

TII TLS1.3

1.1

Generated by Doxygen 1.9.1

1 TII TLS C	1
1.1 Building	1
1.1.1 Miracl	1
1.1.2 Miracl + LibSodium	1
1.1.3 Custom Crypto Library	1
1.1.4 Client side Authentication	4
1.1.5 Testing Pre-shared keys	4
2 Description	5
2.1 Linux Installation	5
2.1.1 Client side Authentication	8
2.1.2 Testing Pre-shared keys	9
2.2 Building the client application on an Arduino board (here Arduino Nano 33 IoT)	9
3 Configure the Arduino Nano RP2040	11
3.1 Building the client application on the Arduino Nano RP2040 board.	11
4 Data Structure Index	13
4.1 Data Structures	13
5 File Index	15
5.1 File List	15
6 Data Structure Documentation	17
6.1 crypto Struct Reference	17
6.1.1 Detailed Description	17
6.1.2 Field Documentation	17
6.1.2.1 active	17
6.1.2.2 k	18
6.1.2.3 iv	18
6.1.2.4 K	18
6.1.2.5 IV	18
6.1.2.6 record	18
6.1.2.7 suite	18
6.1.2.8 taglen	18
6.2 ECCX08Class Class Reference	19
6.3 ee_status Struct Reference	19
6.3.1 Detailed Description	20
6.3.2 Field Documentation	20
6.3.2.1 early_data	20
6.3.2.2 alpn	20
6.3.2.3 server_name	20
6.3.2.4 max_frag_length	20
6.4 octad Struct Reference	20

6.4.1 Detailed Description	21
6.4.2 Field Documentation	21
6.4.2.1 len	21
6.4.2.2 max	21
6.4.2.3 val	21
6.5 pktype Struct Reference	21
6.5.1 Detailed Description	22
6.5.2 Field Documentation	22
6.5.2.1 type	22
6.5.2.2 hash	22
6.5.2.3 curve	22
6.6 ret Struct Reference	22
6.6.1 Detailed Description	23
6.6.2 Field Documentation	23
6.6.2.1 val	23
6.6.2.2 err	23
6.7 Socket Class Reference	23
6.7.1 Detailed Description	24
6.8 ticket Struct Reference	24
6.8.1 Detailed Description	24
6.8.2 Field Documentation	24
6.8.2.1 valid	25
6.8.2.2 tick	25
6.8.2.3 nonce	25
6.8.2.4 psk	25
6.8.2.5 TICK	25
6.8.2.6 NONCE	25
6.8.2.7 PSK	25
6.8.2.8 age_obfuscator	25
6.8.2.9 max_early_data	26
6.8.2.10 birth	26
6.8.2.11 lifetime	26
6.8.2.12 cipher_suite	26
6.8.2.13 favourite_group	26
6.8.2.14 origin	26
6.9 TLS_session Struct Reference	26
6.9.1 Detailed Description	27
6.9.2 Field Documentation	27
6.9.2.1 status	27
6.9.2.2 max_record	27
6.9.2.3 sockptr	27
6.9.2.4 id	28

6.9.2.5 hostname	28
6.9.2.6 cipher_suite	28
6.9.2.7 favourite_group	28
6.9.2.8 K_send	28
6.9.2.9 K_rcv	28
6.9.2.10 HS	28
6.9.2.11 hs	28
6.9.2.12 RMS	29
6.9.2.13 rms	29
6.9.2.14 STS	29
6.9.2.15 sts	29
6.9.2.16 CTS	29
6.9.2.17 cts	29
6.9.2.18 IO	29
6.9.2.19 io	29
6.9.2.20 ptr	30
6.9.2.21 tlshash	30
6.9.2.22 T	30
6.10 unihash Struct Reference	30
6.10.1 Detailed Description	30
6.10.2 Field Documentation	30
6.10.2.1 state	30
6.10.2.2 htype	30
7 File Documentation	31
7.1 tls1_3.h File Reference	31
7.1.1 Detailed Description	34
7.1.2 Macro Definition Documentation	35
7.1.2.1 IO_NONE	35
7.1.2.2 IO_APPLICATION	35
7.1.2.3 IO_PROTOCOL	35
7.1.2.4 IO_DEBUG	35
7.1.2.5 IO_WIRE	35
7.1.2.6 TINY_ECC	35
7.1.2.7 TYPICAL	35
7.1.2.8 POST_QUANTUM	36
7.1.2.9 NOCERT	36
7.1.2.10 RSA_SS	36
7.1.2.11 ECC_SS	36
7.1.2.12 DLT_SS	36
7.1.2.13 HW_1	36
7.1.2.14 HW_2	36

7.1.2.15 VERBOSITY	36
7.1.2.16 THIS_YEAR	37
7.1.2.17 CLIENT_CERT	37
7.1.2.18 TLS_APPLICATION_PROTOCOL	37
7.1.2.19 ALLOW_SELF_SIGNED	37
7.1.2.20 CRYPTO_SETTING	37
7.1.2.21 TRY_EARLY_DATA	37
7.1.2.22 TLS_SHA256_T	37
7.1.2.23 TLS_SHA384_T	38
7.1.2.24 TLS_SHA512_T	38
7.1.2.25 TLS_MAX_HASH_STATE	38
7.1.2.26 TLS_MAX_HASH	38
7.1.2.27 TLS_MAX_KEY	38
7.1.2.28 TLS_X509_MAX_FIELD	38
7.1.2.29 TLS_MAX_EXT_LABEL	38
7.1.2.30 TLS_MAX_FRAG	39
7.1.2.31 TLS_MAX_IO_SIZE	39
7.1.2.32 TLS_MAX_PLAIN_FRAG	39
7.1.2.33 TLS_MAX_CIPHER_FRAG	39
7.1.2.34 TLS_MAX_CERT_SIZE	39
7.1.2.35 TLS_MAX_CERT_B64	39
7.1.2.36 TLS_MAX_HELLO	39
7.1.2.37 TLS_MAX_SIG_PUB_KEY_SIZE	40
7.1.2.38 TLS_MAX_SIG_SECRET_KEY_SIZE	40
7.1.2.39 TLS_MAX_SIGNATURE_SIZE	40
7.1.2.40 TLS_MAX_KEX_PUB_KEY_SIZE	40
7.1.2.41 TLS_MAX_KEX_CIPHERTEXT_SIZE	40
7.1.2.42 TLS_MAX_KEX_SECRET_KEY_SIZE	40
7.1.2.43 TLS_MAX_SERVER_CHAIN_LEN	40
7.1.2.44 TLS_MAX_SERVER_CHAIN_SIZE	40
7.1.2.45 TLS_MAX_CLIENT_CHAIN_LEN	41
7.1.2.46 TLS_MAX_CLIENT_CHAIN_SIZE	41
7.1.2.47 TLS_MAX_SHARED_SECRET_SIZE	41
7.1.2.48 TLS_MAX_TICKET_SIZE	41
7.1.2.49 TLS_MAX_EXTENSIONS	41
7.1.2.50 TLS_MAX_ECC_FIELD	41
7.1.2.51 TLS_MAX_IV_SIZE	41
7.1.2.52 TLS_MAX_TAG_SIZE	41
7.1.2.53 TLS_MAX_COOKIE	42
7.1.2.54 TLS_MAX_SERVER_NAME	42
7.1.2.55 TLS_MAX_SUPPORTED_GROUPS	42
7.1.2.56 TLS_MAX_SUPPORTED_SIGS	42

7.1.2.57 TLS_MAX_PSK_MODES	42
7.1.2.58 TLS_MAX_CIPHER_SUITES	42
7.1.2.59 TLS_AES_128_GCM_SHA256	42
7.1.2.60 TLS_AES_256_GCM_SHA384	42
7.1.2.61 TLS_CHACHA20_POLY1305_SHA256	43
7.1.2.62 TLS_AES_128_CCM_SHA256	43
7.1.2.63 TLS_AES_128_CCM_8_SHA256	43
7.1.2.64 X25519	43
7.1.2.65 SECP256R1	43
7.1.2.66 SECP384R1	43
7.1.2.67 SECP521R1	43
7.1.2.68 X448	43
7.1.2.69 KYBER768	44
7.1.2.70 ECDSA_SECP256R1_SHA256	44
7.1.2.71 ECDSA_SECP384R1_SHA384	44
7.1.2.72 RSA_PSS_RSAE_SHA256	44
7.1.2.73 RSA_PSS_RSAE_SHA384	44
7.1.2.74 RSA_PSS_RSAE_SHA512	44
7.1.2.75 RSA_PKCS1_SHA256	44
7.1.2.76 RSA_PKCS1_SHA384	44
7.1.2.77 RSA_PKCS1_SHA512	45
7.1.2.78 ED25519	45
7.1.2.79 DILITHIUM3	45
7.1.2.80 PSKOK	45
7.1.2.81 PSKWECDHE	45
7.1.2.82 TLS_FULL_HANDSHAKE	45
7.1.2.83 TLS_EXTERNAL_PSK	45
7.1.2.84 TLS1_0	45
7.1.2.85 TLS1_2	46
7.1.2.86 TLS1_3	46
7.1.2.87 SERVER_NAME	46
7.1.2.88 SUPPORTED_GROUPS	46
7.1.2.89 SIG_ALGS	46
7.1.2.90 SIG_ALGS_CERT	46
7.1.2.91 KEY_SHARE	46
7.1.2.92 PSK_MODE	46
7.1.2.93 PRESARED_KEY	47
7.1.2.94 TLS_VER	47
7.1.2.95 COOKIE	47
7.1.2.96 EARLY_DATA	47
7.1.2.97 MAX_FRAG_LENGTH	47
7.1.2.98 PADDING	47

7.1.2.99 APP_PROTOCOL	47
7.1.2.100 RECORD_SIZE_LIMIT	47
7.1.2.101 HSHAKE	48
7.1.2.102 APPLICATION	48
7.1.2.103 ALERT	48
7.1.2.104 CHANGE_CIPHER	48
7.1.2.105 TIMED_OUT	48
7.1.2.106 CLIENT_HELLO	48
7.1.2.107 SERVER_HELLO	48
7.1.2.108 CERTIFICATE	49
7.1.2.109 CERT_REQUEST	49
7.1.2.110 CERT_VERIFY	49
7.1.2.111 FINISHED	49
7.1.2.112 ENCRYPTED_EXTENSIONS	49
7.1.2.113 TICKET	49
7.1.2.114 KEY_UPDATE	49
7.1.2.115 MESSAGE_HASH	49
7.1.2.116 END_OF_EARLY_DATA	50
7.1.2.117 HANDSHAKE_RETRY	50
7.1.2.118 NOT_TLS1_3	50
7.1.2.119 BAD_CERT_CHAIN	50
7.1.2.120 ID_MISMATCH	50
7.1.2.121 UNRECOGNIZED_EXT	50
7.1.2.122 BAD_HELLO	50
7.1.2.123 WRONG_MESSAGE	50
7.1.2.124 MISSING_REQUEST_CONTEXT	51
7.1.2.125 AUTHENTICATION_FAILURE	51
7.1.2.126 BAD_RECORD	51
7.1.2.127 BAD_TICKET	51
7.1.2.128 NOT_EXPECTED	51
7.1.2.129 CA_NOT_FOUND	51
7.1.2.130 CERT_OUTOFDATE	51
7.1.2.131 MEM_OVERFLOW	51
7.1.2.132 FORBIDDEN_EXTENSION	52
7.1.2.133 MAX_EXCEEDED	52
7.1.2.134 EMPTY_CERT_CHAIN	52
7.1.2.135 SELF_SIGNED_CERT	52
7.1.2.136 BAD_MESSAGE	52
7.1.2.137 ILLEGAL_PARAMETER	52
7.1.2.138 UNEXPECTED_MESSAGE	52
7.1.2.139 DECRYPT_ERROR	52
7.1.2.140 BAD_CERTIFICATE	53

7.1.2.141 UNSUPPORTED_EXTENSION	53
7.1.2.142 UNKNOWN_CA	53
7.1.2.143 CERTIFICATE_EXPIRED	53
7.1.2.144 PROTOCOL_VERSION	53
7.1.2.145 DECODE_ERROR	53
7.1.2.146 RECORD_OVERFLOW	53
7.1.2.147 CLOSE_NOTIFY	53
7.1.2.148 LOG_OUTPUT_TRUNCATION	54
7.1.2.149 TLS13_DISCONNECTED	54
7.1.2.150 TLS13_CONNECTED	54
7.1.2.151 TLS_FAILURE	54
7.1.2.152 TLS_SUCCESS	54
7.1.2.153 TLS_RESUMPTION_REQUIRED	54
7.1.2.154 TLS_EARLY_DATA_ACCEPTED	54
7.1.3 Typedef Documentation	54
7.1.3.1 byte	55
7.1.3.2 sign8	55
7.1.3.3 sign16	55
7.1.3.4 sign32	55
7.1.3.5 sign64	55
7.1.3.6 unsign32	55
7.1.3.7 unsign64	55
7.2 tls_cert_chain.h File Reference	56
7.2.1 Detailed Description	56
7.2.2 Function Documentation	56
7.2.2.1 checkServerCertChain()	56
7.2.2.2 getClientPrivateKeyandCertChain()	57
7.3 tls_certs.h File Reference	57
7.3.1 Detailed Description	57
7.3.2 Variable Documentation	58
7.3.2.1 myprivate	58
7.3.2.2 mycert	58
7.3.2.3 cacerts	58
7.4 tls_client_rcv.h File Reference	58
7.4.1 Detailed Description	59
7.4.2 Function Documentation	59
7.4.2.1 parseoctad()	60
7.4.2.2 parsebytes()	60
7.4.2.3 parseInt()	60
7.4.2.4 parseoctadptr()	61
7.4.2.5 getServerFragment()	61
7.4.2.6 parseIntorPull()	62

7.4.2.7 parseoctadorPull()	62
7.4.2.8 parsebytesorPull()	63
7.4.2.9 parseoctadorPullptrX()	63
7.4.2.10 badResponse()	63
7.4.2.11 seeWhatsNext()	64
7.4.2.12 getServerEncryptedExtensions()	64
7.4.2.13 getServerCertVerify()	65
7.4.2.14 getServerFinished()	65
7.4.2.15 getServerHello()	65
7.4.2.16 getCheckServerCertificateChain()	66
7.4.2.17 getCertificateRequest()	66
7.5 <code>tls_client_send.h</code> File Reference	67
7.5.1 Detailed Description	68
7.5.2 Function Documentation	68
7.5.2.1 sendCCCS()	68
7.5.2.2 addPreSharedKeyExt()	69
7.5.2.3 addServerNameExt()	69
7.5.2.4 addSupportedGroupsExt()	69
7.5.2.5 addSigAlgsExt()	70
7.5.2.6 addSigAlgsCertExt()	70
7.5.2.7 addKeyShareExt()	70
7.5.2.8 addALPNExt()	71
7.5.2.9 addMFLExt()	71
7.5.2.10 addRSLExt()	71
7.5.2.11 addPSKModesExt()	72
7.5.2.12 addVersionExt()	72
7.5.2.13 addPadding()	72
7.5.2.14 addCookieExt()	72
7.5.2.15 addEarlyDataExt()	73
7.5.2.16 clientRandom()	73
7.5.2.17 cipherSuites()	73
7.5.2.18 sendFlushIO()	74
7.5.2.19 sendClientMessage()	74
7.5.2.20 sendBinder()	75
7.5.2.21 sendClientHello()	75
7.5.2.22 sendAlert()	75
7.5.2.23 sendClientFinish()	76
7.5.2.24 sendClientCertificateChain()	76
7.5.2.25 sendClientCertVerify()	76
7.5.2.26 sendEndOfEarlyData()	77
7.5.2.27 alert_from_cause()	77
7.6 <code>tls_keys_calc.h</code> File Reference	77

