

# TII TLS1.3

1.1

Generated by Doxygen 1.9.1



<b>1 Description</b>	<b>1</b>
1.1 Client side Authentication	4
1.2 Testing Pre-shared keys	5
1.2.1 How to use it	5
1.2.1.1 Localhost 4433	5
1.2.1.2 Just Host	5
1.2.1.3 Host and port	5
1.2.1.4 AF_UNIX Socket	5
1.2.2 Building the client application on an Arduino board	5
<b>2 Data Structure Index</b>	<b>7</b>
2.1 Data Structures	7
<b>3 File Index</b>	<b>9</b>
3.1 File List	9
<b>4 Data Structure Documentation</b>	<b>11</b>
4.1 capabilities Struct Reference	11
4.1.1 Detailed Description	11
4.1.2 Field Documentation	11
4.1.2.1 nsg	11
4.1.2.2 supportedGroups	12
4.1.2.3 nsc	12
4.1.2.4 ciphers	12
4.1.2.5 nsa	12
4.1.2.6 sigAlgs	12
4.1.2.7 nsac	12
4.1.2.8 sigAlgsCert	12
4.2 crypto Struct Reference	13
4.2.1 Detailed Description	13
4.2.2 Field Documentation	13
4.2.2.1 k	13
4.2.2.2 iv	13
4.2.2.3 K	13
4.2.2.4 IV	13
4.2.2.5 record	14
4.2.2.6 suite	14
4.3 ee_expt Struct Reference	14
4.3.1 Detailed Description	14
4.3.2 Field Documentation	14
4.3.2.1 early_data	14
4.3.2.2 alpn	15
4.3.2.3 server_name	15

4.3.2.4 max_frag_length	15
4.4 ee_resp Struct Reference	15
4.4.1 Detailed Description	15
4.4.2 Field Documentation	15
4.4.2.1 early_data	16
4.4.2.2 alpn	16
4.4.2.3 server_name	16
4.4.2.4 max_frag_length	16
4.5 octad Struct Reference	16
4.5.1 Detailed Description	16
4.5.2 Field Documentation	17
4.5.2.1 len	17
4.5.2.2 max	17
4.5.2.3 val	17
4.6 pktype Struct Reference	17
4.6.1 Detailed Description	17
4.6.2 Field Documentation	18
4.6.2.1 type	18
4.6.2.2 hash	18
4.6.2.3 curve	18
4.7 ret Struct Reference	18
4.7.1 Detailed Description	18
4.7.2 Field Documentation	18
4.7.2.1 val	19
4.7.2.2 err	19
4.8 Socket Class Reference	19
4.8.1 Detailed Description	19
4.9 ticket Struct Reference	20
4.9.1 Detailed Description	20
4.9.2 Field Documentation	20
4.9.2.1 tick	20
4.9.2.2 nonce	20
4.9.2.3 TICK	20
4.9.2.4 NONCE	21
4.9.2.5 lifetime	21
4.9.2.6 age_obfuscator	21
4.9.2.7 max_early_data	21
4.9.2.8 birth	21
4.9.2.9 cipher_suite	21
4.9.2.10 favourite_group	21
4.10 unihash Struct Reference	22
4.10.1 Detailed Description	22

