)
 - origin, [24](#)
 - PSK, [23](#)
 - psk, [22](#)
 - TICK, [22](#)
 - tick, [22](#)
 - valid, [22](#)
- ticket_still_good
 - tls_tickets.h, [139](#)
- TIIGER TLS C++, [1](#)
- TIME_OUT
 - tls1_3.h, [56](#)
- TIMED_OUT
 - tls1_3.h, [52](#)
- TINY_ECC
 - tls1_3.h, [36](#)
- TLS13_clean
 - tls_protocol.h, [121](#)
- TLS13_connect
 - tls_protocol.h, [119](#)
- TLS13_CONNECTED
 - tls1_3.h, [59](#)
- TLS13_DISCONNECTED
 - tls1_3.h, [59](#)
- TLS13_end
 - tls_protocol.h, [119](#)
- TLS13_HANDSHAKING
 - tls1_3.h, [59](#)
- TLS13_recv
 - tls_protocol.h, [120](#)
- TLS13_recv_and_check
 - tls_protocol.h, [120](#)
- TLS13_send
 - tls_protocol.h, [120](#)
- TLS13_start
 - tls_protocol.h, [118](#)
- TLS13_stop
 - tls_protocol.h, [119](#)
- TLS13_UPDATE_NOT_REQUESTED
 - tls1_3.h, [49](#)
- TLS13_UPDATE_REQUESTED
 - tls1_3.h, [49](#)
- TLS1_0

- tls1_3.h, 48
- TLS1_2
 - tls1_3.h, 48
- TLS1_3
 - tls1_3.h, 49
- tls1_3.h, 31, 62
 - ALERT, 52
 - ALLOW_SELF_SIGNED, 39
 - APP_PROTOCOL, 51
 - APPLICATION, 51
 - AUTHENTICATION_FAILURE, 55
 - BAD_CERT_CHAIN, 54
 - BAD_CERTIFICATE, 57
 - BAD_HANDSHAKE, 56
 - BAD_HELLO, 54
 - BAD_MESSAGE, 56
 - BAD_RECORD, 55
 - BAD_RECORD_MAC, 58
 - BAD_REQUEST_UPDATE, 57
 - BAD_TICKET, 55
 - byte, 61
 - CA_NOT_FOUND, 55
 - CERT_AUTHORITIES, 51
 - CERT_OUTOFDATE, 55
 - CERT_REQUEST, 53
 - CERT_VERIFY, 53
 - CERT_VERIFY_FAIL, 56
 - CERTIFICATE, 52
 - CERTIFICATE_EXPIRED, 58
 - CHANGE_CIPHER, 52
 - CHECK_NAME_IN_CERT, 39
 - CLIENT_CERT, 38
 - CLIENT_CERT_KIND, 38
 - CLIENT_CERT_TYPE, 51
 - CLIENT_HELLO, 52
 - CLOSE_NOTIFY, 58
 - CLOSURE_ALERT_RECEIVED, 57
 - COOKIE, 50
 - CRYPTO_SETTING, 38
 - DECODE_ERROR, 58
 - DECRYPT_ERROR, 57
 - DLT_SS, 37
 - EARLY_DATA, 50
 - ECC_SS, 37
 - ECDSA_SECP256R1_SHA256, 46
 - ECDSA_SECP256R1_SHA384, 46
 - ECDSA_SECP384R1_SHA384, 46
 - ED25519, 47
 - ED448, 47
 - EDD_SS, 38
 - EDDSA, 36
 - EMPTY_CERT_CHAIN, 56
 - ENCRYPTED_EXTENSIONS, 53
 - END_OF_EARLY_DATA, 53
 - ERROR_ALERT_RECEIVED, 56
 - FINISHED, 53
 - FORBIDDEN_EXTENSION, 55
 - FROM_FILE, 37
 - FROM_ROM, 37
 - HANDSHAKE_FAILURE, 58
 - HANDSHAKE_RETRY, 54
 - HEART_BEAT, 52
 - HEARTBEAT, 51
 - HSHAKE, 51
 - HW_1, 37
 - HW_2, 38
 - HYB_SS, 37
 - HYBRID, 36
 - HYBRID_KX, 46