7.6.1 Detailed Description	79
7.6.2 Function Documentation	79
7.6.2.1 initTranscriptHash()	79
7.6.2.2 runningHash()	79
7.6.2.3 runningHashIO()	79
7.6.2.4 rewindIO()	80
7.6.2.5 runningHashIOrewind()	80
7.6.2.6 transcriptHash()	80
7.6.2.7 runningSyntheticHash()	81
7.6.2.8 initCryptoContext()	81
7.6.2.9 updateCryptoContext()	81
7.6.2.10 incrementCryptoContext()	82
7.6.2.11 createCryptoContext()	82
7.6.2.12 createSendCryptoContext()	82
7.6.2.13 createRecvCryptoContext()	82
7.6.2.14 recoverPSK()	83
7.6.2.15 deriveEarlySecrets()	83
7.6.2.16 deriveLaterSecrets()	83
7.6.2.17 deriveHandshakeSecrets()	84
7.6.2.18 deriveApplicationSecrets()	84
7.6.2.19 deriveUpdatedKeys()	85
7.6.2.20 checkVerifierData()	85
7.6.2.21 deriveVerifierData()	86
7.6.2.22 checkServerCertVerifier()	86
7.6.2.23 createClientCertVerifier()	86
7.7 tls_logger.h File Reference	87
7.7.1 Detailed Description	87
7.7.2 Function Documentation	88
7.7.2.1 myprintf()	88
7.7.2.2 log()	88
7.7.2.3 logServerHello()	88
7.7.2.4 logTicket()	89
7.7.2.5 logEncExt()	89
7.7.2.6 logCert()	89
7.7.2.7 logCertDetails()	90
7.7.2.8 logServerResponse()	90
7.7.2.9 logAlert()	90
7.7.2.10 logCipherSuite()	91
7.7.2.11 logKeyExchange()	91
7.7.2.12 logSigAlg()	91
7.8 tls_octads.h File Reference	91
7.8.1 Detailed Description	92

7.8.2 Function Documentation	93
7.8.2.1 OCT_append_int()	93
7.8.2.2 OCT_append_octad()	93
7.8.2.3 OCT_compare()	93
7.8.2.4 OCT_shift_left()	94
7.8.2.5 OCT_kill()	94
7.8.2.6 OCT_from_hex()	94
7.8.2.7 OCT_append_string()	95
7.8.2.8 OCT_append_byte()	95
7.8.2.9 OCT_append_bytes()	95
7.8.2.10 OCT_from_base64()	96
7.8.2.11 OCT_reverse()	96
7.8.2.12 OCT_truncate()	96
7.8.2.13 OCT_copy()	96
7.8.2.14 OCT_output_hex()	97
7.8.2.15 OCT_output_string()	97
7.8.2.16 OCT_output_base64()	97
7.9 tls_protocol.h File Reference	98
7.9.1 Detailed Description	98
7.9.2 Function Documentation	99
7.9.2.1 TLS13_start()	99
7.9.2.2 TLS13_end()	99
7.9.2.3 TLS13_connect()	99
7.9.2.4 TLS13_send()	100
7.9.2.5 TLS13_recv()	100
7.9.2.6 TLS13_clean()	100
7.10 tls_sal.h File Reference	101
7.10.1 Detailed Description	102
7.10.2 Function Documentation	102
7.10.2.1 SAL_name()	102
7.10.2.2 SAL_ciphers()	102
7.10.2.3 SAL_groups()	103
7.10.2.4 SAL_sigs()	103
7.10.2.5 SAL_sigCerts()	103
7.10.2.6 SAL_initLib()	105
7.10.2.7 SAL_hashType()	105
7.10.2.8 SAL_hashLen()	105
7.10.2.9 SAL_aeadKeylen()	106
7.10.2.10 SAL_aeadTaglen()	106
7.10.2.11 SAL_randomByte()	106
7.10.2.12 SAL_randomOctad()	107
7.10.2.13 SAL_hkdfExtract()	107

7.10.2.14 SAL_hkdfExpand()	107
7.10.2.15 SAL_hmac()	108
7.10.2.16 SAL_hashNull()	108
7.10.2.17 SAL_hashInit()	108
7.10.2.18 SAL_hashProcessArray()	109
7.10.2.19 SAL_hashOutput()	109
7.10.2.20 SAL_aeadEncrypt()	109
7.10.2.21 SAL_aeadDecrypt()	110
7.10.2.22 SAL_generateKeyPair()	110
7.10.2.23 SAL_generateSharedSecret()	111
7.10.2.24 SAL_tlsSignatureVerify()	111
7.10.2.25 SAL_tlsSignature()	112
7.11 <code>tls_sockets.h</code> File Reference	112
7.11.1 Detailed Description	113
7.11.2 Function Documentation	113
7.11.2.1 <code>setclientsock()</code>	113
7.11.2.2 <code>getIPAddress()</code>	114
7.11.2.3 <code>sendOctad()</code>	114
7.11.2.4 <code>sendLen()</code>	114
7.11.2.5 <code>getBytes()</code>	115
7.11.2.6 <code>getInt16()</code>	115
7.11.2.7 <code>getInt24()</code>	115
7.11.2.8 <code>getByte()</code>	116
7.11.2.9 <code>getOctad()</code>	116
7.12 <code>tls_tickets.h</code> File Reference	117
7.12.1 Detailed Description	117
7.12.2 Function Documentation	117
7.12.2.1 <code>millis()</code>	117
7.12.2.2 <code>parseTicket()</code>	117
7.12.2.3 <code>initTicketContext()</code>	118
7.12.2.4 <code>endTicketContext()</code>	118
7.12.2.5 <code>ticket_still_good()</code>	118
7.13 <code>tls_wifi.h</code> File Reference	119
7.13.1 Detailed Description	119
7.14 <code>tls_x509.h</code> File Reference	119
7.14.1 Detailed Description	120
7.14.2 Macro Definition Documentation	120
7.14.2.1 <code>X509_ECC</code>	120
7.14.2.2 <code>X509_RSA</code>	121
7.14.2.3 <code>X509_ECD</code>	121
7.14.2.4 <code>X509_PQ</code>	121
7.14.2.5 <code>X509_H256</code>	121

7.14.2.6 X509_H384	121
7.14.2.7 X509_H512	121
7.14.2.8 USE_NIST256	121
7.14.2.9 USE_C25519	121
7.14.2.10 USE_NIST384	122
7.14.2.11 USE_NIST521	122
7.14.3 Function Documentation	122
7.14.3.1 X509_extract_private_key()	122
7.14.3.2 X509_extract_cert_sig()	122
7.14.3.3 X509_extract_cert()	123
7.14.3.4 X509_extract_public_key()	123
7.14.3.5 X509_find_issuer()	123
7.14.3.6 X509_find_validity()	124
7.14.3.7 X509_find_subject()	124
7.14.3.8 X509_self_signed()	124
7.14.3.9 X509_find_entity_property()	125
7.14.3.10 X509_find_start_date()	125
7.14.3.11 X509_find_expiry_date()	125
7.14.3.12 X509_find_extensions()	126
7.14.3.13 X509_find_extension()	126
7.14.3.14 X509_find_alt_name()	127
7.14.4 Variable Documentation	127
7.14.4.1 X509_CN	127
7.14.4.2 X509_ON	127
7.14.4.3 X509_EN	127
7.14.4.4 X509_LN	127
7.14.4.5 X509_UN	128
7.14.4.6 X509_MN	128
7.14.4.7 X509_SN	128
7.14.4.8 X509_AN	128
7.14.4.9 X509_KU	128
7.14.4.10 X509_BC	128

Chapter 1

TII TLS C

TLS Client

1.1 Building

The TLS library is designed to support crypto agility by changing cryptographic providers. There are three cryptographic providers one can choose from.

1.1.1 Miracl

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

1.1.2 Miracl + LibSodium

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

1.1.3 Custom Crypto Library

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

To see the Security Abstraction Layer (SAL) capabilities

```
./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= B5B74CF6026CF16FA866BA7E7562C53F67A74949FF040319B0BD2149CF4EF97CAD482463F1746D2C
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= 7DE3D4B470FBCA72FEECBAA1B938F4AF85F0E4D84C8E06E4218A92DF3EE67CF
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= 3CC796B38A7FEB226D9B0CD6B6BB499425329DDF9FF43060C5C30834D75EE79
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= 536F912C98CF4C2D9672DEA57AC8136519607014EFE8BBA289FCED97929EA9633
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. At this stage the tool is still somewhat fragile, and would be expected to sometimes fail. 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)

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

1.1.4 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
```

A way to test less popular 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 -key key.pem -cert cert.pem -accept 4433 -www
```

acts as a normal Website, while

```
openssl s_server -tls1_3 -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
```

1.1.5 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
```

Chapter 2

Description

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

2.1 Linux Installation

Copy the repository to your local machine. Move to sal/miracl/ and download into it all files for the C++ version of MIRACL core from (<https://github.com/miracl/core/cpp>). Build the MIRACL core library by executing

```
python3 config64.py
```

selecting support for C25519, NIST256, NIST384, RSA2048 and RSA4096.

This library provides the default SAL (Security Abstraction Layer), does all the crypto, and can be regarded as a "placeholder" as we may in the future replace its functionality from other sources. Make sure to always use the latest version of this library - as the requirements of this project unfold, some minor updates will be required.

If desired edit the file [include/tls1_3.h](#) to set some project constants. For example you may want to set the verbosity of the output to IO_DEBUG.

Build the tiits library and the client app

```
cmake -DSAL=MIRACL .  
make
```

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

```
cmake -DSAL=MIRACL_SODIUM .  
make
```

To use a SAL which use functions from the tii-crypto library, move to sal/tii-cryptolib/ and clone the tii-cryptolib library there. Build a TLS friendly version of the library following the Opt. 1 Build instructions

```
cmake -DCMAKE_BUILD_TYPE=Release -DCURVE=NIST_P256 -Bcmake-build
cd cmake-build
make
```

Then build the tiits library and the client app

```
cmake -DSAL=MIRACL_TIILIB .
make
```

To see the SAL capabilities

```
./client -s
```

To connect to a Website

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

The output should (if IO_DEBUG has been selected) 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= 093388E25C3F8468DF3A0544683036CBACF5157874CE995C080807559834BCA
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 0A5C155DBDD9F7F6ABE005D351D6E3FF9DEBA799F7479BD33E1C784B63CB4CA695A76815C9B666C24B6E989EE85009A0
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 = 0A5C155DBDD9F7F6ABE005D351D6E3FF9DEBA799F7479BD33E1C784B63CB4CA695A76815C9B666C24B6E989EE85009A0
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), attempt another connection to the same site

```
./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= 0401D908F018811AF140E2D417EB2713492C146C2B73F78A81DEC6C3F6E2A31D5114207D93EC92AEB03D64DAD1

```

```

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= 536F912C98CF4C2D9672DEA57AC8136519607014EFEBA289FCED97929EA9633
Server application traffic secret= 6B797DBC7FB2D9F75A877F1D34EE7CACC6D65C847C085331F8941C81F2884E83
Resumption Handshake concluded
Early data was accepted
Waiting for Server input
Receiving application data (truncated HTML) = 485454502F312E3120323030204F4B0D0A5365727665723A205377696674544C
Alert sent to Server - Close notify
Connection closed

```

Try it out on your favourite websites. It will abort if TLS1.3 is not supported. At this stage the tool is still somewhat fragile, and would be expected to sometimes fail. 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)

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

2.1.1 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
```

A way to test less popular 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 -key key.pem -cert cert.pem -accept 4433 -www
```

acts as a normal Website, while

```
openssl s_server -tls1_3 -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!

2.1.2 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
```

2.2 Building the client application on an Arduino board (here Arduino Nano 33 IoT)

1. Create working directory directory with name tiits
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 code from the lib/, 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_sockets.h](#) to define TLS_ARDUINO. Optionally set VERBOSITY in [tls1_3.h](#) to IO_DEBUG.
8. Edit the file client.cpp to set your wifi SSID and password (near line 45)
9. Run py config.py, and select options 2,8,41 and 43. This creates the SAL (in this case using miracl + ECC608A hardware).
10. Drop the working directory into where the Arduino IDE expects it.
11. (In the IDE select File->Preferences and find the Sketchbook location - its the libraries directory off that.)
12. Open the Arduino app, and look in File->Examples->tiits, and look for the example "client"
13. Upload to the board and run it! Tools->Serial Monitor to see the output

Chapter 3

Configure the Arduino Nano RP2040

This build is specifically for the Arduino Nano version of the Raspberry Pi Pico (RP2040)

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.

3.1 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 code from the `lib/`, `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_sockets.h](#) to define `TLS_ARDUINO` (line 13). Optionally define `VERBOSITY` in [tls1_3.h](#) as `IO_DEBUG`.
8. Edit the file `client.cpp` to set your wifi SSID and password (near line 62)
9. Run `py config.py`, and select options 2, 8, 41 and 43. This creates the SAL (in this case using `miracl` + ECC608A hardware).
10. Drop the working directory into where the Arduino IDE expects it.

11. (In the IDE select File->Preferences and find the Sketchbook location - its the libraries directory off that.)
12. Open the Arduino app, and look in File->Examples->tiitls, and look for the example "client"
13. Upload to the board and run it. Open Tools->Serial Monitor to see the output.
14. Enter URL (e.g. www.bbc.co.uk) when prompted, and press return. A full TLS1.3 handshake followed by a resumption is attempted.
15. Click on Clear Output and Send to repeat for a different URL (or click Send again to see SAL capabilities).

or before executing step 9, search for `$*$$$*$` in `config.py` and make change as indicated. Copy `x25519.S` from <https://github.com/pornin/x25519-cm0/blob/main/src/x25519-cm0.S> into working directory. Replace step 9 with

9a. Run `py config.py`, and select options 8, 41 and 43. This creates the SAL (in this case using `miracl` + `ECC608A` hardware + Pornin's `x25519`).

Chapter 4

Data Structure Index

4.1 Data Structures

Here are the data structures with brief descriptions:

crypto	Crypto context structure	17
ECCX08Class	19
ee_status	Server encrypted extensions expectations/responses	19
octad	Safe representation of an octad	20
pktype	Public key type	21
ret	Function return structure	22
Socket	Socket instance	23
ticket	Ticket context structure	24
TLS_session	TLS1.3 session state	26
unihash	Universal Hash Function	30

Chapter 5

File Index

5.1 File List

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

ECCX08.h	??
tls1_3.h	
Main TLS 1.3 Header File for constants and structures	31
tls_cert_chain.h	
Process Certificate Chain	56
tls_certs.h	
Certificate Authority root certificate store	57
tls_client_recv.h	
Process Input received from the Server	58
tls_client_send.h	
Process Output to be sent to the Server	67
tls_keys_calc.h	
TLS 1.3 crypto support functions	77
tls_logger.h	
TLS 1.3 logging	87
tls_octads.h	
Octad handling routines - octads don't overflow, they truncate	91
tls_protocol.h	
TLS 1.3 main client-side protocol functions	98
tls_sal.h	
Security Abstraction Layer for TLS	101
tls_sockets.h	
Set up sockets for reading and writing	112
tls_tickets.h	
TLS 1.3 process resumption tickets	117
tls_wifi.h	
Define Socket structure depending on processor context	119
tls_x509.h	
X509 function Header File	119

Chapter 6

Data Structure Documentation

6.1 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](#)

6.1.1 Detailed Description

crypto context structure

6.1.2 Field Documentation

6.1.2.1 active

```
bool crypto::active
```

Indicates if encryption has been activated

6.1.2.2 k

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

AEAD cryptographic Key bytes

6.1.2.3 iv

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

AEAD cryptographic IV bytes

6.1.2.4 K

```
octad crypto::K
```

Key as octad

6.1.2.5 IV

```
octad crypto::IV
```

IV as octad

6.1.2.6 record

```
unsign32 crypto::record
```

current record number - to be incremented

6.1.2.7 suite

```
int crypto::suite
```

Cipher Suite

6.1.2.8 taglen

```
int crypto::taglen
```

Tag Length

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

- [tls1_3.h](#)

6.2 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

6.3 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**

6.3.1 Detailed Description

server encrypted extensions expectations/responses

6.3.2 Field Documentation

6.3.2.1 early_data

```
bool ee_status::early_data
```

true if early data accepted

6.3.2.2 alpn

```
bool ee_status::alpn
```

true if ALPN accepted

6.3.2.3 server_name

```
bool ee_status::server_name
```

true if server name accepted

6.3.2.4 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](#)

6.4 octad Struct Reference

Safe representation of an octad.