---

4.10.2 Field Documentation . . . . .	22
4.10.2.1 state . . . . .	22
4.10.2.2 htype . . . . .	22
<b>5 File Documentation</b>	<b>23</b>
5.1 tls1_3.h File Reference . . . . .	23
5.1.1 Detailed Description . . . . .	26
5.1.2 Macro Definition Documentation . . . . .	26
5.1.2.1 IO_NONE . . . . .	26
5.1.2.2 IO_APPLICATION . . . . .	26
5.1.2.3 IO_PROTOCOL . . . . .	26
5.1.2.4 IO_DEBUG . . . . .	26
5.1.2.5 IO_WIRE . . . . .	26
5.1.2.6 TLS_HTTP_PROTOCOL . . . . .	26
5.1.2.7 VERBOSITY . . . . .	27
5.1.2.8 THIS_YEAR . . . . .	27
5.1.2.9 HAVE_A_CLIENT_CERT . . . . .	27
5.1.2.10 TLS_PROTOCOL . . . . .	27
5.1.2.11 TLS_AES_128 . . . . .	27
5.1.2.12 TLS_AES_256 . . . . .	27
5.1.2.13 TLS_CHA_256 . . . . .	27
5.1.2.14 TLS_MAX_HASH_STATE . . . . .	27
5.1.2.15 TLS_MAX_HASH . . . . .	28
5.1.2.16 TLS_MAX_KEY . . . . .	28
5.1.2.17 TLS_X509_MAX_FIELD . . . . .	28
5.1.2.18 TLS_MAX_ROOT_CERT_SIZE . . . . .	28
5.1.2.19 TLS_MAX_ROOT_CERT_B64 . . . . .	28
5.1.2.20 TLS_MAX_MYCERT_SIZE . . . . .	28
5.1.2.21 TLS_MAX_MYCERT_B64 . . . . .	28
5.1.2.22 TLS_MAX_CLIENT_HELLO . . . . .	28
5.1.2.23 TLS_MAX_EXT_LABEL . . . . .	29
5.1.2.24 TLS_MAX_TICKET_SIZE . . . . .	29
5.1.2.25 TLS_MAX_EXTENSIONS . . . . .	29
5.1.2.26 TLS_MAX_IO_SIZE . . . . .	29
5.1.2.27 TLS_MAX_SIGNATURE_SIZE . . . . .	29
5.1.2.28 TLS_MAX_PUB_KEY_SIZE . . . . .	29
5.1.2.29 TLS_MAX_SECRET_KEY_SIZE . . . . .	29
5.1.2.30 TLS_MAX_ECC_FIELD . . . . .	30
5.1.2.31 TLS_IV_SIZE . . . . .	30
5.1.2.32 TLS_TAG_SIZE . . . . .	30
5.1.2.33 TLS_MAX_COOKIE . . . . .	30
5.1.2.34 TLS_MAX_SERVER_NAME . . . . .	30

5.1.2.35 TLS_MAX_SUPPORTED_GROUPS . . . . .	30
5.1.2.36 TLS_MAX_SUPPORTED_SIGS . . . . .	30
5.1.2.37 TLS_MAX_PSK_MODES . . . . .	30
5.1.2.38 TLS_MAX_CIPHER_SUITES . . . . .	31
5.1.2.39 TLS_AES_128_GCM_SHA256 . . . . .	31
5.1.2.40 TLS_AES_256_GCM_SHA384 . . . . .	31
5.1.2.41 TLS_CHACHA20_POLY1305_SHA256 . . . . .	31
5.1.2.42 X25519 . . . . .	31
5.1.2.43 SECP256R1 . . . . .	31
5.1.2.44 SECP384R1 . . . . .	31
5.1.2.45 ECDSA_SECP256R1_SHA256 . . . . .	31
5.1.2.46 ECDSA_SECP384R1_SHA384 . . . . .	32
5.1.2.47 RSA_PSS_RSAE_SHA256 . . . . .	32
5.1.2.48 RSA_PSS_RSAE_SHA384 . . . . .	32
5.1.2.49 RSA_PSS_RSAE_SHA512 . . . . .	32
5.1.2.50 RSA_PKCS1_SHA256 . . . . .	32
5.1.2.51 RSA_PKCS1_SHA384 . . . . .	32
5.1.2.52 RSA_PKCS1_SHA512 . . . . .	32
5.1.2.53 ED25519 . . . . .	32
5.1.2.54 PSKOK . . . . .	33
5.1.2.55 PSKWECDHE . . . . .	33
5.1.2.56 TLS1_0 . . . . .	33
5.1.2.57 TLS1_2 . . . . .	33
5.1.2.58 TLS1_3 . . . . .	33
5.1.2.59 SERVER_NAME . . . . .	33
5.1.2.60 SUPPORTED_GROUPS . . . . .	33
5.1.2.61 SIG_ALGS . . . . .	33
5.1.2.62 SIG_ALGS_CERT . . . . .	34
5.1.2.63 KEY_SHARE . . . . .	34
5.1.2.64 PSK_MODE . . . . .	34
5.1.2.65 PRESERVED_KEY . . . . .	34
5.1.2.66 TLS_VER . . . . .	34
5.1.2.67 COOKIE . . . . .	34
5.1.2.68 EARLY_DATA . . . . .	34
5.1.2.69 MAX_FRAG_LENGTH . . . . .	34
5.1.2.70 PADDING . . . . .	35
5.1.2.71 APP_PROTOCOL . . . . .	35
5.1.2.72 HSHAKE . . . . .	35
5.1.2.73 APPLICATION . . . . .	35
5.1.2.74 ALERT . . . . .	35
5.1.2.75 CHANGE_CIPHER . . . . .	35
5.1.2.76 TIME_OUT . . . . .	35