 - ID_MISMATCH, 54
 - ILLEGAL_PARAMETER, 57
 - IO_APPLICATION, 35
 - IO_DEBUG, 36
 - IO_NONE, 35
 - IO_PROTOCOL, 36
 - IO_WIRE, 36
 - KEY_SHARE, 50
 - KEY_UPDATE, 53
 - LOG_OUTPUT_TRUNCATION, 59
 - MAX_EXCEEDED, 56
 - MAX_FRAG_LENGTH, 50
 - MEM_OVERFLOW, 55
 - MERGE_MESSAGES, 39
 - MESSAGE_HASH, 53
 - MISSING_EXTENSION, 59
 - MISSING_EXTENSIONS, 57
 - MISSING_REQUEST_CONTEXT, 54
 - MLDSA44, 47
 - MLDSA65, 48
 - MLDSA87, 48
 - MLKEM1024, 46
 - MLKEM512, 46
 - MLKEM768, 46
 - NO_CERT, 37
 - NOT_EXPECTED, 55
 - NOT_TLS1_3, 54
 - PADDING, 50
 - POST_HANDSHAKE_AUTH, 49
 - POST_HS_AUTH, 38
 - POST_QUANTUM, 36
 - PRESHARED_KEY, 50
 - PROTOCOL_VERSION, 58
 - PSK_IBE, 60
 - PSK_KEY, 60
 - PSK_MODE, 50
 - PSK_NOT, 60
 - PSKOK, 48
 - PSKWECDHE, 48
 - RAW_PUBLIC_KEY, 60
 - RECORD_OVERFLOW, 58
 - RECORD_SIZE_LIMIT, 51
 - RSA_PKCS1_SHA256, 47
 - RSA_PKCS1_SHA384, 47
 - RSA_PKCS1_SHA512, 47
 - RSA_PSS_RSAE_SHA256, 46
 - RSA_PSS_RSAE_SHA384, 47

RSA_PSS_RSAE_SHA512, 47
RSA_SS, 37
SECP256R1, 45
SECP384R1, 45
SECP521R1, 45
SELF_SIGNED_CERT, 56
SERVER_CERT_TYPE, 51
SERVER_HELLO, 52
SERVER_NAME, 49
SIG_ALGS, 49
SIG_ALGS_CERT, 49
sign16, 61
sign32, 61
sign64, 61
sign8, 61
SUPPORTED_GROUPS, 49
TICKET, 53
TIME_OUT, 56
TIMED_OUT, 52
TINY_ECC, 36
TLS13_CONNECTED, 59
TLS13_DISCONNECTED, 59
TLS13_HANDSHAKING, 59
TLS13_UPDATE_NOT_REQUESTED, 49
TLS13_UPDATE_REQUESTED, 49
TLS1_0, 48
TLS1_2, 48
TLS1_3, 49
TLS_AES_128_CCM_8_SHA256, 45
TLS_AES_128_CCM_SHA256, 45
TLS_AES_128_GCM_SHA256, 44
TLS_AES_256_GCM_SHA384, 44
TLS_APPLICATION_PROTOCOL, 38
TLS_CHACHA20_POLY1305_SHA256, 45
TLS_EARLY_DATA_ACCEPTED, 60
TLS_EXTERNAL_PSK, 48
TLS_FAILURE, 59
TLS_FULL_HANDSHAKE, 48
TLS_MAX_CERT_B64, 41
TLS_MAX_CERT_SIZE, 41
TLS_MAX_CIPHER_FRAG, 41
TLS_MAX_CIPHER_SUITES, 44
TLS_MAX_CLIENT_CHAIN_LEN, 42
TLS_MAX_CLIENT_CHAIN_SIZE, 42
TLS_MAX_COOKIE, 43
TLS_MAX_ECC_FIELD, 43
TLS_MAX_EXT_LABEL, 40
TLS_MAX_EXTENSIONS, 43
TLS_MAX_FRAG, 40
TLS_MAX_HASH, 40
TLS_MAX_HASH_STATE, 40
TLS_MAX_HELLO, 41
TLS_MAX_IBUFF_SIZE, 40
TLS_MAX_IV_SIZE, 43
TLS_MAX_KEX_CIPHERTEXT_SIZE, 42
TLS_MAX_KEX_PUB_KEY_SIZE, 42
TLS_MAX_KEX_SECRET_KEY_SIZE, 42
TLS_MAX_KEY, 40
TLS_MAX_OBUFF_SIZE, 44
TLS_MAX_OUTPUT_RECORD_SIZE, 43
TLS_MAX_PLAIN_FRAG, 41
TLS_MAX_PSK_MODES, 44