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

Data Fields

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

6.4.1 Detailed Description

Safe representation of an octad.

6.4.2 Field Documentation

6.4.2.1 len

```
int octad::len
```

length in bytes

6.4.2.2 max

```
int octad::max
```

max length allowed - enforce truncation

6.4.2.3 val

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

byte array

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

- [tls_octads.h](#)

6.5 pktype Struct Reference

Public key type.

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

Data Fields

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

6.5.1 Detailed Description

Public key type.

6.5.2 Field Documentation

6.5.2.1 type

```
int pktype::type
```

signature type (ECC or RSA)

6.5.2.2 hash

```
int pktype::hash
```

hash type

6.5.2.3 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](#)

6.6 ret Struct Reference

function return structure

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

Data Fields

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

6.6.1 Detailed Description

function return structure

6.6.2 Field Documentation

6.6.2.1 val

```
unsign32 ret::val
```

return value

6.6.2.2 err

```
int ret::err
```

error return

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

- [tls1_3.h](#)

6.7 Socket Class Reference

[Socket](#) instance.

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

Public Member Functions

- bool **connect** (char *host, int port)
- 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](#) ()

6.7.1 Detailed Description

[Socket](#) instance.

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

- [tls_sockets.h](#)

6.8 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](#)

6.8.1 Detailed Description

ticket context structure

6.8.2 Field Documentation

6.8.2.1 valid

```
bool ticket::valid
```

Is ticket valid?

6.8.2.2 tick

```
char ticket::tick[TLS_MAX_TICKET_SIZE]
```

Ticket bytes

6.8.2.3 nonce

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

nonce

6.8.2.4 psk

```
char ticket::psk[TLS_MAX_HASH]
```

pre-shared key

6.8.2.5 TICK

```
octad ticket::TICK
```

Ticket or external PSK label as octad

6.8.2.6 NONCE

```
octad ticket::NONCE
```

Nonce as octad

6.8.2.7 PSK

```
octad ticket::PSK
```

PSK as octad

6.8.2.8 age_obfuscator

```
unsign32 ticket::age_obfuscator
```

ticket age obfuscator - 0 for external PSK

6.8.2.9 max_early_data

```
uint32 ticket::max_early_data
```

Maximum early data allowed for this ticket

6.8.2.10 birth

```
uint32 ticket::birth
```

Birth time of this ticket

6.8.2.11 lifetime

```
int ticket::lifetime
```

ticket lifetime

6.8.2.12 cipher_suite

```
int ticket::cipher_suite
```

Cipher suite used

6.8.2.13 favourite_group

```
int ticket::favourite_group
```

the server's favourite group

6.8.2.14 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](#)

6.9 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](#)
- [crypto](#) K_send
- [crypto](#) K_recv
- [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](#) IO
- char [io](#) [TLS_MAX_IO_SIZE]
- int [ptr](#)
- [unihash](#) [tlshash](#)
- [ticket](#) T

6.9.1 Detailed Description

TLS1.3 session state.

6.9.2 Field Documentation

6.9.2.1 status

```
int TLS_session::status
```

Connection status

6.9.2.2 max_record

```
int TLS_session::max_record
```

max record size I should send

6.9.2.3 sockptr

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

Pointer to socket

6.9.2.4 id

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

Session ID

6.9.2.5 hostname

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

Server name for connection

6.9.2.6 cipher_suite

```
int TLS_session::cipher_suite
```

agreed cipher suite

6.9.2.7 favourite_group

```
int TLS_session::favourite_group
```

favourite key exchange group - may be changed on handshake retry

6.9.2.8 K_send

```
crypto TLS_session::K_send
```

Sending Key

6.9.2.9 K_rcv

```
crypto TLS_session::K_rcv
```

Receiving Key

6.9.2.10 HS

```
octad TLS_session::HS
```

Handshake secret

6.9.2.11 hs

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

Handshake secret data

6.9.2.12 RMS

```
octad TLS_session::RMS
```

Resumption Master Secret

6.9.2.13 rms

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

Resumption Master Secret data

6.9.2.14 STS

```
octad TLS_session::STS
```

Server Traffic secret

6.9.2.15 sts

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

Server Traffic secret data

6.9.2.16 CTS

```
octad TLS_session::CTS
```

Client Traffic secret

6.9.2.17 cts

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

Client Traffic secret data

6.9.2.18 IO

```
octad TLS_session::IO
```

Main IO buffer for this connection

6.9.2.19 io

```
char TLS_session::io[TLS_MAX_IO_SIZE]
```

Byte array for main IO buffer for this connection

6.9.2.20 ptr

```
int TLS_session::ptr
```

pointer into IO buffer

6.9.2.21 tlshash

```
unihash TLS_session::tlshash
```

Transcript hash recorder

6.9.2.22 T

```
ticket TLS_session::T
```

resumption ticket

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

- [tls1_3.h](#)

6.10 unihash Struct Reference

Universal Hash Function.

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

Data Fields

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

6.10.1 Detailed Description

Universal Hash Function.

6.10.2 Field Documentation

6.10.2.1 state

```
char unihash::state[TLS\_MAX\_HASH\_STATE]
```

hash function state

6.10.2.2 htype

```
int unihash::htype
```

The hash type (typically SHA256)

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

- [tls1_3.h](#)

Chapter 7

File Documentation

7.1 `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 `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 2`

- `#define NOCERT 0`
- `#define RSA_SS 1`
- `#define ECC_SS 2`
- `#define DLT_SS 3`
- `#define HW_1 4`
- `#define HW_2 5`
- `#define VERBOSITY IO_PROTOCOL`
- `#define THIS_YEAR 2022`
- `#define CLIENT_CERT ECC_SS`
- `#define TLS_APPLICATION_PROTOCOL (char *)("http/1.1")`
- `#define ALLOW_SELF_SIGNED`
- `#define CRYPTO_SETTING TYPICAL`
- `#define TRY_EARLY_DATA`
- `#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 4`
- `#define TLS_MAX_IO_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 512`
- `#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 512`
- `#define TLS_MAX_EXTENSIONS 2048`
- `#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_SERVER_NAME 128`
- `#define TLS_MAX_SUPPORTED_GROUPS 5`
- `#define TLS_MAX_SUPPORTED_SIGS 16`
- `#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 KYBER768 0x4242
- #define ECDSA_SECP256R1_SHA256 0x0403
- #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 DILITHIUM3 0x0903
- #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 SERVER_NAME 0x0000
- #define SUPPORTED_GROUPS 0x000a
- #define SIG_ALGS 0x000d
- #define SIG_ALGS_CERT 0x0032
- #define KEY_SHARE 0x0033
- #define PSK_MODE 0x002d
- #define PRESARED_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 HSHAKE 0x16
- #define APPLICATION 0x17
- #define ALERT 0x15
- #define CHANGE_CIPHER 0x14
- #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 BAD_MESSAGE -23`
- `#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 CLOSE_NOTIFY 0x00`
- `#define LOG_OUTPUT_TRUNCATION 256`
- `#define TLS13_DISCONNECTED 0`
- `#define TLS13_CONNECTED 1`
- `#define TLS_FAILURE 0`
- `#define TLS_SUCCESS 1`
- `#define TLS_RESUMPTION_REQUIRED 2`
- `#define TLS_EARLY_DATA_ACCEPTED 3`

Typedefs

- `typedef uint8_t byte`
- `typedef int8_t sign8`
- `typedef int16_t sign16`
- `typedef int32_t sign32`
- `typedef int64_t sign64`
- `typedef uint32_t unsign32`
- `typedef uint64_t unsign64`

7.1.1 Detailed Description

Main TLS 1.3 Header File for constants and structures.

Author

Mike Scott

7.1.2 Macro Definition Documentation

7.1.2.1 IO_NONE

```
#define IO_NONE 0
```

Run silently

7.1.2.2 IO_APPLICATION

```
#define IO_APPLICATION 1
```

just print application traffic

7.1.2.3 IO_PROTOCOL

```
#define IO_PROTOCOL 2
```

print protocol progress + application traffic

7.1.2.4 IO_DEBUG

```
#define IO_DEBUG 3
```

print lots of debug information + protocol progress + application progress

7.1.2.5 IO_WIRE

```
#define IO_WIRE 4
```

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

7.1.2.6 TINY_ECC

```
#define TINY_ECC 0
```

ECC keys only

7.1.2.7 TYPICAL

```
#define TYPICAL 1
```

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

7.1.2.8 POST_QUANTUM

```
#define POST_QUANTUM 2
```

Post quantum (Dilithium+Kyber?)

7.1.2.9 NOCERT

```
#define NOCERT 0
```

Don't have a Client Cert

7.1.2.10 RSA_SS

```
#define RSA_SS 1
```

self signed RSA cert

7.1.2.11 ECC_SS

```
#define ECC_SS 2
```

self signed ECC cert

7.1.2.12 DLT_SS

```
#define DLT_SS 3
```

self signed Dilithium cert

7.1.2.13 HW_1

```
#define HW_1 4
```

RP2040 1 Hardware cert

7.1.2.14 HW_2

```
#define HW_2 5
```

RP2040 2 Hardware cert

7.1.2.15 VERBOSITY

```
#define VERBOSITY IO\_PROTOCOL
```

Set to level of output information desired - see above

7.1.2.16 THIS_YEAR

```
#define THIS_YEAR 2022
```

Set to this year - crudely used to deprecate old certificates

7.1.2.17 CLIENT_CERT

```
#define CLIENT_CERT ECC_SS
```

Indicate capability of authenticating with a cert plus signing key

7.1.2.18 TLS_APPLICATION_PROTOCOL

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

Support ALPN protocol

7.1.2.19 ALLOW_SELF_SIGNED

```
#define ALLOW_SELF_SIGNED
```

allow self-signed server cert

7.1.2.20 CRYPTO_SETTING

```
#define CRYPTO_SETTING TYPICAL
```

Determine Cryptography settings

7.1.2.21 TRY_EARLY_DATA

```
#define TRY_EARLY_DATA
```

Try to send early data on resumptions

7.1.2.22 TLS_SHA256_T

```
#define TLS_SHA256_T 1
```

SHA256 hash

7.1.2.23 TLS_SHA384_T

```
#define TLS_SHA384_T 2
```

SHA384 hash

7.1.2.24 TLS_SHA512_T

```
#define TLS_SHA512_T 3
```

SHA512 hash

7.1.2.25 TLS_MAX_HASH_STATE

```
#define TLS_MAX_HASH_STATE 768
```

Maximum memory required to store hash function state

7.1.2.26 TLS_MAX_HASH

```
#define TLS_MAX_HASH 64
```

Maximum hash output length in bytes

7.1.2.27 TLS_MAX_KEY

```
#define TLS_MAX_KEY 32
```

Maximum key length in bytes

7.1.2.28 TLS_X509_MAX_FIELD

```
#define TLS_X509_MAX_FIELD 256
```

Maximum X.509 field size

7.1.2.29 TLS_MAX_EXT_LABEL

```
#define TLS_MAX_EXT_LABEL 256
```

Max external psk label size

7.1.2.30 TLS_MAX_FRAG

```
#define TLS_MAX_FRAG 4
```

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

7.1.2.31 TLS_MAX_IO_SIZE

```
#define TLS_MAX_IO_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

7.1.2.32 TLS_MAX_PLAIN_FRAG

```
#define TLS_MAX_PLAIN_FRAG 16384
```

Max Plaintext Fragment size

7.1.2.33 TLS_MAX_CIPHER_FRAG

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

Max Ciphertext Fragment size

7.1.2.34 TLS_MAX_CERT_SIZE

```
#define TLS_MAX_CERT_SIZE 2048
```

I checked - current max for root CAs is 2016

7.1.2.35 TLS_MAX_CERT_B64

```
#define TLS_MAX_CERT_B64 2800
```

In base64 - current max for root CAs is 2688

7.1.2.36 TLS_MAX_HELLO

```
#define TLS_MAX_HELLO 1024
```

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

7.1.2.37 TLS_MAX_SIG_PUB_KEY_SIZE

```
#define TLS_MAX_SIG_PUB_KEY_SIZE 512
```

Max signature public key size in bytes RSA

7.1.2.38 TLS_MAX_SIG_SECRET_KEY_SIZE

```
#define TLS_MAX_SIG_SECRET_KEY_SIZE 512
```

Max signature private key size in bytes RSA

7.1.2.39 TLS_MAX_SIGNATURE_SIZE

```
#define TLS_MAX_SIGNATURE_SIZE 512
```

Max digital signature size in bytes RSA

7.1.2.40 TLS_MAX_KEX_PUB_KEY_SIZE

```
#define TLS_MAX_KEX_PUB_KEY_SIZE 97
```

Max key exchange public key size in bytes ECC

7.1.2.41 TLS_MAX_KEX_CIPHERTEXT_SIZE

```
#define TLS_MAX_KEX_CIPHERTEXT_SIZE 97
```

Max key exchange (KEM) ciphertext size ECC

7.1.2.42 TLS_MAX_KEX_SECRET_KEY_SIZE

```
#define TLS_MAX_KEX_SECRET_KEY_SIZE 48
```

Max key exchange private key size in bytes ECC

7.1.2.43 TLS_MAX_SERVER_CHAIN_LEN

```
#define TLS_MAX_SERVER_CHAIN_LEN 2
```

Maximum Server Certificate chain length - omitting root CA

7.1.2.44 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

7.1.2.45 TLS_MAX_CLIENT_CHAIN_LEN

```
#define TLS_MAX_CLIENT_CHAIN_LEN 1
```

Maximum Client Certificate chain length - one self signed here

7.1.2.46 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

7.1.2.47 TLS_MAX_SHARED_SECRET_SIZE

```
#define TLS_MAX_SHARED_SECRET_SIZE 256
```

Max key exchange Shared secret size

7.1.2.48 TLS_MAX_TICKET_SIZE

```
#define TLS_MAX_TICKET_SIZE 512
```

maximum resumption ticket size

7.1.2.49 TLS_MAX_EXTENSIONS

```
#define TLS_MAX_EXTENSIONS 2048
```

Max extensions size

7.1.2.50 TLS_MAX_ECC_FIELD

```
#define TLS_MAX_ECC_FIELD 66
```

Max ECC field size in bytes

7.1.2.51 TLS_MAX_IV_SIZE

```
#define TLS_MAX_IV_SIZE 12
```

Max IV size in bytes

7.1.2.52 TLS_MAX_TAG_SIZE

```
#define TLS_MAX_TAG_SIZE 16
```

Max HMAC tag length in bytes

7.1.2.53 TLS_MAX_COOKIE

```
#define TLS_MAX_COOKIE 128
```

Max Cookie size

7.1.2.54 TLS_MAX_SERVER_NAME

```
#define TLS_MAX_SERVER_NAME 128
```

Max server name size in bytes

7.1.2.55 TLS_MAX_SUPPORTED_GROUPS

```
#define TLS_MAX_SUPPORTED_GROUPS 5
```

Max number of supported crypto groups

7.1.2.56 TLS_MAX_SUPPORTED_SIGS

```
#define TLS_MAX_SUPPORTED_SIGS 16
```

Max number of supported signature schemes

7.1.2.57 TLS_MAX_PSK_MODES

```
#define TLS_MAX_PSK_MODES 2
```

Max preshared key modes

7.1.2.58 TLS_MAX_CIPHER_SUITES

```
#define TLS_MAX_CIPHER_SUITES 5
```

Max number of supported cipher suites

7.1.2.59 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

7.1.2.60 TLS_AES_256_GCM_SHA384