5.1.2.77 HANDSHAKE_RETRY . . . . .	36
5.1.2.78 STRANGE_EXTENSION . . . . .	36
5.1.2.79 CLIENT_HELLO . . . . .	36
5.1.2.80 SERVER_HELLO . . . . .	36
5.1.2.81 CERTIFICATE . . . . .	36
5.1.2.82 CERT_REQUEST . . . . .	36
5.1.2.83 CERT_VERIFY . . . . .	36
5.1.2.84 FINISHED . . . . .	36
5.1.2.85 ENCRYPTED_EXTENSIONS . . . . .	37
5.1.2.86 TICKET . . . . .	37
5.1.2.87 KEY_UPDATE . . . . .	37
5.1.2.88 MESSAGE_HASH . . . . .	37
5.1.2.89 END_OF_EARLY_DATA . . . . .	37
5.1.2.90 NOT_TLS1_3 . . . . .	37
5.1.2.91 BAD_CERT_CHAIN . . . . .	37
5.1.2.92 ID_MISMATCH . . . . .	37
5.1.2.93 UNRECOGNIZED_EXT . . . . .	38
5.1.2.94 BAD_HELLO . . . . .	38
5.1.2.95 WRONG_MESSAGE . . . . .	38
5.1.2.96 MISSING_REQUEST_CONTEXT . . . . .	38
5.1.2.97 AUTHENTICATION_FAILURE . . . . .	38
5.1.2.98 BAD_RECORD . . . . .	38
5.1.2.99 BAD_TICKET . . . . .	38
5.1.2.100 NOT_EXPECTED . . . . .	38
5.1.2.101 CA_NOT_FOUND . . . . .	39
5.1.2.102 ILLEGAL_PARAMETER . . . . .	39
5.1.2.103 UNEXPECTED_MESSAGE . . . . .	39
5.1.2.104 DECRYPT_ERROR . . . . .	39
5.1.2.105 BAD_CERTIFICATE . . . . .	39
5.1.2.106 UNSUPPORTED_EXTENSION . . . . .	39
5.1.2.107 UNKNOWN_CA . . . . .	39
5.1.3 Typedef Documentation . . . . .	39
5.1.3.1 byte . . . . .	40
5.1.3.2 sign8 . . . . .	40
5.1.3.3 sign16 . . . . .	40
5.1.3.4 sign32 . . . . .	40
5.1.3.5 sign64 . . . . .	40
5.1.3.6 unsign32 . . . . .	40
5.1.3.7 unsign64 . . . . .	40
5.2 tls_cacerts.h File Reference . . . . .	41
5.2.1 Detailed Description . . . . .	41
5.2.2 Variable Documentation . . . . .	41