TLS_MAX_SERVER_CHAIN_LEN, 42
TLS_MAX_SERVER_CHAIN_SIZE, 42
TLS_MAX_SERVER_NAME, 44
TLS_MAX_SHARED_SECRET_SIZE, 43
TLS_MAX_SIG_PUB_KEY_SIZE, 41
TLS_MAX_SIG_SECRET_KEY_SIZE, 41
TLS_MAX_SIGNATURE_SIZE, 42
TLS_MAX_SUPPORTED_GROUPS, 44
TLS_MAX_SUPPORTED_SIGS, 44
TLS_MAX_TAG_SIZE, 43
TLS_MAX_TICKET_SIZE, 43
TLS_RESUMPTION_REQUIRED, 59
TLS_SHA256_T, 39
TLS_SHA384_T, 39
TLS_SHA512_T, 39
TLS_SUCCESS, 59
TLS_VER, 50
TLS_X509_MAX_FIELD, 40
TRY_EARLY_DATA, 39
TYPICAL, 36
UNEXPECTED_MESSAGE, 57
UNKNOWN_CA, 58
UNRECOGNIZED_EXT, 54
unsign16, 61
unsign32, 61
unsign64, 61
UNSUPPORTED_EXTENSION, 57
VERBOSITY, 38
WRONG_MESSAGE, 54
X25519, 45
X448, 45
X509_CERT, 60
TLS_AES_128_CCM_8_SHA256
 tls1_3.h, 45
TLS_AES_128_CCM_SHA256
 tls1_3.h, 45
TLS_AES_128_GCM_SHA256
 tls1_3.h, 44
TLS_AES_256_GCM_SHA384
 tls1_3.h, 44
TLS_APPLICATION_PROTOCOL
 tls1_3.h, 38
tls_bfibe.h, 67, 68
 BFIBE_CCA_DECRYPT, 68
 BFIBE_CCA_ENCRYPT, 67
tls_cert_chain.h, 69, 70
 checkServerCertChain, 69
tls_certs.h, 70, 71
 cacerts, 71
 mycert, 71
 myprivate, 71
 setCredential, 70
TLS_CHACHA20_POLY1305_SHA256
 tls1_3.h, 45

- tls_client_rcv.h, 72, 79
 - badResponse, 76
 - getCertificateRequest, 79
 - getCheckServerCertificateChain, 79
 - getServerCertVerify, 77
 - getServerEncryptedExtensions, 77
 - getServerFinished, 78
 - getServerHello, 78
 - getServerRecord, 74
 - parsebytes, 73
 - parsebytesorPull, 75
 - parseInt, 74
 - parseIntorPull, 75
 - parseoctad, 73
 - parseoctadorPull, 75
 - parseoctadorPullptrX, 76
 - parseoctadptr, 74
 - seeWhatsNext, 77
- tls_client_send.h, 80, 92
 - addALPNExt, 85
 - addCertAuthorities, 82
 - addClientRawPublicKey, 84
 - addCookieExt, 87
 - addEarlyDataExt, 87
 - addHeartbeat, 82
 - addKeyShareExt, 85
 - addMFLExt, 85
 - addPadding, 86
 - addPostHSAuth, 87
 - addPreSharedKeyExt, 83
 - addPSKModesExt, 86
 - addRSLExt, 86
 - addServerNameExt, 83
 - addServerRawPublicKey, 84
 - addSigAlgsCertExt, 85
 - addSigAlgsExt, 84
 - addSupportedGroupsExt, 83
 - addVersionExt, 86
 - alert_from_cause, 92
 - cipherSuites, 88
 - clientRandom, 88
 - sendAlert, 90
 - sendBinder, 89
 - sendCCCS, 82
 - sendClientCertificateChain, 91
 - sendClientCertVerify, 91
 - sendClientFinish, 91
 - sendClientHello, 90