```
#define TLS_AES_256_GCM_SHA384 0x1302
```

AES256/SHA384/GCM cipher suite

7.1.2.61 TLS_CHACHA20_POLY1305_SHA256

```
#define TLS_CHACHA20_POLY1305_SHA256 0x1303
```

CHACHA20/SHA256/POLY1305 cipher suite

7.1.2.62 TLS_AES_128_CCM_SHA256

```
#define TLS_AES_128_CCM_SHA256 0x1304
```

AES/SHA256/CCM cipher suite - optional

7.1.2.63 TLS_AES_128_CCM_8_SHA256

```
#define TLS_AES_128_CCM_8_SHA256 0x1305
```

AES/SHA256/CCM 8 cipher suite - optional

7.1.2.64 X25519

```
#define X25519 0x001d
```

X25519 elliptic curve key exchange

7.1.2.65 SECP256R1

```
#define SECP256R1 0x0017
```

NIST SECP256R1 elliptic curve key exchange

7.1.2.66 SECP384R1

```
#define SECP384R1 0x0018
```

NIST SECP384R1 elliptic curve key exchange

7.1.2.67 SECP521R1

```
#define SECP521R1 0x0019
```

NIST SECP521R1 elliptic curve key exchange

7.1.2.68 X448

```
#define X448 0x001e
```

X448 elliptic curve key exchange

7.1.2.69 KYBER768

```
#define KYBER768 0x4242
```

Kyber PQ key exchange

7.1.2.70 ECDSA_SECP256R1_SHA256

```
#define ECDSA_SECP256R1_SHA256 0x0403
```

Supported ECDSA Signature algorithm

7.1.2.71 ECDSA_SECP384R1_SHA384

```
#define ECDSA_SECP384R1_SHA384 0x0503
```

Supported ECDSA Signature algorithm

7.1.2.72 RSA_PSS_RSAE_SHA256

```
#define RSA_PSS_RSAE_SHA256 0x0804
```

Supported RSA Signature algorithm

7.1.2.73 RSA_PSS_RSAE_SHA384

```
#define RSA_PSS_RSAE_SHA384 0x0805
```

Supported RSA Signature algorithm

7.1.2.74 RSA_PSS_RSAE_SHA512

```
#define RSA_PSS_RSAE_SHA512 0x0806
```

Supported RSA Signature algorithm

7.1.2.75 RSA_PKCS1_SHA256

```
#define RSA_PKCS1_SHA256 0x0401
```

Supported RSA Signature algorithm

7.1.2.76 RSA_PKCS1_SHA384

```
#define RSA_PKCS1_SHA384 0x0501
```

Supported RSA Signature algorithm

7.1.2.77 RSA_PKCS1_SHA512

```
#define RSA_PKCS1_SHA512 0x0601
```

Supported RSA Signature algorithm

7.1.2.78 ED25519

```
#define ED25519 0x0807
```

Ed25519 EdDSA Signature algorithm

7.1.2.79 DILITHIUM3

```
#define DILITHIUM3 0x0903
```

Dilithium3 Signature algorithm

7.1.2.80 PSKOK

```
#define PSKOK 0x00
```

Preshared Key only mode

7.1.2.81 PSKWECDHE

```
#define PSKWECDHE 0x01
```

Preshared Key with Diffie-Hellman key exchange mode

7.1.2.82 TLS_FULL_HANDSHAKE

```
#define TLS_FULL_HANDSHAKE 1
```

Came from Full Handshake

7.1.2.83 TLS_EXTERNAL_PSK

```
#define TLS_EXTERNAL_PSK 2
```

External Pre-Shared Key

7.1.2.84 TLS1_0

```
#define TLS1_0 0x0301
```

TLS 1.0 version

7.1.2.85 TLS1_2

```
#define TLS1_2 0x0303
```

TLS 1.2 version

7.1.2.86 TLS1_3

```
#define TLS1_3 0x0304
```

TLS 1.3 version

7.1.2.87 SERVER_NAME

```
#define SERVER_NAME 0x0000
```

Server Name extension

7.1.2.88 SUPPORTED_GROUPS

```
#define SUPPORTED_GROUPS 0x000a
```

Supported Group extension

7.1.2.89 SIG_ALGS

```
#define SIG_ALGS 0x000d
```

Signature algorithms extension

7.1.2.90 SIG_ALGS_CERT

```
#define SIG_ALGS_CERT 0x0032
```

Signature algorithms Certificate extension

7.1.2.91 KEY_SHARE

```
#define KEY_SHARE 0x0033
```

Key Share extension

7.1.2.92 PSK_MODE

```
#define PSK_MODE 0x002d
```

Preshared key mode extension

7.1.2.93 PRESHARED_KEY

```
#define PRESHARED_KEY 0x0029
```

Preshared key extension

7.1.2.94 TLS_VER

```
#define TLS_VER 0x002b
```

TLS version extension

7.1.2.95 COOKIE

```
#define COOKIE 0x002c
```

Cookie extension

7.1.2.96 EARLY_DATA

```
#define EARLY_DATA 0x002a
```

Early Data extension

7.1.2.97 MAX_FRAG_LENGTH

```
#define MAX_FRAG_LENGTH 0x0001
```

max fragmentation length extension

7.1.2.98 PADDING

```
#define PADDING 0x0015
```

Padding extension

7.1.2.99 APP_PROTOCOL

```
#define APP_PROTOCOL 0x0010
```

Application Layer Protocol Negotiation (ALPN)

7.1.2.100 RECORD_SIZE_LIMIT

```
#define RECORD_SIZE_LIMIT 0x001c
```

Record Size Limit

7.1.2.101 HSHAKE

```
#define HSHAKE 0x16
```

Handshake record

7.1.2.102 APPLICATION

```
#define APPLICATION 0x17
```

Application record

7.1.2.103 ALERT

```
#define ALERT 0x15
```

Alert record

7.1.2.104 CHANGE_CIPHER

```
#define CHANGE_CIPHER 0x14
```

Change Cipher record

7.1.2.105 TIMED_OUT

```
#define TIMED_OUT 0x01
```

Time-out

7.1.2.106 CLIENT_HELLO

```
#define CLIENT_HELLO 0x01
```

Client Hello message

7.1.2.107 SERVER_HELLO

```
#define SERVER_HELLO 0x02
```

Server Hello message

7.1.2.108 CERTIFICATE

```
#define CERTIFICATE 0x0b
```

Certificate message

7.1.2.109 CERT_REQUEST

```
#define CERT_REQUEST 0x0d
```

Certificate Request

7.1.2.110 CERT_VERIFY

```
#define CERT_VERIFY 0x0f
```

Certificate Verify message

7.1.2.111 FINISHED

```
#define FINISHED 0x14
```

Handshake Finished message

7.1.2.112 ENCRYPTED_EXTENSIONS

```
#define ENCRYPTED_EXTENSIONS 0x08
```

Encrypted Extensions message

7.1.2.113 TICKET

```
#define TICKET 0x04
```

Ticket message

7.1.2.114 KEY_UPDATE

```
#define KEY_UPDATE 0x18
```

Key Update message

7.1.2.115 MESSAGE_HASH

```
#define MESSAGE_HASH 0xFE
```

Special synthetic message hash message

7.1.2.116 END_OF_EARLY_DATA

```
#define END_OF_EARLY_DATA 0x05
```

End of Early Data message

7.1.2.117 HANDSHAKE_RETRY

```
#define HANDSHAKE_RETRY 0x102
```

Handshake retry

7.1.2.118 NOT_TLS1_3

```
#define NOT_TLS1_3 -2
```

Wrong version error, not TLS1.3

7.1.2.119 BAD_CERT_CHAIN

```
#define BAD_CERT_CHAIN -3
```

Bad Certificate Chain error

7.1.2.120 ID_MISMATCH

```
#define ID_MISMATCH -4
```

Session ID mismatch error

7.1.2.121 UNRECOGNIZED_EXT

```
#define UNRECOGNIZED_EXT -5
```

Unrecognised extension error

7.1.2.122 BAD_HELLO

```
#define BAD_HELLO -6
```

badly formed Hello message error

7.1.2.123 WRONG_MESSAGE

```
#define WRONG_MESSAGE -7
```

Message out-of-order error

7.1.2.124 MISSING_REQUEST_CONTEXT

```
#define MISSING_REQUEST_CONTEXT -8
```

Request context missing error

7.1.2.125 AUTHENTICATION_FAILURE

```
#define AUTHENTICATION_FAILURE -9
```

Authentication error - AEAD Tag incorrect

7.1.2.126 BAD_RECORD

```
#define BAD_RECORD -10
```

Badly formed Record received

7.1.2.127 BAD_TICKET

```
#define BAD_TICKET -11
```

Badly formed Ticket received

7.1.2.128 NOT_EXPECTED

```
#define NOT_EXPECTED -12
```

Received ack for something not requested

7.1.2.129 CA_NOT_FOUND

```
#define CA_NOT_FOUND -13
```

Certificate Authority not found

7.1.2.130 CERT_OUTOFDATE

```
#define CERT_OUTOFDATE -14
```

Certificate Expired

7.1.2.131 MEM_OVERFLOW

```
#define MEM_OVERFLOW -15
```

Memory Overflow

7.1.2.132 FORBIDDEN_EXTENSION

```
#define FORBIDDEN_EXTENSION -16
```

Forbidden Encrypted Extension

7.1.2.133 MAX_EXCEEDED

```
#define MAX_EXCEEDED -17
```

Maximum record size exceeded

7.1.2.134 EMPTY_CERT_CHAIN

```
#define EMPTY_CERT_CHAIN -18
```

Empty Certificate Message

7.1.2.135 SELF_SIGNED_CERT

```
#define SELF_SIGNED_CERT -20
```

Self signed certificate

7.1.2.136 BAD_MESSAGE

```
#define BAD_MESSAGE -23
```

Badly formed message

7.1.2.137 ILLEGAL_PARAMETER

```
#define ILLEGAL_PARAMETER 0x2F
```

Illegal parameter alert

7.1.2.138 UNEXPECTED_MESSAGE

```
#define UNEXPECTED_MESSAGE 0x0A
```

Unexpected message alert

7.1.2.139 DECRYPT_ERROR

```
#define DECRYPT_ERROR 0x33
```

Decryption error alert

7.1.2.140 BAD_CERTIFICATE

```
#define BAD_CERTIFICATE 0x2A
```

Bad certificate alert

7.1.2.141 UNSUPPORTED_EXTENSION

```
#define UNSUPPORTED_EXTENSION 0x6E
```

Unsupported extension alert

7.1.2.142 UNKNOWN_CA

```
#define UNKNOWN_CA 0x30
```

Unrecognised Certificate Authority

7.1.2.143 CERTIFICATE_EXPIRED

```
#define CERTIFICATE_EXPIRED 0x2D
```

Certificate Expired

7.1.2.144 PROTOCOL_VERSION

```
#define PROTOCOL_VERSION 0x46
```

Wrong TLS version

7.1.2.145 DECODE_ERROR

```
#define DECODE_ERROR 0x32
```

Decode error alert

7.1.2.146 RECORD_OVERFLOW

```
#define RECORD_OVERFLOW 0x16
```

Record Overflow

7.1.2.147 CLOSE_NOTIFY

```
#define CLOSE_NOTIFY 0x00
```

Orderly shut down of connection

7.1.2.148 LOG_OUTPUT_TRUNCATION

```
#define LOG_OUTPUT_TRUNCATION 256
```

Output Hex digits before truncation

7.1.2.149 TLS13_DISCONNECTED

```
#define TLS13_DISCONNECTED 0
```

TLS1.3 Connection is broken

7.1.2.150 TLS13_CONNECTED

```
#define TLS13_CONNECTED 1
```

TLS1.3 Connection is made

7.1.2.151 TLS_FAILURE

```
#define TLS_FAILURE 0
```

Failed to cmake TLS1.3 connection

7.1.2.152 TLS_SUCCESS

```
#define TLS_SUCCESS 1
```

Succeeded in making TLS1.3 connection

7.1.2.153 TLS_RESUMPTION_REQUIRED

```
#define TLS_RESUMPTION_REQUIRED 2
```

Connection succeeded, but handshake retry was needed

7.1.2.154 TLS_EARLY_DATA_ACCEPTED

```
#define TLS_EARLY_DATA_ACCEPTED 3
```

Connection succeeded, and early data was accepted

7.1.3 Typedef Documentation

7.1.3.1 byte

```
typedef uint8_t byte
```

8-bit unsigned integer

7.1.3.2 sign8

```
typedef int8_t sign8
```

8-bit signed integer

7.1.3.3 sign16

```
typedef int16_t sign16
```

16-bit signed integer

7.1.3.4 sign32

```
typedef int32_t sign32
```

32-bit signed integer

7.1.3.5 sign64

```
typedef int64_t sign64
```

64-bit signed integer

7.1.3.6 unsign32

```
typedef uint32_t unsign32
```

32-bit unsigned integer

7.1.3.7 unsign64

```
typedef uint64_t unsign64
```

64-bit unsigned integer

7.2 `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, `octad` *PUBKEY, `octad` *SIG)
Check Certificate Chain for hostname, and extract public key.
- int `getClientPrivateKeyandCertChain` (int nccsalgs, int *csigAlgs, `octad` *PRIVKEY, `octad` *CERTCHAIN)
Get Client private key and Certificate chain from .pem files.

7.2.1 Detailed Description

Process Certificate Chain.

Author

Mike Scott

7.2.2 Function Documentation

7.2.2.1 `checkServerCertChain()`

```
int checkServerCertChain (
    octad * CERTCHAIN,
    char * hostname,
    octad * PUBKEY,
    octad * SIG )
```

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>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

7.2.2.2 `getClientPrivateKeyandCertChain()`

```
int getClientPrivateKeyandCertChain (
    int nccsalgs,
    int * csigAlgs,
    octad * PRIVKEY,
    octad * CERTCHAIN )
```

Get Client private key and Certificate chain from .pem files.

Parameters

<i>nccsalgs</i>	the number of acceptable signature algorithms
<i>csigAlgs</i>	acceptable signature algorithms
<i>PRIVKEY</i>	the Client's private key
<i>CERTCHAIN</i>	the Client's certificate chain

Returns

type of private key, ECC or RSA

7.3 `tls_certs.h` File Reference

Certificate Authority root certificate store.

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

Variables

- const char * `myprivate` =NULL
- const char * `mycert`
- const char * `cacerts`

7.3.1 Detailed Description

Certificate Authority root certificate store.

Author

Mike Scott

7.3.2 Variable Documentation

7.3.2.1 myprivate

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

Client private key

7.3.2.2 mycert

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

Initial value:

```
=(char *)
"-----BEGIN CERTIFICATE-----\n"
"MIIBKzCB0aADAgECAgEBMAoGCCqGSM49BAMCMB0xGzAZBgNVBAMTEjAxMjM0NjI0QjIwMjYwRDdF\n"
"RTAeFw0yMTExMTgxMTAwMDBaFw0yNjExMTgxMTAwMDBaMB0xGzAZBgNVBAMTEjAxMjM0NjI0QjIw\n"
"MjYwRDdFRTEBZMBMGByqGSM49AgEGCCqGSM49AwEHA0IABDOFj/SnArwqM15cZs/bXppfTuAxgMzB\n"
"N3LS48xHSqpLhHlVnvOvWqyhE8v+ZX4Jz1o7Z9LGOG537EeldBeGjYi jA jAAMAoGCCqGSM49BAMC\n"
"A0kAMEYCIQC9O1l85YX1+9vZ0t/SHQ3zFH5e7Vc8XtrZ+mTtMc5riwIhAL/SektrG3C0JwII0VV5\n"
"pSR9RRnuwo810km81P4S56/m\n"
"-----END CERTIFICATE-----\n"
```

Client certificate

7.3.2.3 cacerts

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

The Root Certificate store

7.4 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 getServerFragment** (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, int &nalgs, int *sigalgs)
process a Certificate Request

7.4.1 Detailed Description

Process Input received from the Server.

Author

Mike Scott

7.4.2 Function Documentation

7.4.2.1 parseoctad()

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

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

7.4.2.2 parsebytes()

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

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

7.4.2.3 parseInt()

```
ret parseInt (
    octad * M,
```

```
int len,  
int & ptr )
```

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 M, which advances after use

Returns

the integer value, and an error flag

7.4.2.4 `parseoctadptr()`

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

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 M
<i>len</i>	the expected length of the 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, and an error flag

7.4.2.5 `getServerFragment()`

```
int getServerFragment (  
    TLS_session * session )
```

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

7.4.2.6 parseIntorPull()

```
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.