5.2.2.1 myprivate . . . . .	41
5.2.2.2 mycert . . . . .	41
5.2.2.3 cacerts . . . . .	41
5.3 tls_cert_chain.h File Reference . . . . .	42
5.3.1 Detailed Description . . . . .	42
5.3.2 Macro Definition Documentation . . . . .	42
5.3.2.1 TLS_SHA256 . . . . .	42
5.3.2.2 TLS_SHA384 . . . . .	43
5.3.2.3 TLS_SHA512 . . . . .	43
5.3.3 Function Documentation . . . . .	43
5.3.3.1 CHECK_CERT_CHAIN() . . . . .	43
5.3.3.2 IS_SERVER_CERT_VERIFY() . . . . .	43
5.3.3.3 GET_CLIENT_KEY_AND_CERTCHAIN() . . . . .	44
5.3.3.4 CREATE_CLIENT_CERT_VERIFIER() . . . . .	44
5.4 tls_client_rcv.h File Reference . . . . .	45
5.4.1 Detailed Description . . . . .	46
5.4.2 Function Documentation . . . . .	46
5.4.2.1 parseoctad() . . . . .	46
5.4.2.2 parseInt16() . . . . .	46
5.4.2.3 parseInt24() . . . . .	47
5.4.2.4 parseInt32() . . . . .	47
5.4.2.5 parseByte() . . . . .	48
5.4.2.6 parseoctadptr() . . . . .	48
5.4.2.7 getServerFragment() . . . . .	48
5.4.2.8 parseByteorPull() . . . . .	49
5.4.2.9 parseInt32orPull() . . . . .	49
5.4.2.10 parseInt24orPull() . . . . .	50
5.4.2.11 parseInt16orPull() . . . . .	50
5.4.2.12 parseoctadorPull() . . . . .	51
5.4.2.13 parseoctadorPullptr() . . . . .	51
5.4.2.14 getWhatsNext() . . . . .	52
5.4.2.15 getServerEncryptedExtensions() . . . . .	52
5.4.2.16 getServerCertVerify() . . . . .	53
5.4.2.17 getServerFinished() . . . . .	53
5.4.2.18 getServerHello() . . . . .	54
5.4.2.19 getCheckServerCertificateChain() . . . . .	54
5.4.2.20 getCertificateRequest() . . . . .	55
5.5 tls_client_send.h File Reference . . . . .	55
5.5.1 Detailed Description . . . . .	57
5.5.2 Function Documentation . . . . .	57
5.5.2.1 sendCCCS() . . . . .	57
5.5.2.2 addPreSharedKeyExt() . . . . .	57



5.5.2.3 addServerNameExt()	58
5.5.2.4 addSupportedGroupsExt()	58
5.5.2.5 addSigAlgsExt()	58
5.5.2.6 addSigAlgsCertExt()	59
5.5.2.7 addKeyShareExt()	59
5.5.2.8 addALPNExt()	59
5.5.2.9 addMFLExt()	60
5.5.2.10 addPSKModesExt()	60
5.5.2.11 addVersionExt()	60
5.5.2.12 addPadding()	61
5.5.2.13 addCookieExt()	61
5.5.2.14 addEarlyDataExt()	61
5.5.2.15 clientRandom()	61
5.5.2.16 sessionID()	62
5.5.2.17 cipherSuites()	62
5.5.2.18 sendClientMessage()	63
5.5.2.19 sendBinder()	63
5.5.2.20 sendClientHello()	64
5.5.2.21 sendClientAlert()	64
5.5.2.22 sendClientFinish()	64
5.5.2.23 sendClientCertificateChain()	65
5.5.2.24 sendClientCertVerify()	65
5.5.2.25 sendEndOfEarlyData()	66
5.5.2.26 alert_from_cause()	66
5.6 tls_keys_calc.h File Reference	66
5.6.1 Detailed Description	67
5.6.2 Function Documentation	68
5.6.2.1 running_hash()	68
5.6.2.2 transcript_hash()	68
5.6.2.3 running_syn_hash()	68
5.6.2.4 init_crypto_context()	69
5.6.2.5 create_crypto_context()	69
5.6.2.6 increment_crypto_context()	69
5.6.2.7 GET_KEY_AND_IV()	69
5.6.2.8 RECOVER_PSK()	70
5.6.2.9 GET_EARLY_SECRET()	70
5.6.2.10 GET_LATER_SECRETS()	71
5.6.2.11 GET_HANDSHAKE_SECRETS()	71
5.6.2.12 GET_APPLICATION_SECRETS()	72
5.6.2.13 UPDATE_KEYS()	72
5.6.2.14 IS_VERIFY_DATA()	72
5.6.2.15 VERIFY_DATA()	74