 - sendClientMessage, 89
 - sendEndOfEarlyData, 92
 - sendHeartbeatRequest, 89
 - sendKeyUpdate, 90
 - sendRecord, 88
 - sendZeroRecord, 89
- TLS_EARLY_DATA_ACCEPTED
 - tls1_3.h, 60
- TLS_EXTERNAL_PSK
 - tls1_3.h, 48
- TLS_FAILURE
 - tls1_3.h, 59
- TLS_FULL_HANDSHAKE
 - tls1_3.h, 48
- tls_keys_calc.h, 93, 103
 - checkServerCertVerifier, 101
 - checkVerifierData, 100
 - createClientCertVerifier, 101
 - createCryptoContext, 97
 - createRecvCryptoContext, 98
 - createSendCryptoContext, 98
 - deriveApplicationSecrets, 100
 - deriveEarlySecrets, 98
 - deriveHandshakeSecrets, 99
 - deriveLaterSecrets, 99
 - deriveUpdatedKeys, 100
 - deriveVerifierData, 101
 - incrementCryptoContext, 97
 - initCryptoContext, 97
 - initTranscriptHash, 95
 - recoverPSK, 98
 - rewindIO, 95
 - runningHash, 95
 - runningHashIO, 95
 - runningHashIOrewind, 96
 - runningSyntheticHash, 96
 - transcriptHash, 96
 - updateCryptoContext, 97
- tls_logger.h, 103, 108
 - log, 105
 - logAlert, 107
 - logCert, 106
 - logCertDetails, 106
 - logCipherSuite, 107
 - logEncExt, 106
 - logKeyExchange, 107
 - logServerHello, 105
 - logServerResponse, 107
 - logSigAlg, 107
 - logTicket, 105
 - myprintf, 104
- TLS_MAX_CERT_B64
 - tls1_3.h, 41
- TLS_MAX_CERT_SIZE
 - tls1_3.h, 41
- TLS_MAX_CIPHER_FRAG
 - tls1_3.h, 41
- TLS_MAX_CIPHER_SUITES
 - tls1_3.h, 44
- TLS_MAX_CLIENT_CHAIN_LEN
 - tls1_3.h, 42
- TLS_MAX_CLIENT_CHAIN_SIZE
 - tls1_3.h, 42
- TLS_MAX_COOKIE
 - tls1_3.h, 43
- TLS_MAX_ECC_FIELD
 - tls1_3.h, 43
- TLS_MAX_EXT_LABEL

- tls1_3.h, [40](#)
- TLS_MAX_EXTENSIONS
 - tls1_3.h, [43](#)
- TLS_MAX_FRAG
 - tls1_3.h, [40](#)
- TLS_MAX_HASH
 - tls1_3.h, [40](#)
- TLS_MAX_HASH_STATE
 - tls1_3.h, [40](#)
- TLS_MAX_HELLO
 - tls1_3.h, [41](#)
- TLS_MAX_IBUFF_SIZE
 - tls1_3.h, [40](#)
- TLS_MAX_IV_SIZE
 - tls1_3.h, [43](#)
- TLS_MAX_KEX_CIPHERTEXT_SIZE
 - tls1_3.h, [42](#)
- TLS_MAX_KEX_PUB_KEY_SIZE
 - tls1_3.h, [42](#)
- TLS_MAX_KEX_SECRET_KEY_SIZE
 - tls1_3.h, [42](#)
- TLS_MAX_KEY
 - tls1_3.h, [40](#)
- TLS_MAX_OBUFF_SIZE
 - tls1_3.h, [44](#)
- TLS_MAX_OUTPUT_RECORD_SIZE
 - tls1_3.h, [43](#)
- TLS_MAX_PLAIN_FRAG
 - tls1_3.h, [41](#)
- TLS_MAX_PSK_MODES
 - tls1_3.h, [44](#)
- TLS_MAX_SERVER_CHAIN_LEN
 - tls1_3.h, [42](#)
- TLS_MAX_SERVER_CHAIN_SIZE