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

7.4.2.7 parseoctadorPull()

```
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.

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

7.4.2.8 `parsebytesorPull()`

```
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.

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

7.4.2.9 `parseoctadorPullptrX()`

```
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.

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 <i>O</i>

Returns

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

7.4.2.10 `badResponse()`

```
bool badResponse (
    TLS_session * session,
    ret r )
```

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

7.4.2.11 seeWhatsNext()

```
ret seeWhatsNext (
    TLS_session * session )
```

Identify type of incoming message.

Parameters

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

Returns

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

7.4.2.12 getServerEncryptedExtensions()

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

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

7.4.2.13 getServerCertVerify()

```
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.

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

7.4.2.14 getServerFinished()

```
ret getServerFinished (
    TLS_session * session,
    octad * HFIN )
```

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

7.4.2.15 getServerHello()

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

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

7.4.2.16 getCheckServerCertificateChain()

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

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

7.4.2.17 getCertificateRequest()

```
ret getCertificateRequest (
    TLS_session * session,
    int & nalgs,
    int * sigalgs )
```

process a Certificate Request

Parameters

<i>session</i>	the TLS session structure
<i>nalgs</i>	the number of acceptable signature algorithms
<i>sigalgs</i>	an array of nalgs signature algorithms

Returns

response structure

7.5 `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.
- 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 `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.
- 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 `sendFlushIO` (`TLS_session` *session)
Flush IO buffer.
- 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 `sendBinder` (`TLS_session` *session, `octad` *BND, bool flush)
Send a preshared key binder message to the Server.
- void `sendClientHello` (`TLS_session` *session, int version, `octad` *CH, 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 `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.

7.5.1 Detailed Description

Process Output to be sent to the Server.

Author

Mike Scott

7.5.2 Function Documentation

7.5.2.1 sendCCCS()

```
void sendCCCS (
    TLS_session * session )
```

Send Change Cipher Suite message.

Parameters

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

7.5.2.2 addPreSharedKeyExt()

```
int addPreSharedKeyExt (
    octad * EXT,
    uint32 age,
    octad * IDS,
    int sha )
```

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

7.5.2.3 addServerNameExt()

```
void addServerNameExt (
    octad * EXT,
    char * servername )
```

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

7.5.2.4 addSupportedGroupsExt()

```
void addSupportedGroupsExt (
    octad * EXT,
    int nsg,
    int * supportedGroups )
```

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

7.5.2.5 addSigAlgsExt()

```
void addSigAlgsExt (
    octad * EXT,
    int nsa,
    int * sigAlgs )
```

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

7.5.2.6 addSigAlgsCertExt()

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

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

7.5.2.7 addKeyShareExt()

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

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

7.5.2.8 `addALPNExt()`

```
void addALPNExt (
    octad * EXT,
    octad * AP )
```

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

7.5.2.9 `addMFLExt()`

```
void addMFLExt (
    octad * EXT,
    int mode )
```

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

7.5.2.10 `addRSLExt()`

```
void addRSLExt (
    octad * EXT,
    int size )
```

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

7.5.2.11 addPSKModesExt()

```
void addPSKModesExt (
    octad * EXT,
    int mode )
```

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

7.5.2.12 addVersionExt()

```
void addVersionExt (
    octad * EXT,
    int version )
```

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

7.5.2.13 addPadding()

```
void addPadding (
    octad * EXT,
    int n )
```

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

7.5.2.14 addCookieExt()

```
void addCookieExt (
```

```
    octad * EXT,  
    octad * CK )
```

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

7.5.2.15 addEarlyDataExt()

```
void addEarlyDataExt (  
    octad * EXT )
```

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

Parameters

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

7.5.2.16 clientRandom()

```
int clientRandom (  
    octad * RN )
```

Generate 32-byte random octad.

Parameters

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

Returns

length of output octad

7.5.2.17 cipherSuites()

```
int cipherSuites (  
    octad * CS,  
    int ncs,  
    int * ciphers )
```

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

7.5.2.18 sendFlushIO()

```
void sendFlushIO (
    TLS_session * session )
```

Flush IO buffer.

Parameters

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

7.5.2.19 sendClientMessage()

```
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.

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

7.5.2.20 sendBinder()

```
void sendBinder (
    TLS_session * session,
    octad * BND,
    bool flush )
```

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
<i>flush</i>	transmit immediately if true

7.5.2.21 sendClientHello()

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

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>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

7.5.2.22 sendAlert()

```
void sendAlert (
    TLS_session * session,
    int type )
```

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

7.5.2.23 sendClientFinish()

```
void sendClientFinish (
    TLS_session * session,
    octad * CHF )
```

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

7.5.2.24 sendClientCertificateChain()

```
void sendClientCertificateChain (
    TLS_session * session,
    octad * CERTCHAIN )
```

Prepare and send client certificate message to the Server.

Parameters

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

7.5.2.25 sendClientCertVerify()

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

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

7.5.2.26 `sendEndOfEarlyData()`

```
void sendEndOfEarlyData (
    TLS_session * session )
```

Indicate End of Early Data in message to the Server.

Parameters

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

7.5.2.27 `alert_from_cause()`

```
int alert_from_cause (
    int rtn )
```

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

7.6 `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 *`)
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.

7.6.1 Detailed Description

TLS 1.3 crypto support functions.

Author

Mike Scott

7.6.2 Function Documentation

7.6.2.1 `initTranscriptHash()`

```
void initTranscriptHash (  
    TLS_session * session )
```

Initialise Transcript hash.

Parameters

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

7.6.2.2 `runningHash()`

```
void runningHash (  
    TLS_session * session,  
    octad * O )
```

Accumulate octad into ongoing hashing.

Parameters

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

7.6.2.3 `runningHashIO()`

```
void runningHashIO (  
    TLS_session * session )
```

Accumulate transcript hash from IO buffer.

Parameters

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

7.6.2.4 rewindIO()

```
void rewindIO (  
    TLS_session * session )
```

rewind the IO buffer

Parameters

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

7.6.2.5 runningHashIOrewind()

```
void runningHashIOrewind (  
    TLS_session * session )
```

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

Parameters

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

7.6.2.6 transcriptHash()

```
void transcriptHash (  
    TLS_session * session,  
    octad * O )
```

Output current hash value.

Parameters

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

7.6.2.7 runningSyntheticHash()

```
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)

Parameters

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

7.6.2.8 initCryptoContext()

```
void initCryptoContext (
    crypto * C )
```

Initiate a Crypto Context.

Parameters

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

7.6.2.9 updateCryptoContext()

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

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

7.6.2.10 incrementCryptoContext()

```
void incrementCryptoContext (
    crypto * C )
```

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

Parameters

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

7.6.2.11 createCryptoContext()

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

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

7.6.2.12 createSendCryptoContext()

```
void createSendCryptoContext (
    TLS_session * session,
    octad * TS )
```

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

7.6.2.13 createRecvCryptoContext()

```
void createRecvCryptoContext (
```



```
TLS_session * session,  
octad * TS )
```

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

7.6.2.14 recoverPSK()

```
void recoverPSK (  
    TLS_session * session )
```

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

Parameters

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

7.6.2.15 deriveEarlySecrets()

```
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)

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)

7.6.2.16 deriveLaterSecrets()

```
void deriveLaterSecrets (  

```

```

int htype,
octad * H,
octad * ES,
octad * CETS,
octad * EEMS )

```

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)

7.6.2.17 deriveHandshakeSecrets()

```

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.

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

7.6.2.18 deriveApplicationSecrets()

```

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.

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)

7.6.2.19 deriveUpdatedKeys()

```
void deriveUpdatedKeys (
    crypto * context,
    octad * TS )
```

Perform a Key Update on a crypto context.

Parameters

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

7.6.2.20 checkVerifierData()

```
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.

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

7.6.2.21 deriveVerifierData()

```
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.

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

7.6.2.22 checkServerCertVerifier()

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

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

7.6.2.23 createClientCertVerifier()

```
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.

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

7.7 `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

7.7.1 Detailed Description

TLS 1.3 logging.

Author

Mike Scott

7.7.2 Function Documentation

7.7.2.1 myprintf()

```
void myprintf (
    char * s )
```

internal printf function - all output funnels through this function

Parameters

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

7.7.2.2 log()

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

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)

7.7.2.3 logServerHello()

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

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)

7.7.2.4 logTicket()

```
void logTicket (
    ticket * T )
```

logging a resumption ticket

Parameters

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

7.7.2.5 logEncExt()

```
void logEncExt (
    ee_status * e,
    ee_status * r )
```

logging server extended extensions responses vs expectations

Parameters

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

7.7.2.6 logCert()

```
void logCert (
    octad * CERT )
```

logging a Certificate in standard base 64 format

Parameters

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

7.7.2.7 logCertDetails()

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

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

7.7.2.8 logServerResponse()

```
void logServerResponse (
    ret r )
```

log client processing of a Server response

Parameters

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

7.7.2.9 logAlert()

```
void logAlert (
    int detail )
```

log Server Alert

Parameters

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

7.7.2.10 `logCipherSuite()`

```
void logCipherSuite (
    int cipher_suite )
```

log Cipher Suite

Parameters

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

7.7.2.11 `logKeyExchange()`

```
void logKeyExchange (
    int kex )
```

log Key Exchange Group

Parameters

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

7.7.2.12 `logSigAlg()`

```
void logSigAlg (
    int sigAlg )
```

log Signature Algorithm

Parameters

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

7.8 `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

- 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.

7.8.1 Detailed Description

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

Author

Mike Scott

7.8.2 Function Documentation

7.8.2.1 OCT_append_int()

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

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

7.8.2.2 OCT_append_octad()

```
void OCT_append_octad (
    octad * O,
    octad * P )
```

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

7.8.2.3 OCT_compare()

```
bool OCT_compare (
    octad * O,
    octad * P )
```

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

7.8.2.4 OCT_shift_left()

```
void OCT_shift_left (
    octad * O,
    int n )
```

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

7.8.2.5 OCT_kill()

```
void OCT_kill (
    octad * O )
```

Wipe clean an octad.

Parameters

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

7.8.2.6 OCT_from_hex()

```
void OCT_from_hex (
    octad * O,
    char * src )
```

Convert a hex number to an octad.

Parameters

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

7.8.2.7 OCT_append_string()

```
void OCT_append_string (
    octad * O,
    char * s )
```

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

7.8.2.8 OCT_append_byte()

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

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

7.8.2.9 OCT_append_bytes()

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

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

7.8.2.10 OCT_from_base64()

```
void OCT_from_base64 (
    octad * O,
    char * b )
```

Create an octad from a base64 number.

Parameters

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

7.8.2.11 OCT_reverse()

```
void OCT_reverse (
    octad * O )
```

Reverse bytes in an octad.

Parameters

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

7.8.2.12 OCT_truncate()

```
void OCT_truncate (
    octad * O,
    int n )
```

Reverse bytes in an octad.

Parameters

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

7.8.2.13 OCT_copy()

```
void OCT_copy (
```

```
    octad * O,  
    octad * P )
```

Copy one octad into another.

Parameters

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

7.8.2.14 OCT_output_hex()

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

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

7.8.2.15 OCT_output_string()

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

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

7.8.2.16 OCT_output_base64()

```
void OCT_output_base64 (  

```

```

    octad * O,
    int max,
    char * s )

```

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

7.9 tls_protocol.h File Reference

TLS 1.3 main client-side protocol functions.

```

#include "tls_keys_calc.h"
#include "tls_cert_chain.h"
#include "tls_client_recv.h"
#include "tls_client_send.h"
#include "tls_tickets.h"
#include "tls_logger.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
- bool [TLS13_connect](#) ([TLS_session](#) *session, [octad](#) *EARLY)
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)
TLS 1.3 receive data.
- void [TLS13_clean](#) ([TLS_session](#) *session)
TLS 1.3 end session, delete keys, clean up buffers.

7.9.1 Detailed Description

TLS 1.3 main client-side protocol functions.

Author

Mike Scott

7.9.2 Function Documentation

7.9.2.1 TLS13_start()

```
TLS_session TLS13_start (
    Socket * client,
    char * hostname )
```

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

7.9.2.2 TLS13_end()

```
void TLS13_end (
    TLS_session * session )
```

terminate a session structure

Parameters

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

7.9.2.3 TLS13_connect()

```
bool TLS13_connect (
    TLS_session * session,
    octad * EARLY )
```

TLS 1.3 forge connection.

Parameters

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

Returns

false for failure, true for success

7.9.2.4 TLS13_send()

```
void TLS13_send (
    TLS_session * session,
    octad * DATA )
```

TLS 1.3 send data.

Parameters

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

7.9.2.5 TLS13_rcv()

```
int TLS13_rcv (
    TLS_session * session,
    octad * DATA )
```

TLS 1.3 receive data.

Parameters

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

Returns

0 for failure, otherwise success

7.9.2.6 TLS13_clean()

```
void TLS13_clean (
    TLS_session * session )
```

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

Parameters

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

7.10 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*
- void `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.*

7.10.1 Detailed Description

Security Abstraction Layer for TLS.

Author

Mike Scott

7.10.2 Function Documentation

7.10.2.1 SAL_name()

```
char* SAL_name ( )
```

Return name of SAL provider.

Returns

name of SAL provider

7.10.2.2 SAL_ciphers()

```
int SAL_ciphers (
    int * ciphers )
```

Return supported ciphers.

Parameters

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

Returns

number of supported ciphers

7.10.2.3 SAL_groups()

```
int SAL_groups (
    int * groups )
```

Return supported groups in preferred order.

Parameters

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

Returns

number of supported groups

7.10.2.4 SAL_sigs()

```
int SAL_sigs (
    int * sigAlgs )
```

Return supported TLS signature algorithms in preferred order.

Parameters

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

Returns

number of supported groups

7.10.2.5 SAL_sigCerts()

```
int SAL_sigCerts (
    int * sigAlgsCert )
```

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

7.10.2.6 SAL_initLib()

```
bool SAL_initLib ( )
```

Initialise library for use.

Returns

return true if successful, else false

7.10.2.7 SAL_hashType()

```
int SAL_hashType (
    int cipher_suite )
```

return hash type associated with a cipher suite

Parameters

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

Returns

hash function output length

7.10.2.8 SAL_hashLen()

```
int SAL_hashLen (
    int hash_type )
```

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

7.10.2.9 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

7.10.2.10 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

7.10.2.11 SAL_randomByte()

```
int SAL_randomByte ( )
```

get a random byte

Returns

a random byte

7.10.2.12 SAL_randomOctad()

```
void SAL_randomOctad (
    int len,
    octad * R )
```

get a random octad

Parameters

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

7.10.2.13 SAL_hkdfExtract()

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

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

7.10.2.14 SAL_hkdfExpand()

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

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

7.10.2.15 SAL_hmac()

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

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

7.10.2.16 SAL_hashNull()

```
void SAL_hashNull (
    int sha,
    octad * H )
```

simple HASH of nothing function

Parameters

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

7.10.2.17 SAL_hashInit()

```
void SAL_hashInit (
    int hlen,
    unihash * h )
```

Initiate Hashing context.

Parameters

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

7.10.2.18 SAL_hashProcessArray()

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

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

7.10.2.19 SAL_hashOutput()