5.7 <code>tls_logger.h</code> File Reference	74
5.7.1 Detailed Description	75
5.7.2 Function Documentation	75
5.7.2.1 <code>myprintf()</code>	75
5.7.2.2 <code>logger()</code>	76
5.7.2.3 <code>logServerHello()</code>	76
5.7.2.4 <code>logTicket()</code>	76
5.7.2.5 <code>logEncExt()</code>	77
5.7.2.6 <code>logCert()</code>	77
5.7.2.7 <code>logCertDetails()</code>	77
5.7.2.8 <code>logServerResponse()</code>	79
5.7.2.9 <code>logAlert()</code>	79
5.7.2.10 <code>logCipherSuite()</code>	79
5.7.2.11 <code>logKeyExchange()</code>	80
5.7.2.12 <code>logSigAlg()</code>	80
5.8 <code>tls_octads.h</code> File Reference	80
5.8.1 Detailed Description	81
5.8.2 Function Documentation	81
5.8.2.1 <code>OCT_append_int()</code>	81
5.8.2.2 <code>OCT_append_octad()</code>	82
5.8.2.3 <code>OCT_compare()</code>	82
5.8.2.4 <code>OCT_shift_left()</code>	82
5.8.2.5 <code>OCT_kill()</code>	83
5.8.2.6 <code>OCT_from_hex()</code>	83
5.8.2.7 <code>OCT_append_string()</code>	83
5.8.2.8 <code>OCT_append_byte()</code>	84
5.8.2.9 <code>OCT_append_bytes()</code>	84
5.8.2.10 <code>OCT_from_base64()</code>	84
5.8.2.11 <code>OCT_reverse()</code>	85
5.8.2.12 <code>OCT_truncate()</code>	85
5.8.2.13 <code>OCT_copy()</code>	85
5.8.2.14 <code>OCT_output_hex()</code>	85
5.8.2.15 <code>OCT_output_string()</code>	86
5.8.2.16 <code>OCT_output_base64()</code>	86
5.9 <code>tls_protocol.h</code> File Reference	87
5.9.1 Detailed Description	87
5.9.2 Function Documentation	87
5.9.2.1 <code>TLS13_full()</code>	87
5.9.2.2 <code>TLS13_resume()</code>	88
5.10 <code>tls_sal.h</code> File Reference	88
5.10.1 Detailed Description	90
5.10.2 Function Documentation	90

5.10.2.1 TLS_SAL_CIPHERS()	90
5.10.2.2 TLS_SAL_GROUPS()	90
5.10.2.3 TLS_SAL_SIGS()	91
5.10.2.4 TLS_SAL_SIGCERTS()	91
5.10.2.5 TLS_SAL_INITLIB()	91
5.10.2.6 TLS_SAL_HASHTYPE()	91
5.10.2.7 TLS_SAL_HASHLEN()	92
5.10.2.8 TLS_RANDOM_BYTE()	92
5.10.2.9 TLS_RANDOM_OCTAD()	92
5.10.2.10 TLS_HKDF_Extract()	93
5.10.2.11 TLS_HKDF_Expand_Label()	93
5.10.2.12 TLS_HMAC()	94
5.10.2.13 TLS_HASH_NULL()	94
5.10.2.14 Hash_Init()	94
5.10.2.15 Hash_Process()	95
5.10.2.16 Hash_Output()	95
5.10.2.17 AEAD_ENCRYPT()	95
5.10.2.18 AEAD_DECRYPT()	96
5.10.2.19 GENERATE_KEY_PAIR()	96
5.10.2.20 GENERATE_SHARED_SECRET()	97
5.10.2.21 CERT_SIGNATURE_VERIFY()	97
5.10.2.22 TLS_SIGNATURE_VERIFY()	97
5.10.2.23 TLS_SIGNATURE_SIGN()	98
5.11 tls_sockets.h File Reference	98
5.11.1 Detailed Description	99
5.11.2 Function Documentation	99
5.11.2.1 setclientsock()	99
5.11.2.2 getIpAddress()	100
5.11.2.3 sendOctad()	100
5.11.2.4 sendLen()	101
5.11.2.5 getBytes()	101
5.11.2.6 getInt16()	101
5.11.2.7 getInt24()	102
5.11.2.8 getByte()	102
5.11.2.9 getOctad()	102
5.12 tls_tickets.h File Reference	103
5.12.1 Detailed Description	103
5.12.2 Function Documentation	103
5.12.2.1 millis()	103
5.12.2.2 parseTicket()	104
5.12.2.3 init_ticket_context()	104
5.13 tls_wifi.h File Reference	104