 - tls1_3.h, [42](#)
- TLS_MAX_SERVER_NAME
 - tls1_3.h, [44](#)
- TLS_MAX_SHARED_SECRET_SIZE
 - tls1_3.h, [43](#)
- TLS_MAX_SIG_PUB_KEY_SIZE
 - tls1_3.h, [41](#)
- TLS_MAX_SIG_SECRET_KEY_SIZE
 - tls1_3.h, [41](#)
- TLS_MAX_SIGNATURE_SIZE
 - tls1_3.h, [42](#)
- TLS_MAX_SUPPORTED_GROUPS
 - tls1_3.h, [44](#)
- TLS_MAX_SUPPORTED_SIGS
 - tls1_3.h, [44](#)
- TLS_MAX_TAG_SIZE
 - tls1_3.h, [43](#)
- TLS_MAX_TICKET_SIZE
 - tls1_3.h, [43](#)
- tls_octads.h, [108](#), [115](#)
 - millis, [110](#)
 - OCT_append_byte, [112](#)
 - OCT_append_bytes, [112](#)
 - OCT_append_int, [110](#)
 - OCT_append_octad, [110](#)
 - OCT_append_string, [112](#)
 - OCT_compare, [110](#)
 - OCT_copy, [113](#)
 - OCT_from_base64, [112](#)
 - OCT_from_hex, [111](#)
 - OCT_kill, [111](#)
 - OCT_output_base64, [114](#)
 - OCT_output_hex, [113](#)
 - OCT_output_string, [114](#)
 - OCT_reverse, [113](#)
 - OCT_shift_left, [111](#)
 - OCT_truncate, [113](#)
- tls_pqibe.h, [115](#), [116](#)
 - PQIBE_CCA_DECRYPT, [116](#)
 - PQIBE_CCA_ENCRYPT, [116](#)
- tls_protocol.h, [117](#), [121](#)
 - TLS13_clean, [121](#)
 - TLS13_connect, [119](#)
 - TLS13_end, [119](#)
 - TLS13_rcv, [120](#)
 - TLS13_rcv_and_check, [120](#)
 - TLS13_send, [120](#)
 - TLS13_start, [118](#)
 - TLS13_stop, [119](#)
- TLS_RESUMPTION_REQUIRED
 - tls1_3.h, [59](#)
- tls_sal.h, [121](#), [131](#)
 - SAL_aeadDecrypt, [129](#)
 - SAL_aeadEncrypt, [128](#)
 - SAL_aeadKeylen, [125](#)
 - SAL_aeadTaglen, [125](#)
 - SAL_ciphers, [123](#)
 - SAL_generateKeyPair, [129](#)
 - SAL_generateSharedSecret, [130](#)
 - SAL_groups, [123](#)
 - SAL_hashInit, [127](#)
 - SAL_hashLen, [125](#)
 - SAL_hashNull, [127](#)
 - SAL_hashOutput, [128](#)
 - SAL_hashProcessArray, [128](#)
 - SAL_hashType, [124](#)
 - SAL_hkdfExpand, [127](#)
 - SAL_hkdfExtract, [126](#)
 - SAL_hmac, [127](#)
 - SAL_initLib, [124](#)
 - SAL_name, [123](#)
 - SAL_randomByte, [126](#)
 - SAL_randomOctad, [126](#)
 - SAL_sigCerts, [124](#)
 - SAL_sigs, [124](#)
 - SAL_tlsSignature, [130](#)
 - SAL_tlsSignatureVerify, [130](#)
- TLS_session, [24](#)
 - allowed_to_heartbeat, [28](#)
 - cipher_suite, [25](#)
 - client_cert_type, [26](#)
 - CTS, [27](#)

- cts, [27](#)
- CTX, [27](#)