```
int SAL_hashOutput (
    unihash * h,
    char * d )
```

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

7.10.2.20 SAL_aeadEncrypt()

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

```

    int hdrlen,
    char * hdr,
    int ptlen,
    char * pt,
    octad * TAG )

```

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

7.10.2.21 SAL_aeadDecrypt()

```

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

```

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

7.10.2.22 SAL_generateKeyPair()

```

void SAL_generateKeyPair (
    int group,

```

```
    octad * SK,  
    octad * PK )
```

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

7.10.2.23 SAL_generateSharedSecret()

```
void SAL_generateSharedSecret (  
    int group,  
    octad * SK,  
    octad * PK,  
    octad * SS )
```

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

7.10.2.24 SAL_tlsSignatureVerify()

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

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

7.10.2.25 SAL_tlsSignature()

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

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

7.11 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 octet over a socket
- `void sendLen (Socket *client, int len)`
send a 16-bit integer as an octet 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 getByte (Socket *client)`
receive a single byte from a socket
- `int getOctad (Socket *client, octad *B, int expected)`
receive an octet from a socket

7.11.1 Detailed Description

set up sockets for reading and writing

Author

Mike Scott

7.11.2 Function Documentation

7.11.2.1 `setclientsock()`

```
int setclientsock (  
    int port,  
    char * ip,  
    int toms )
```

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

7.11.2.2 getIPAddress()

```
int getIPAddress (
    char * ip,
    char * hostname )
```

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

7.11.2.3 sendOctad()

```
void sendOctad (
    Socket * client,
    octad * B )
```

send an octet over a socket

Parameters

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

7.11.2.4 sendLen()

```
void sendLen (
    Socket * client,
    int len )
```

send a 16-bit integer as an octet to Server

Parameters

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

7.11.2.5 `getBytes()`

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

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

7.11.2.6 `getInt16()`

```
int getInt16 (
    Socket * client )
```

receive 16-bit integer from a socket

Parameters

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

Returns

a 16-bit integer

7.11.2.7 `getInt24()`

```
int getInt24 (
    Socket * client )
```

receive 24-bit integer from a socket

Parameters

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

Returns

a 24-bit integer

7.11.2.8 `getBytes()`

```
int getByte (
    Socket * client )
```

receive a single byte from a socket

Parameters

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

Returns

a byte

7.11.2.9 `getOctad()`

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

receive an octet from a socket

Parameters

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

Returns

-1 on failure, 0 on success

7.12 tls_tickets.h File Reference

TLS 1.3 process resumption tickets.

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

Functions

- unsigned long `millis` ()
read milliseconds from a stop-watch
- 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.

7.12.1 Detailed Description

TLS 1.3 process resumption tickets.

Author

Mike Scott

7.12.2 Function Documentation

7.12.2.1 `millis()`

```
unsigned long millis ( )
```

read milliseconds from a stop-watch

Returns

milliseconds read from stop-watch

7.12.2.2 `parseTicket()`

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

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

7.12.2.3 initTicketContext()

```
void initTicketContext (
    ticket * T )
```

initialize a ticket structure

Parameters

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

7.12.2.4 endTicketContext()

```
void endTicketContext (
    ticket * T )
```

terminate a ticket structure

Parameters

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

7.12.2.5 ticket_still_good()

```
bool ticket_still_good (
    ticket * T )
```

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

7.13 `tls_wifi.h` File Reference

define `Socket` structure depending on processor context

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

7.13.1 Detailed Description

define `Socket` structure depending on processor context

Author

Mike Scott

7.14 `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_H256` 2
- #define `X509_H384` 3
- #define `X509_H512` 4
- #define `USE_NIST256` 0
- #define `USE_C25519` 1
- #define `USE_NIST384` 10
- #define `USE_NIST521` 12

Functions

- `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)`
- `pktype X509_extract_public_key (octad *c, octad *k)`
- `int X509_find_issuer (octad *c)`
- `int X509_find_validity (octad *c)`
- `int X509_find_subject (octad *c)`
- `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`

7.14.1 Detailed Description

X509 function Header File.

Author

Mike Scott

defines structures declares functions

7.14.2 Macro Definition Documentation

7.14.2.1 X509_ECC

```
#define X509_ECC 1
```

Elliptic Curve data type detected

7.14.2.2 X509_RSA

```
#define X509_RSA 2
```

RSA data type detected

7.14.2.3 X509_ECD

```
#define X509_ECD 3
```

Elliptic Curve (Ed25519) detected

7.14.2.4 X509_PQ

```
#define X509_PQ 4
```

Post Quantum method

7.14.2.5 X509_H256

```
#define X509_H256 2
```

SHA256 hash algorithm used

7.14.2.6 X509_H384

```
#define X509_H384 3
```

SHA384 hash algorithm used

7.14.2.7 X509_H512

```
#define X509_H512 4
```

SHA512 hash algorithm used

7.14.2.8 USE_NIST256

```
#define USE_NIST256 0
```

For the NIST 256-bit standard curve - WEIERSTRASS only

7.14.2.9 USE_C25519

```
#define USE_C25519 1
```

Bernstein's Modulus 2^{255-19} - EDWARDS or MONTGOMERY only

7.14.2.10 USE_NIST384

```
#define USE_NIST384 10
```

For the NIST 384-bit standard curve - WEIERSTRASS only

7.14.2.11 USE_NIST521

```
#define USE_NIST521 12
```

For the NIST 521-bit standard curve - WEIERSTRASS only

7.14.3 Function Documentation

7.14.3.1 X509_extract_private_key()

```
pktype X509_extract_private_key (
    octad * c,
    octad * pk )
```

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)

7.14.3.2 X509_extract_cert_sig()

```
pktype X509_extract_cert_sig (
    octad * c,
    octad * s )
```

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)

7.14.3.3 X509_extract_cert()

```
int X509_extract_cert (
    octad * sc,
    octad * c )
```

Parameters

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

Returns

0 on failure

7.14.3.4 X509_extract_public_key()

```
pktype X509_extract_public_key (
    octad * c,
    octad * k )
```

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)

7.14.3.5 X509_find_issuer()

```
int X509_find_issuer (
    octad * c )
```

Parameters

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

Returns

0 on failure, or pointer to issuer field in cert

7.14.3.6 X509_find_validity()

```
int X509_find_validity (  
    octad * c )
```

Parameters

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

Returns

0 on failure, or pointer to validity field in cert

7.14.3.7 X509_find_subject()

```
int X509_find_subject (  
    octad * c )
```

Parameters

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

Returns

0 on failure, or pointer to subject field in cert

7.14.3.8 X509_self_signed()

```
int X509_self_signed (  
    octad * c )
```

Parameters

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

Returns

true if self-signed, else false

7.14.3.9 X509_find_entity_property()

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

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

7.14.3.10 X509_find_start_date()

```
int X509_find_start_date (
    octad * c,
    int s )
```

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

7.14.3.11 X509_find_expiry_date()

```
int X509_find_expiry_date (
    octad * c,
    int s )
```

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

7.14.3.12 X509_find_extensions()

```
int X509_find_extensions (
    octad * c )
```

Parameters

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

Returns

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

7.14.3.13 X509_find_extension()

```
int X509_find_extension (
    octad * c,
    octad * S,
    int s,
    int * f )
```

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

7.14.3.14 X509_find_alt_name()

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

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

7.14.4 Variable Documentation

7.14.4.1 X509_CN