5.13.1 Detailed Description	104
5.14 tls_x509.h File Reference	105
5.14.1 Detailed Description	106
5.14.2 Macro Definition Documentation	106
5.14.2.1 X509_ECC	106
5.14.2.2 X509_RSA	106
5.14.2.3 X509_ECD	106
5.14.2.4 X509_H256	106
5.14.2.5 X509_H384	106
5.14.2.6 X509_H512	107
5.14.2.7 USE_NIST256	107
5.14.2.8 USE_C25519	107
5.14.2.9 USE_NIST384	107
5.14.2.10 USE_NIST521	107
5.14.3 Function Documentation	107
5.14.3.1 X509_extract_private_key()	107
5.14.3.2 X509_extract_cert_sig()	108
5.14.3.3 X509_extract_cert()	108
5.14.3.4 X509_extract_public_key()	108
5.14.3.5 X509_find_issuer()	109
5.14.3.6 X509_find_validity()	109
5.14.3.7 X509_find_subject()	109
5.14.3.8 X509_self_signed()	110
5.14.3.9 X509_find_entity_property()	110
5.14.3.10 X509_find_start_date()	111
5.14.3.11 X509_find_expiry_date()	111
5.14.3.12 X509_find_extensions()	111
5.14.3.13 X509_find_extension()	112
5.14.3.14 X509_find_alt_name()	112
5.14.4 Variable Documentation	112
5.14.4.1 X509_CN	112
5.14.4.2 X509_ON	113
5.14.4.3 X509_EN	113
5.14.4.4 X509_LN	113
5.14.4.5 X509_UN	113
5.14.4.6 X509_MN	113
5.14.4.7 X509_SN	113
5.14.4.8 X509_AN	113
5.14.4.9 X509_KU	113
5.14.4.10 X509_BC	113

# Chapter 1

## Description

UPDATE: The Crypto support functions are now all concentrated in the `tls_sal_*.xpp` files. This will make it easier to use alternate crypto providers.

This C++ version is really just C plus namespaces plus pass-by-reference. These the only features of C++ that are used. The Rust version will come later. Documentation can be found in the doxygen generated file `refman.pdf`

First inside a working directory build the C++ version of MIRACL core ( <https://github.com/miracl/core>), selecting support for C25519, NIST256, NIST384, RSA2048 and RSA4096.

This library 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.

Then copy the contents of this archive to the same directory, in particular `client.cpp` and `tls*.*`

Set the verbosity of the output in `tls1_3.h` to `IO_DEBUG`.

Decide which crypto providers to use.

If using only the `miracl` library

```
cp tls_sal_m.xpp tls_sal.cpp
```

If using `miracl+libsodium`

```
cp tls_sal_ms.xpp tls_sal.cpp
```

If using `miracl+tiicrypto`

```
cp tls_sal_mt.xpp tls_sal.cpp
```

Build the `tls` library and the client app by

```
g++ -O2 -c tls*.cpp
ar rc tls.a tls_protocol.o tls_keys_calc.o tls_sockets.o tls_cert_chain.o tls_client_recv.o tls_client_send.o
```

If using `miracl` only

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

If using miracl+libsodium

```
g++ -O2 client.cpp tls.a core.a -lsodium -o client
```

If using miracl+TlIcrypto

```
g++ -O2 client.cpp tls.a core.a libtiicrypto-v2.3.0.a -o client
```

Or by using CMake. If you follow this alternative, copy the header files into `vendor/miracl/includes`, and the `core.a` to `vendor/miracl/`

The test TLS client app tries to connect to a Web site, drops the connections, and then attempts a resumption. Execute the client process as for example