- ctx, [27](#)
- expect_heartbeats, [28](#)
- favourite_group, [25](#)
- heartbeat_req_in_flight, [28](#)
- hostname, [25](#)
- HS, [26](#)
- hs, [26](#)
- IBUFF, [27](#)
- ibuff, [28](#)
- id, [25](#)
- K_rcv, [26](#)
- K_send, [26](#)
- max_record, [25](#)
- OBUFF, [27](#)
- obuf, [28](#)
- ptr, [28](#)
- RMS, [26](#)
- rms, [26](#)
- server_cert_type, [26](#)
- sockptr, [25](#)
- status, [25](#)
- STS, [27](#)
- sts, [27](#)
- T, [28](#)
- tlshash, [28](#)
- TLS_SHA256_T
 - tls1_3.h, [39](#)
- TLS_SHA384_T
 - tls1_3.h, [39](#)
- TLS_SHA512_T
 - tls1_3.h, [39](#)
- tls_sockets.h, [132](#), [136](#)
 - getBytes, [135](#)
 - getBytes, [134](#)
 - getInt16, [134](#)
 - getInt24, [135](#)
 - getIPaddress, [133](#)
 - getOctad, [135](#)
 - sendLen, [134](#)
 - sendOctad, [133](#)
 - setclientsock, [133](#)
- TLS_SUCCESS
 - tls1_3.h, [59](#)
- tls_tickets.h, [137](#), [139](#)
 - endTicketContext, [138](#)
 - initTicketContext, [138](#)
 - parseTicket, [138](#)
 - ticket_still_good, [139](#)
- TLS_VER
 - tls1_3.h, [50](#)
- tls_wifi.h, [139](#), [140](#)
- tls_x509.h, [140](#), [151](#)
 - ecdsa_sig_decode, [144](#)
 - ecdsa_sig_encode, [143](#)
 - USE_ED25519, [143](#)
 - USE_ED448, [143](#)
 - USE_NIST256, [143](#)
 - USE_NIST384, [143](#)
 - USE_NIST521, [143](#)
 - X509_AN, [150](#)
 - X509_BC, [150](#)
 - X509_CN, [149](#)
 - X509_ECC, [142](#)
 - X509_ECD, [142](#)
 - X509_EN, [149](#)
 - X509_extract_cert, [145](#)
 - X509_extract_cert_sig, [144](#)
 - X509_extract_private_key, [144](#)
 - X509_extract_public_key, [146](#)
 - X509_find_alt_name, [149](#)
 - X509_find_entity_property, [147](#)
 - X509_find_expiry_date, [148](#)
 - X509_find_extension, [148](#)
 - X509_find_extensions, [148](#)
 - X509_find_issuer, [146](#)
 - X509_find_public_key, [145](#)
 - X509_find_start_date, [147](#)
 - X509_find_subject, [146](#)
 - X509_find_validity, [146](#)
 - X509_get_public_key, [145](#)
 - X509_H256, [142](#)
 - X509_H384, [142](#)
 - X509_H512, [143](#)
 - X509_HY, [142](#)
 - X509_KU, [150](#)
 - X509_LN, [150](#)
 - X509_MN, [150](#)
 - X509_ON, [149](#)
 - X509_PQ, [142](#)
 - X509_RSA, [142](#)
 - X509_self_signed, [147](#)
 - X509_SHAKE256, [143](#)
 - X509_SN, [150](#)