```
octad X509_CN [extern]
```

Country Name

7.14.4.2 X509_ON

```
octad X509_ON [extern]
```

organisation Name

7.14.4.3 X509_EN

```
octad X509_EN [extern]
```

email

7.14.4.4 X509_LN

```
octad X509_LN [extern]
```

local name

7.14.4.5 X509_UN

`octad X509_UN [extern]`

Unit name (aka Organisation Unit OU)

7.14.4.6 X509_MN

`octad X509_MN [extern]`

My Name (aka Common Name)

7.14.4.7 X509_SN

`octad X509_SN [extern]`

State Name

7.14.4.8 X509_AN

`octad X509_AN [extern]`

Alternate Name

7.14.4.9 X509_KU

`octad X509_KU [extern]`

Key Usage

7.14.4.10 X509_BC

`octad X509_BC [extern]`

Basic Constraints

Index

active
 crypto, [17](#)
addALPNExt
 tls_client_send.h, [71](#)
addCookieExt
 tls_client_send.h, [72](#)
addEarlyDataExt
 tls_client_send.h, [73](#)
addKeyShareExt
 tls_client_send.h, [70](#)
addMFLExt
 tls_client_send.h, [71](#)
addPadding
 tls_client_send.h, [72](#)
addPreSharedKeyExt
 tls_client_send.h, [68](#)
addPSKModesExt
 tls_client_send.h, [71](#)
addRSLEExt
 tls_client_send.h, [71](#)
addServerNameExt
 tls_client_send.h, [69](#)
addSigAlgsCertExt
 tls_client_send.h, [70](#)
addSigAlgsExt
 tls_client_send.h, [70](#)
addSupportedGroupsExt
 tls_client_send.h, [69](#)
addVersionExt
 tls_client_send.h, [72](#)
age_obfuscator
 ticket, [25](#)
ALERT
 tls1_3.h, [48](#)
alert_from_cause
 tls_client_send.h, [77](#)
ALLOW_SELF_SIGNED
 tls1_3.h, [37](#)
alpn
 ee_status, [20](#)
APP_PROTOCOL
 tls1_3.h, [47](#)
APPLICATION
 tls1_3.h, [48](#)
AUTHENTICATION_FAILURE
 tls1_3.h, [51](#)

BAD_CERT_CHAIN
 tls1_3.h, [50](#)
BAD_CERTIFICATE
 tls1_3.h, [52](#)
BAD_HELLO
 tls1_3.h, [50](#)
BAD_MESSAGE
 tls1_3.h, [52](#)
BAD_RECORD
 tls1_3.h, [51](#)
BAD_TICKET
 tls1_3.h, [51](#)
badResponse
 tls_client_recv.h, [63](#)
birth
 ticket, [26](#)
byte
 tls1_3.h, [54](#)

CA_NOT_FOUND
 tls1_3.h, [51](#)
cacerts
 tls_certs.h, [58](#)
CERT_OUTOFDATE
 tls1_3.h, [51](#)
CERT_REQUEST
 tls1_3.h, [49](#)
CERT_VERIFY
 tls1_3.h, [49](#)
CERTIFICATE
 tls1_3.h, [48](#)
CERTIFICATE_EXPIRED
 tls1_3.h, [53](#)
CHANGE_CIPHER
 tls1_3.h, [48](#)
checkServerCertChain
 tls_cert_chain.h, [56](#)
checkServerCertVerifier
 tls_keys_calc.h, [86](#)
checkVerifierData
 tls_keys_calc.h, [85](#)
cipher_suite
 ticket, [26](#)
 TLS_session, [28](#)
cipherSuites
 tls_client_send.h, [73](#)
CLIENT_CERT
 tls1_3.h, [37](#)
CLIENT_HELLO
 tls1_3.h, [48](#)
clientRandom
 tls_client_send.h, [73](#)
CLOSE_NOTIFY

- tls1_3.h, [53](#)
- COOKIE
 - tls1_3.h, [47](#)
- createClientCertVerifier
 - tls_keys_calc.h, [86](#)
- createCryptoContext
 - tls_keys_calc.h, [82](#)
- createRecvCryptoContext
 - tls_keys_calc.h, [82](#)
- createSendCryptoContext
 - tls_keys_calc.h, [82](#)
- crypto, [17](#)
 - active, [17](#)
 - IV, [18](#)
 - iv, [18](#)
 - K, [18](#)
 - k, [17](#)
 - record, [18](#)
 - suite, [18](#)
 - taglen, [18](#)
- CRYPTO_SETTING
 - tls1_3.h, [37](#)
- CTS
 - TLS_session, [29](#)
- cts
 - TLS_session, [29](#)
- curve
 - pktype, [22](#)
- DECODE_ERROR
 - tls1_3.h, [53](#)
- DECRYPT_ERROR
 - tls1_3.h, [52](#)
- deriveApplicationSecrets
 - tls_keys_calc.h, [84](#)
- deriveEarlySecrets
 - tls_keys_calc.h, [83](#)
- deriveHandshakeSecrets
 - tls_keys_calc.h, [84](#)
- deriveLaterSecrets
 - tls_keys_calc.h, [83](#)
- deriveUpdatedKeys
 - tls_keys_calc.h, [85](#)
- deriveVerifierData
 - tls_keys_calc.h, [85](#)
- DILITHIUM3
 - tls1_3.h, [45](#)
- DLT_SS
 - tls1_3.h, [36](#)
- EARLY_DATA
 - tls1_3.h, [47](#)
- early_data
 - ee_status, [20](#)
- ECC_SS
 - tls1_3.h, [36](#)
- ECCX08Class, [19](#)
- ECDSA_SECP256R1_SHA256
 - tls1_3.h, [44](#)
- ECDSA_SECP384R1_SHA384
 - tls1_3.h, [44](#)
- ED25519
 - tls1_3.h, [45](#)
- ee_status, [19](#)
 - alpn, [20](#)
 - early_data, [20](#)
 - max_frag_length, [20](#)
 - server_name, [20](#)
- EMPTY_CERT_CHAIN
 - tls1_3.h, [52](#)
- ENCRYPTED_EXTENSIONS
 - tls1_3.h, [49](#)
- END_OF_EARLY_DATA
 - tls1_3.h, [49](#)
- endTicketContext
 - tls_tickets.h, [118](#)
- err
 - ret, [23](#)
- favourite_group
 - ticket, [26](#)
 - TLS_session, [28](#)
- FINISHED
 - tls1_3.h, [49](#)
- FORBIDDEN_EXTENSION
 - tls1_3.h, [51](#)
- getBytes
 - tls_sockets.h, [116](#)
- getBytes
 - tls_sockets.h, [115](#)
- getCertificateRequest
 - tls_client_recv.h, [66](#)
- getCheckServerCertificateChain
 - tls_client_recv.h, [66](#)
- getClientPrivateKeyandCertChain
 - tls_cert_chain.h, [57](#)
- getInt16
 - tls_sockets.h, [115](#)
- getInt24
 - tls_sockets.h, [115](#)
- getIPAddress
 - tls_sockets.h, [114](#)
- getOctad
 - tls_sockets.h, [116](#)
- getServerCertVerify
 - tls_client_recv.h, [64](#)
- getServerEncryptedExtensions
 - tls_client_recv.h, [64](#)
- getServerFinished
 - tls_client_recv.h, [65](#)
- getServerFragment
 - tls_client_recv.h, [61](#)
- getServerHello
 - tls_client_recv.h, [65](#)
- HANDSHAKE_RETRY
 - tls1_3.h, [50](#)

- hash
 - pktype, [22](#)
- hostname
 - TLS_session, [28](#)
- HS
 - TLS_session, [28](#)
- hs
 - TLS_session, [28](#)
- HSHAKE
 - tls1_3.h, [47](#)
- htype
 - unihash, [30](#)
- HW_1
 - tls1_3.h, [36](#)
- HW_2
 - tls1_3.h, [36](#)
- id
 - TLS_session, [27](#)
- ID_MISMATCH
 - tls1_3.h, [50](#)
- ILLEGAL_PARAMETER
 - tls1_3.h, [52](#)
- incrementCryptoContext
 - tls_keys_calc.h, [81](#)
- initCryptoContext
 - tls_keys_calc.h, [81](#)
- initTicketContext
 - tls_tickets.h, [118](#)
- initTranscriptHash
 - tls_keys_calc.h, [79](#)
- IO
 - TLS_session, [29](#)
- io
 - TLS_session, [29](#)
- IO_APPLICATION
 - tls1_3.h, [35](#)
- IO_DEBUG
 - tls1_3.h, [35](#)
- IO_NONE
 - tls1_3.h, [35](#)
- IO_PROTOCOL
 - tls1_3.h, [35](#)
- IO_WIRE
 - tls1_3.h, [35](#)
- IV
 - crypto, [18](#)
- iv
 - crypto, [18](#)
- K
 - crypto, [18](#)
- k
 - crypto, [17](#)
- K_rcv
 - TLS_session, [28](#)
- K_send
 - TLS_session, [28](#)
- KEY_SHARE
 - tls1_3.h, [46](#)
- KEY_UPDATE
 - tls1_3.h, [49](#)
- KYBER768
 - tls1_3.h, [43](#)
- len
 - octad, [21](#)
- lifetime
 - ticket, [26](#)
- log
 - tls_logger.h, [88](#)
- LOG_OUTPUT_TRUNCATION
 - tls1_3.h, [53](#)
- logAlert
 - tls_logger.h, [90](#)
- logCert
 - tls_logger.h, [89](#)
- logCertDetails
 - tls_logger.h, [90](#)
- logCipherSuite
 - tls_logger.h, [91](#)
- logEncExt
 - tls_logger.h, [89](#)
- logKeyExchange
 - tls_logger.h, [91](#)
- logServerHello
 - tls_logger.h, [88](#)
- logServerResponse
 - tls_logger.h, [90](#)
- logSigAlg
 - tls_logger.h, [91](#)
- logTicket
 - tls_logger.h, [89](#)
- max
 - octad, [21](#)
- max_early_data
 - ticket, [25](#)
- MAX_EXCEEDED
 - tls1_3.h, [52](#)
- MAX_FRAG_LENGTH
 - tls1_3.h, [47](#)
- max_frag_length
 - ee_status, [20](#)
- max_record
 - TLS_session, [27](#)
- MEM_OVERFLOW
 - tls1_3.h, [51](#)
- MESSAGE_HASH
 - tls1_3.h, [49](#)
- millis
 - tls_tickets.h, [117](#)
- MISSING_REQUEST_CONTEXT
 - tls1_3.h, [50](#)
- mycert
 - tls_certs.h, [58](#)
- myprintf
 - tls_logger.h, [88](#)

- myprivate
 - tls_certs.h, 58
- NOCERT
 - tls1_3.h, 36
- NONCE
 - ticket, 25
- nonce
 - ticket, 25
- NOT_EXPECTED
 - tls1_3.h, 51
- NOT_TLS1_3
 - tls1_3.h, 50
- OCT_append_byte
 - tls_octads.h, 95
- OCT_append_bytes
 - tls_octads.h, 95
- OCT_append_int
 - tls_octads.h, 93
- OCT_append_octad
 - tls_octads.h, 93
- OCT_append_string
 - tls_octads.h, 95
- OCT_compare
 - tls_octads.h, 93
- OCT_copy
 - tls_octads.h, 96
- OCT_from_base64
 - tls_octads.h, 96
- OCT_from_hex
 - tls_octads.h, 94
- OCT_kill
 - tls_octads.h, 94
- OCT_output_base64
 - tls_octads.h, 97
- OCT_output_hex
 - tls_octads.h, 97
- OCT_output_string
 - tls_octads.h, 97
- OCT_reverse
 - tls_octads.h, 96
- OCT_shift_left
 - tls_octads.h, 94
- OCT_truncate
 - tls_octads.h, 96
- octad, 20
 - len, 21
 - max, 21
 - val, 21
- origin
 - ticket, 26
- PADDING
 - tls1_3.h, 47
- parsebytes
 - tls_client_recv.h, 60
- parsebytesorPull
 - tls_client_recv.h, 62
- parseInt
 - tls_client_recv.h, 60
- parseIntorPull
 - tls_client_recv.h, 62
- parseoctad
 - tls_client_recv.h, 59
- parseoctadorPull
 - tls_client_recv.h, 62
- parseoctadorPullptrX
 - tls_client_recv.h, 63
- parseoctadptr
 - tls_client_recv.h, 61
- parseTicket
 - tls_tickets.h, 117
- pktype, 21
 - curve, 22
 - hash, 22
 - type, 22
- POST_QUANTUM
 - tls1_3.h, 35
- PRESHARED_KEY
 - tls1_3.h, 46
- PROTOCOL_VERSION
 - tls1_3.h, 53
- PSK
 - ticket, 25
- psk
 - ticket, 25
- PSK_MODE
 - tls1_3.h, 46
- PSKOK
 - tls1_3.h, 45
- PSKWECDHE
 - tls1_3.h, 45
- ptr
 - TLS_session, 29
- record
 - crypto, 18
- RECORD_OVERFLOW
 - tls1_3.h, 53
- RECORD_SIZE_LIMIT
 - tls1_3.h, 47
- recoverPSK
 - tls_keys_calc.h, 83
- ret, 22
 - err, 23
 - val, 23
- rewindIO
 - tls_keys_calc.h, 80
- RMS
 - TLS_session, 28
- rms
 - TLS_session, 29
- RSA_PKCS1_SHA256
 - tls1_3.h, 44
- RSA_PKCS1_SHA384
 - tls1_3.h, 44
- RSA_PKCS1_SHA512
 - tls1_3.h, 44

- tls1_3.h, [44](#)
- RSA_PSS_RSAE_SHA256
 - tls1_3.h, [44](#)
- RSA_PSS_RSAE_SHA384
 - tls1_3.h, [44](#)
- RSA_PSS_RSAE_SHA512
 - tls1_3.h, [44](#)
- RSA_SS
 - tls1_3.h, [36](#)
- runningHash
 - tls_keys_calc.h, [79](#)
- runningHashIO
 - tls_keys_calc.h, [79](#)
- runningHashIOrewind
 - tls_keys_calc.h, [80](#)
- runningSyntheticHash
 - tls_keys_calc.h, [80](#)
- SAL_aeadDecrypt
 - tls_sal.h, [110](#)
- SAL_aeadEncrypt
 - tls_sal.h, [109](#)
- SAL_aeadKeylen
 - tls_sal.h, [106](#)
- SAL_aeadTaglen
 - tls_sal.h, [106](#)
- SAL_ciphers
 - tls_sal.h, [102](#)
- SAL_generateKeyPair
 - tls_sal.h, [110](#)
- SAL_generateSharedSecret
 - tls_sal.h, [111](#)
- SAL_groups
 - tls_sal.h, [103](#)
- SAL_hashInit
 - tls_sal.h, [108](#)
- SAL_hashLen
 - tls_sal.h, [105](#)
- SAL_hashNull
 - tls_sal.h, [108](#)
- SAL_hashOutput
 - tls_sal.h, [109](#)
- SAL_hashProcessArray
 - tls_sal.h, [109](#)
- SAL_hashType
 - tls_sal.h, [105](#)
- SAL_hkdfExpand
 - tls_sal.h, [107](#)
- SAL_hkdfExtract
 - tls_sal.h, [107](#)
- SAL_hmac
 - tls_sal.h, [108](#)
- SAL_initLib
 - tls_sal.h, [105](#)
- SAL_name
 - tls_sal.h, [102](#)
- SAL_randomByte
 - tls_sal.h, [106](#)
- SAL_randomOctad
 - tls_sal.h, [107](#)
- SAL_sigCerts
 - tls_sal.h, [103](#)
- SAL_sigs
 - tls_sal.h, [103](#)
- SAL_tlsSignature
 - tls_sal.h, [112](#)
- SAL_tlsSignatureVerify
 - tls_sal.h, [111](#)
- SECP256R1
 - tls1_3.h, [43](#)
- SECP384R1
 - tls1_3.h, [43](#)
- SECP521R1
 - tls1_3.h, [43](#)
- seeWhatsNext
 - tls_client_rcv.h, [64](#)
- SELF_SIGNED_CERT
 - tls1_3.h, [52](#)
- sendAlert
 - tls_client_send.h, [75](#)
- sendBinder
 - tls_client_send.h, [74](#)
- sendCCCS
 - tls_client_send.h, [68](#)
- sendClientCertificateChain
 - tls_client_send.h, [76](#)
- sendClientCertVerify
 - tls_client_send.h, [76](#)
- sendClientFinish
 - tls_client_send.h, [76](#)
- sendClientHello
 - tls_client_send.h, [75](#)
- sendClientMessage
 - tls_client_send.h, [74](#)
- sendEndOfEarlyData
 - tls_client_send.h, [77](#)
- sendFlushIO
 - tls_client_send.h, [74](#)
- sendLen
 - tls_sockets.h, [114](#)
- sendOctad
 - tls_sockets.h, [114](#)
- SERVER_HELLO
 - tls1_3.h, [48](#)
- SERVER_NAME
 - tls1_3.h, [46](#)
- server_name
 - ee_status, [20](#)
- setclientsock
 - tls_sockets.h, [113](#)
- SIG_ALGS
 - tls1_3.h, [46](#)
- SIG_ALGS_CERT
 - tls1_3.h, [46](#)
- sign16
 - tls1_3.h, [55](#)
- sign32

- tls1_3.h, [55](#)
- sign64
 - tls1_3.h, [55](#)
- sign8
 - tls1_3.h, [55](#)
- Socket, [23](#)
- sockptr
 - TLS_session, [27](#)
- state
 - unihash, [30](#)
- status
 - TLS_session, [27](#)
- STS
 - TLS_session, [29](#)
- sts
 - TLS_session, [29](#)
- suite
 - crypto, [18](#)
- SUPPORTED_GROUPS
 - tls1_3.h, [46](#)
- T
 - TLS_session, [30](#)
- taglen
 - crypto, [18](#)
- THIS_YEAR
 - tls1_3.h, [36](#)
- TICK
 - ticket, [25](#)
- tick
 - ticket, [25](#)
- TICKET
 - tls1_3.h, [49](#)
- ticket, [24](#)
 - age_obfuscator, [25](#)
 - birth, [26](#)
 - cipher_suite, [26](#)
 - favourite_group, [26](#)
 - lifetime, [26](#)
 - max_early_data, [25](#)
 - NONCE, [25](#)
 - nonce, [25](#)
 - origin, [26](#)
 - PSK, [25](#)
 - psk, [25](#)
 - TICK, [25](#)
 - tick, [25](#)
 - valid, [24](#)
- ticket_still_good
 - tls_tickets.h, [118](#)
- TIMED_OUT
 - tls1_3.h, [48](#)
- TINY_ECC
 - tls1_3.h, [35](#)
- TLS13_clean
 - tls_protocol.h, [100](#)
- TLS13_connect
 - tls_protocol.h, [99](#)
- TLS13_CONNECTED
 - tls1_3.h, [54](#)
- TLS13_DISCONNECTED
 - tls1_3.h, [54](#)
- TLS13_end
 - tls_protocol.h, [99](#)
- TLS13_recv
 - tls_protocol.h, [100](#)
- TLS13_send
 - tls_protocol.h, [100](#)
- TLS13_start
 - tls_protocol.h, [99](#)
- TLS1_0