 - X509_UN, [150](#)
- TLS_X509_MAX_FIELD
 - tls1_3.h, [40](#)
- tlshash
 - TLS_session, [28](#)
- transcriptHash
 - tls_keys_calc.h, [96](#)
- TRY_EARLY_DATA
 - tls1_3.h, [39](#)
- type
 - pktype, [20](#)
- TYPICAL
 - tls1_3.h, [36](#)
- UNEXPECTED_MESSAGE
 - tls1_3.h, [57](#)
- unihash, [29](#)
 - htype, [29](#)
 - state, [29](#)
- UNKNOWN_CA
 - tls1_3.h, [58](#)
- UNRECOGNIZED_EXT

tls1_3.h, [54](#)
unsign16
 tls1_3.h, [61](#)
unsign32
 tls1_3.h, [61](#)
unsign64
 tls1_3.h, [61](#)
UNSUPPORTED_EXTENSION
 tls1_3.h, [57](#)
updateCryptoContext
 tls_keys_calc.h, [97](#)
USE_ED25519
 tls_x509.h, [143](#)
USE_ED448
 tls_x509.h, [143](#)
USE_NIST256
 tls_x509.h, [143](#)
USE_NIST384
 tls_x509.h, [143](#)
USE_NIST521
 tls_x509.h, [143](#)

val
 octad, [19](#)
 ret, [20](#)
valid
 ticket, [22](#)
VERBOSITY
 tls1_3.h, [38](#)

WRONG_MESSAGE
 tls1_3.h, [54](#)

X25519
 tls1_3.h, [45](#)
X448
 tls1_3.h, [45](#)
X509_AN
 tls_x509.h, [150](#)
X509_BC
 tls_x509.h, [150](#)
X509_CERT
 tls1_3.h, [60](#)
X509_CN
 tls_x509.h, [149](#)
X509_ECC
 tls_x509.h, [142](#)
X509_ECD
 tls_x509.h, [142](#)
X509_EN
 tls_x509.h, [149](#)
X509_extract_cert
 tls_x509.h, [145](#)
X509_extract_cert_sig
 tls_x509.h, [144](#)
X509_extract_private_key
 tls_x509.h, [144](#)
X509_extract_public_key
 tls_x509.h, [146](#)

X509_find_alt_name
 tls_x509.h, [149](#)
X509_find_entity_property
 tls_x509.h, [147](#)
X509_find_expiry_date
 tls_x509.h, [148](#)
X509_find_extension
 tls_x509.h, [148](#)
X509_find_extensions
 tls_x509.h, [148](#)
X509_find_issuer
 tls_x509.h, [146](#)
X509_find_public_key
 tls_x509.h, [145](#)
X509_find_start_date
 tls_x509.h, [147](#)
X509_find_subject
 tls_x509.h, [146](#)
X509_find_validity
 tls_x509.h, [146](#)
X509_get_public_key
 tls_x509.h, [145](#)
X509_H256
 tls_x509.h, [142](#)
X509_H384
 tls_x509.h, [142](#)
X509_H512
 tls_x509.h, [143](#)
X509_HY
 tls_x509.h, [142](#)
X509_KU
 tls_x509.h, [150](#)
X509_LN
 tls_x509.h, [150](#)
X509_MN
 tls_x509.h, [150](#)
X509_ON
 tls_x509.h, [149](#)
X509_PQ
 tls_x509.h, [142](#)
X509_RSA
 tls_x509.h, [142](#)
X509_self_signed
 tls_x509.h, [147](#)
X509_SHAKE256
 tls_x509.h, [143](#)
X509_SN
 tls_x509.h, [150](#)
X509_UN
 tls_x509.h, [150](#)