 - tls1_3.h, [45](#)
- TLS1_2
 - tls1_3.h, [45](#)
- TLS1_3
 - tls1_3.h, [46](#)
- tls1_3.h, [31](#)
 - ALERT, [48](#)
 - ALLOW_SELF_SIGNED, [37](#)
 - APP_PROTOCOL, [47](#)
 - APPLICATION, [48](#)
 - AUTHENTICATION_FAILURE, [51](#)
 - BAD_CERT_CHAIN, [50](#)
 - BAD_CERTIFICATE, [52](#)
 - BAD_HELLO, [50](#)
 - BAD_MESSAGE, [52](#)
 - BAD_RECORD, [51](#)
 - BAD_TICKET, [51](#)
 - byte, [54](#)
 - CA_NOT_FOUND, [51](#)
 - CERT_OUTOFDATE, [51](#)
 - CERT_REQUEST, [49](#)
 - CERT_VERIFY, [49](#)
 - CERTIFICATE, [48](#)
 - CERTIFICATE_EXPIRED, [53](#)
 - CHANGE_CIPHER, [48](#)
 - CLIENT_CERT, [37](#)
 - CLIENT_HELLO, [48](#)
 - CLOSE_NOTIFY, [53](#)
 - COOKIE, [47](#)
 - CRYPTO_SETTING, [37](#)
 - DECODE_ERROR, [53](#)
 - DECRYPT_ERROR, [52](#)
 - DILITHIUM3, [45](#)
 - DLT_SS, [36](#)
 - EARLY_DATA, [47](#)
 - ECC_SS, [36](#)
 - ECDSA_SECP256R1_SHA256, [44](#)
 - ECDSA_SECP384R1_SHA384, [44](#)
 - ED25519, [45](#)
 - EMPTY_CERT_CHAIN, [52](#)
 - ENCRYPTED_EXTENSIONS, [49](#)
 - END_OF_EARLY_DATA, [49](#)
 - FINISHED, [49](#)
 - FORBIDDEN_EXTENSION, [51](#)
 - HANDSHAKE_RETRY, [50](#)
 - HSHAKE, [47](#)

HW_1, 36
HW_2, 36
ID_MISMATCH, 50
ILLEGAL_PARAMETER, 52
IO_APPLICATION, 35
IO_DEBUG, 35
IO_NONE, 35
IO_PROTOCOL, 35
IO_WIRE, 35
KEY_SHARE, 46
KEY_UPDATE, 49
KYBER768, 43
LOG_OUTPUT_TRUNCATION, 53
MAX_EXCEEDED, 52
MAX_FRAG_LENGTH, 47
MEM_OVERFLOW, 51
MESSAGE_HASH, 49
MISSING_REQUEST_CONTEXT, 50
NOCERT, 36
NOT_EXPECTED, 51
NOT_TLS1_3, 50
PADDING, 47
POST_QUANTUM, 35
PRESHARED_KEY, 46
PROTOCOL_VERSION, 53
PSK_MODE, 46
PSKOK, 45
PSKWECDHE, 45
RECORD_OVERFLOW, 53
RECORD_SIZE_LIMIT, 47
RSA_PKCS1_SHA256, 44
RSA_PKCS1_SHA384, 44
RSA_PKCS1_SHA512, 44
RSA_PSS_RSAE_SHA256, 44
RSA_PSS_RSAE_SHA384, 44
RSA_PSS_RSAE_SHA512, 44
RSA_SS, 36
SECP256R1, 43
SECP384R1, 43
SECP521R1, 43
SELF_SIGNED_CERT, 52
SERVER_HELLO, 48
SERVER_NAME, 46
SIG_ALGS, 46
SIG_ALGS_CERT, 46
sign16, 55
sign32, 55
sign64, 55
sign8, 55
SUPPORTED_GROUPS, 46
THIS_YEAR, 36
TICKET, 49
TIMED_OUT, 48
TINY_ECC, 35
TLS13_CONNECTED, 54
TLS13_DISCONNECTED, 54
TLS1_0, 45
TLS1_2, 45
TLS1_3, 46
TLS_AES_128_CCM_8_SHA256, 43
TLS_AES_128_CCM_SHA256, 43
TLS_AES_128_GCM_SHA256, 42
TLS_AES_256_GCM_SHA384, 42
TLS_APPLICATION_PROTOCOL, 37
TLS_CHACHA20_POLY1305_SHA256, 42
TLS_EARLY_DATA_ACCEPTED, 54
TLS_EXTERNAL_PSK, 45
TLS_FAILURE, 54
TLS_FULL_HANDSHAKE, 45
TLS_MAX_CERT_B64, 39
TLS_MAX_CERT_SIZE, 39
TLS_MAX_CIPHER_FRAG, 39
TLS_MAX_CIPHER_SUITES, 42
TLS_MAX_CLIENT_CHAIN_LEN, 40
TLS_MAX_CLIENT_CHAIN_SIZE, 41
TLS_MAX_COOKIE, 41
TLS_MAX_ECC_FIELD, 41
TLS_MAX_EXT_LABEL, 38
TLS_MAX_EXTENSIONS, 41
TLS_MAX_FRAG, 38
TLS_MAX_HASH, 38
TLS_MAX_HASH_STATE, 38
TLS_MAX_HELLO, 39
TLS_MAX_IO_SIZE, 39
TLS_MAX_IV_SIZE, 41
TLS_MAX_KEX_CIPHERTEXT_SIZE, 40
TLS_MAX_KEX_PUB_KEY_SIZE, 40
TLS_MAX_KEX_SECRET_KEY_SIZE, 40
TLS_MAX_KEY, 38
TLS_MAX_PLAIN_FRAG, 39
TLS_MAX_PSK_MODES, 42
TLS_MAX_SERVER_CHAIN_LEN, 40
TLS_MAX_SERVER_CHAIN_SIZE, 40
TLS_MAX_SERVER_NAME, 42
TLS_MAX_SHARED_SECRET_SIZE, 41
TLS_MAX_SIG_PUB_KEY_SIZE, 39
TLS_MAX_SIG_SECRET_KEY_SIZE, 40
TLS_MAX_SIGNATURE_SIZE, 40
TLS_MAX_SUPPORTED_GROUPS, 42
TLS_MAX_SUPPORTED_SIGS, 42
TLS_MAX_TAG_SIZE, 41
TLS_MAX_TICKET_SIZE, 41
TLS_RESUMPTION_REQUIRED, 54
TLS_SHA256_T, 37
TLS_SHA384_T, 37
TLS_SHA512_T, 38
TLS_SUCCESS, 54
TLS_VER, 47
TLS_X509_MAX_FIELD, 38
TRY_EARLY_DATA, 37
TYPICAL, 35
UNEXPECTED_MESSAGE, 52
UNKNOWN_CA, 53
UNRECOGNIZED_EXT, 50
unsign32, 55
unsign64, 55

- UNSUPPORTED_EXTENSION, 53
- VERBOSITY, 36
- WRONG_MESSAGE, 50
- X25519, 43
- X448, 43
- TLS_AES_128_CCM_8_SHA256
 - tls1_3.h, 43
- TLS_AES_128_CCM_SHA256
 - tls1_3.h, 43
- TLS_AES_128_GCM_SHA256
 - tls1_3.h, 42
- TLS_AES_256_GCM_SHA384
 - tls1_3.h, 42
- TLS_APPLICATION_PROTOCOL
 - tls1_3.h, 37
- tls_cert_chain.h, 56
 - checkServerCertChain, 56
 - getClientPrivateKeyandCertChain, 57
- tls_certs.h, 57
 - cacerts, 58
 - mycert, 58
 - myprivate, 58
- TLS_CHACHA20_POLY1305_SHA256
 - tls1_3.h, 42
- tls_client_rcv.h, 58
 - badResponse, 63
 - getCertificateRequest, 66
 - getCheckServerCertificateChain, 66
 - getServerCertVerify, 64
 - getServerEncryptedExtensions, 64
 - getServerFinished, 65
 - getServerFragment, 61
 - getServerHello, 65
 - parsebytes, 60
 - parsebytesorPull, 62
 - parseInt, 60
 - parseIntorPull, 62
 - parseoctad, 59
 - parseoctadorPull, 62
 - parseoctadorPullptrX, 63
 - parseoctadptr, 61
 - seeWhatsNext, 64
- tls_client_send.h, 67
 - addALPNExt, 71
 - addCookieExt, 72
 - addEarlyDataExt, 73
 - addKeyShareExt, 70
 - addMFLExt, 71
 - addPadding, 72
 - addPreSharedKeyExt, 68
 - addPSKModesExt, 71
 - addRSLExt, 71
 - addServerNameExt, 69
 - addSigAlgsCertExt, 70
 - addSigAlgsExt, 70
 - addSupportedGroupsExt, 69
 - addVersionExt, 72
 - alert_from_cause, 77
 - cipherSuites, 73
 - clientRandom, 73
 - sendAlert, 75
 - sendBinder, 74
 - sendCCCS, 68
 - sendClientCertificateChain, 76
 - sendClientCertVerify, 76
 - sendClientFinish, 76
 - sendClientHello, 75
 - sendClientMessage, 74
 - sendEndOfEarlyData, 77
 - sendFlushIO, 74
- TLS_EARLY_DATA_ACCEPTED
 - tls1_3.h, 54
- TLS_EXTERNAL_PSK
 - tls1_3.h, 45
- TLS_FAILURE
 - tls1_3.h, 54
- TLS_FULL_HANDSHAKE
 - tls1_3.h, 45
- tls_keys_calc.h, 77
 - checkServerCertVerifier, 86
 - checkVerifierData, 85
 - createClientCertVerifier, 86
 - createCryptoContext, 82
 - createRecvCryptoContext, 82
 - createSendCryptoContext, 82
 - deriveApplicationSecrets, 84
 - deriveEarlySecrets, 83
 - deriveHandshakeSecrets, 84
 - deriveLaterSecrets, 83
 - deriveUpdatedKeys, 85
 - deriveVerifierData, 85
 - incrementCryptoContext, 81
 - initCryptoContext, 81
 - initTranscriptHash, 79
 - recoverPSK, 83
 - rewindIO, 80
 - runningHash, 79
 - runningHashIO, 79
 - runningHashIOrewind, 80
 - runningSyntheticHash, 80
 - transcriptHash, 80
 - updateCryptoContext, 81
- tls_logger.h, 87
 - log, 88
 - logAlert, 90
 - logCert, 89
 - logCertDetails, 90
 - logCipherSuite, 91
 - logEncExt, 89
 - logKeyExchange, 91
 - logServerHello, 88
 - logServerResponse, 90
 - logSigAlg, 91
 - logTicket, 89
 - myprintf, 88
- TLS_MAX_CERT_B64

- tls1_3.h, [39](#)
- TLS_MAX_CERT_SIZE
 - tls1_3.h, [39](#)
- TLS_MAX_CIPHER_FRAG
 - tls1_3.h, [39](#)
- TLS_MAX_CIPHER_SUITES
 - tls1_3.h, [42](#)
- TLS_MAX_CLIENT_CHAIN_LEN
 - tls1_3.h, [40](#)
- TLS_MAX_CLIENT_CHAIN_SIZE
 - tls1_3.h, [41](#)
- TLS_MAX_COOKIE
 - tls1_3.h, [41](#)
- TLS_MAX_ECC_FIELD
 - tls1_3.h, [41](#)
- TLS_MAX_EXT_LABEL
 - tls1_3.h, [38](#)
- TLS_MAX_EXTENSIONS
 - tls1_3.h, [41](#)
- TLS_MAX_FRAG
 - tls1_3.h, [38](#)
- TLS_MAX_HASH
 - tls1_3.h, [38](#)
- TLS_MAX_HASH_STATE
 - tls1_3.h, [38](#)
- TLS_MAX_HELLO
 - tls1_3.h, [39](#)
- TLS_MAX_IO_SIZE
 - tls1_3.h, [39](#)
- TLS_MAX_IV_SIZE
 - tls1_3.h, [41](#)
- TLS_MAX_KEX_CIPHERTEXT_SIZE
 - tls1_3.h, [40](#)
- TLS_MAX_KEX_PUB_KEY_SIZE
 - tls1_3.h, [40](#)
- TLS_MAX_KEX_SECRET_KEY_SIZE
 - tls1_3.h, [40](#)
- TLS_MAX_KEY
 - tls1_3.h, [38](#)
- TLS_MAX_PLAIN_FRAG
 - tls1_3.h, [39](#)
- TLS_MAX_PSK_MODES
 - tls1_3.h, [42](#)
- TLS_MAX_SERVER_CHAIN_LEN
 - tls1_3.h, [40](#)
- TLS_MAX_SERVER_CHAIN_SIZE
 - tls1_3.h, [40](#)
- TLS_MAX_SERVER_NAME
 - tls1_3.h, [42](#)
- TLS_MAX_SHARED_SECRET_SIZE
 - tls1_3.h, [41](#)
- TLS_MAX_SIG_PUB_KEY_SIZE
 - tls1_3.h, [39](#)
- TLS_MAX_SIG_SECRET_KEY_SIZE
 - tls1_3.h, [40](#)
- TLS_MAX_SIGNATURE_SIZE
 - tls1_3.h, [40](#)
- TLS_MAX_SUPPORTED_GROUPS
 - tls1_3.h, [42](#)
- TLS_MAX_SUPPORTED_SIGS
 - tls1_3.h, [42](#)
- TLS_MAX_TAG_SIZE
 - tls1_3.h, [41](#)
- TLS_MAX_TICKET_SIZE
 - tls1_3.h, [41](#)
- tls_octads.h, [91](#)
 - OCT_append_byte, [95](#)
 - OCT_append_bytes, [95](#)
 - OCT_append_int, [93](#)
 - OCT_append_octad, [93](#)
 - OCT_append_string, [95](#)
 - OCT_compare, [93](#)
 - OCT_copy, [96](#)
 - OCT_from_base64, [96](#)
 - OCT_from_hex, [94](#)
 - OCT_kill, [94](#)
 - OCT_output_base64, [97](#)
 - OCT_output_hex, [97](#)
 - OCT_output_string, [97](#)
 - OCT_reverse, [96](#)
 - OCT_shift_left, [94](#)
 - OCT_truncate, [96](#)
- tls_protocol.h, [98](#)
 - TLS13_clean, [100](#)
 - TLS13_connect, [99](#)
 - TLS13_end, [99](#)
 - TLS13_rcv, [100](#)
 - TLS13_send, [100](#)
 - TLS13_start, [99](#)
- TLS_RESUMPTION_REQUIRED
 - tls1_3.h, [54](#)
- tls_sal.h, [101](#)
 - SAL_aeadDecrypt, [110](#)
 - SAL_aeadEncrypt, [109](#)
 - SAL_aeadKeylen, [106](#)
 - SAL_aeadTaglen, [106](#)
 - SAL_ciphers, [102](#)
 - SAL_generateKeyPair, [110](#)
 - SAL_generateSharedSecret, [111](#)
 - SAL_groups, [103](#)
 - SAL_hashInit, [108](#)
 - SAL_hashLen, [105](#)
 - SAL_hashNull, [108](#)
 - SAL_hashOutput, [109](#)
 - SAL_hashProcessArray, [109](#)
 - SAL_hashType, [105](#)
 - SAL_hkdfExpand, [107](#)
 - SAL_hkdfExtract, [107](#)
 - SAL_hmac, [108](#)
 - SAL_initLib, [105](#)
 - SAL_name, [102](#)
 - SAL_randomByte, [106](#)
 - SAL_randomOctad, [107](#)
 - SAL_sigCerts, [103](#)
 - SAL_sigs, [103](#)
 - SAL_tlsSignature, [112](#)

- SAL_tlsSignatureVerify, 111
- TLS_session, 26
 - cipher_suite, 28
 - CTS, 29
 - cts, 29
 - favourite_group, 28
 - hostname, 28
 - HS, 28
 - hs, 28
 - id, 27
 - IO, 29
 - io, 29
 - K_rcv, 28
 - K_send, 28
 - max_record, 27
 - ptr, 29
 - RMS, 28
 - rms, 29
 - sockptr, 27
 - status, 27
 - STS, 29
 - sts, 29
 - T, 30
 - tlshash, 30
- TLS_SHA256_T
 - tls1_3.h, 37
- TLS_SHA384_T
 - tls1_3.h, 37
- TLS_SHA512_T
 - tls1_3.h, 38
- tls_sockets.h, 112
 - getByte, 116
 - getBytes, 115
 - getInt16, 115
 - getInt24, 115
 - getIPaddress, 114
 - getOctad, 116
 - sendLen, 114
 - sendOctad, 114
 - setclientsock, 113
- TLS_SUCCESS
 - tls1_3.h, 54
- tls_tickets.h, 117
 - endTicketContext, 118
 - initTicketContext, 118
 - millis, 117
 - parseTicket, 117
 - ticket_still_good, 118
- TLS_VER
 - tls1_3.h, 47
- tls_wifi.h, 119
- tls_x509.h, 119
 - USE_C25519, 121
 - USE_NIST256, 121
 - USE_NIST384, 121
 - USE_NIST521, 122
 - X509_AN, 128
 - X509_BC, 128
 - X509_CN, 127
 - X509_ECC, 120
 - X509_ECD, 121
 - X509_EN, 127
 - X509_extract_cert, 123
 - X509_extract_cert_sig, 122
 - X509_extract_private_key, 122
 - X509_extract_public_key, 123
 - X509_find_alt_name, 126
 - X509_find_entity_property, 125
 - X509_find_expiry_date, 125
 - X509_find_extension, 126
 - X509_find_extensions, 126
 - X509_find_issuer, 123
 - X509_find_start_date, 125
 - X509_find_subject, 124
 - X509_find_validity, 124
 - X509_H256, 121
 - X509_H384, 121
 - X509_H512, 121
 - X509_KU, 128
 - X509_LN, 127
 - X509_MN, 128
 - X509_ON, 127
 - X509_PQ, 121
 - X509_RSA, 120
 - X509_self_signed, 124
 - X509_SN, 128
 - X509_UN, 127
- TLS_X509_MAX_FIELD
 - tls1_3.h, 38
- tlshash
 - TLS_session, 30
- transcriptHash
 - tls_keys_calc.h, 80
- TRY_EARLY_DATA
 - tls1_3.h, 37
- type
 - pktype, 22
- TYPICAL
 - tls1_3.h, 35
- UNEXPECTED_MESSAGE
 - tls1_3.h, 52
- unihash, 30
 - htype, 30
 - state, 30
- UNKNOWN_CA
 - tls1_3.h, 53
- UNRECOGNIZED_EXT
 - tls1_3.h, 50
- unsign32
 - tls1_3.h, 55
- unsign64
 - tls1_3.h, 55
- UNSUPPORTED_EXTENSION
 - tls1_3.h, 53
- updateCryptoContext
 - tls_keys_calc.h, 81

USE_C25519
 tls_x509.h, [121](#)
USE_NIST256
 tls_x509.h, [121](#)
USE_NIST384
 tls_x509.h, [121](#)
USE_NIST521
 tls_x509.h, [122](#)

val
 octad, [21](#)
 ret, [23](#)
valid
 ticket, [24](#)
VERBOSITY
 tls1_3.h, [36](#)

WRONG_MESSAGE
 tls1_3.h, [50](#)

X25519
 tls1_3.h, [43](#)
X448
 tls1_3.h, [43](#)
X509_AN
 tls_x509.h, [128](#)
X509_BC
 tls_x509.h, [128](#)
X509_CN
 tls_x509.h, [127](#)
X509_ECC
 tls_x509.h, [120](#)
X509_ECD
 tls_x509.h, [121](#)
X509_EN
 tls_x509.h, [127](#)
X509_extract_cert
 tls_x509.h, [123](#)
X509_extract_cert_sig
 tls_x509.h, [122](#)
X509_extract_private_key
 tls_x509.h, [122](#)
X509_extract_public_key
 tls_x509.h, [123](#)
X509_find_alt_name
 tls_x509.h, [126](#)
X509_find_entity_property
 tls_x509.h, [125](#)
X509_find_expiry_date
 tls_x509.h, [125](#)
X509_find_extension
 tls_x509.h, [126](#)
X509_find_extensions
 tls_x509.h, [126](#)
X509_find_issuer
 tls_x509.h, [123](#)
X509_find_start_date
 tls_x509.h, [125](#)
X509_find_subject
 tls_x509.h, [124](#)
X509_find_validity
 tls_x509.h, [124](#)
X509_H256
 tls_x509.h, [121](#)
X509_H384
 tls_x509.h, [121](#)
X509_H512
 tls_x509.h, [121](#)
X509_KU
 tls_x509.h, [128](#)
X509_LN
 tls_x509.h, [127](#)
X509_MN
 tls_x509.h, [128](#)
X509_ON
 tls_x509.h, [127](#)
X509_PQ
 tls_x509.h, [121](#)
X509_RSA
 tls_x509.h, [120](#)
X509_self_signed
 tls_x509.h, [124](#)
X509_SN
 tls_x509.h, [128](#)
X509_UN
 tls_x509.h, [127](#)