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

The output should look something like

```
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= 020000970303267B440A8FDD5AAFCCE89BCCC40729F090B688743E160AB6F4F00CC9D8A077B7720A0E658C6A5BB912768
Parsing serverHello
Cipher Suite is TLS_AES_128_GCM_SHA256
Key Exchange Group is SECP256R1
Server Public Key= 0445F74493E090E5C6F9734E753181615E70F262773B2631ED648107E42D8D7C8CA276254781BA448C15873668E

Shared Secret= 720BAE2BC1B3E57CBBDB7FCBE67EAE9D53A418FC447FE5FAEE6CF03CD96F46AC
Handshake Secret= D8B86E75D506C134176EDDFA7BA97895206FFB5628FF2D8BC303708E622F6CF9
Client handshake traffic secret= 67E1C0882706014596274D5A34E338D84BCE68D96E4DEEFF29840D62F053569E
Server handshake traffic secret= 83B7A088247EB6EFB78B1A02315DB241594B40C1F561572ED97D294FBEB6D1E5
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 0A5C155DBDD9F7F6ABE005D351D6E3FF9DEBA799F7479BD33E1C784B63CB4CA695A76815C9B666C24B6E989EE85009A6
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 = 0A5C155DBDD9F7F6ABE005D351D6E3FF9DEBA799F7479BD33E1C784B63CB4CA695A76815C9B666C24B6E989EE85009A6
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) = 6598500DBD640A84AD24325C5D730BDE380365598A7960264597A5A02327879C  
Transcript Hash (CH+SH+EE+SCT+SCV) = 9AAF4F6B618D4F0F841F1757E4A6245CF05C2561B5E6BB84943BA337AD382CA3  
Server Certificate Signature= 1876F917CC0580BD9E2BFDBC9D693FD3B286DBD9065B6EC29C04A860A0E61882AB436B57984A0F0  
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]) = 860D9C88942778F7F83D3DFAC9D2B347C3BABF9BCBA1A3EAB966E03806C6  
Client Verify Data= CD0D6D333B0C52D179A0C3008B9E13F16CCD3DB7365FDFBA2D64AF0D2EC66640  
Transcript Hash (CH+SH+EE+SCT+SCV+SF+[CCT+CSV]+CF) = 7F9ADB9127FAAF7C4698BEF3DB26227B88FD3D0844C529DFBBAA8C913  
Client application traffic secret= 28570358B3B6BD371F8E6D067733350EA4AD47890A70CC5DC5CE6BCCFD0A2ECF  
Server application traffic secret= 42F6933BFFB06207BAFFAA810CE7C84EA1AE04001A9F2373225DE1BADB07E8F0  
Full Handshake succeeded  
... 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) = 485454502F312E3120323030204F4B0D0A5365727665723A205377696674544C  
Alert sent to Server - Close notify  
Connection closed

Attempting resumption

Parsing Ticket  
life time in minutes = 60  
Age obfuscator = 6a84f357  
Nonce = 00  
Ticket = B89AA26EED944AAB8C8B48293ACED3DF806DDA0A64796C865A2FDAF8CCD0E284  
max\_early\_data = 40960

PSK= 74B636463CEBE875B5CADEBC1FB022810B6EA6984FA58E957795E9979D74BF29  
Binder Key= 60C2EC05E424A9033530B4A4644CE5BF893AADBEFC50B69962A1552CD2EAA47E  
Early Secret= 5D518801BE5D23F8A341C6ACDC134797594E2642735B552C38E9F0B075BE484D  
Private key= BE24A6B02F166305D4B3A5CC644BAFD31EADDDF28EA783EA5850FB046D230E98  
Client Public key= 04B593D22237E22583533A37898CDB74D86D500CAD7BCE59EF32F0166D977C3E76A23DF81121310093635CFE9C1  
Ticket age= 38  
obfuscated age = 6a84f38f  
Client Hello sent  
BND= 4F4856443D2B30D07AC0639C29365FA9A3BB2B05E1DA9E991D7A8AC4A400888C  
Sending Binders  
Client Early Traffic Secret= 54D871D0579263A798C71620748FE34805C3F300C71B3598C9D4C089F2B59BBF  
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= 042E45D04810BC6A16EDE859EE44284CAD0C46FC63D9802E628F4D2AFFE55092CFEA64B91B33A3EC49ABC3B2505  
serverHello= 0200009D0303267B440B183D03A4C84BCD0219A0E2990DBCA4723510F88F3FD2870CA369900D20898C827DD669FF948C  
Shared Secret= C9C6823FE64A2A0E28525C59654ACEEEF42C59E2B2AD6216703A7B747C8C1D31  
Handshake Secret= E961F5E140CD939D2CF18E1BC7AF2995D6D60CF495CA4EA7D86B92D86FE4F046  
Client handshake traffic secret= FBFA2AE7DC1A3EB3FF23AE969C8EED24EE802DE467AB9B72F9CED1201AB6A9C2  
Server handshake traffic secret= E2E5BFA257DD140CA74232DB5048C643E40239C07929D0AC34A54316A47605C8  
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) = 0C3AE9A9DE38AC962674A532D90F3623774DD2E7D3C2A4704429060206DD29DF
Send End of Early Data
Transcript Hash (CH+SH+EE+SF+ED) = 1E536FCCA58F32F7061A38748577ECA163D6BA5455CBDAD923E69FF39E26AA11
Server Data is verified
Client Verify Data= 51A23C4E4EA8F3D7600C3C406647B3A4F13D6398A829455AEBFB0A3F867D04E7
Client application traffic secret= ACD82D6A9F00258A012191DA0CD976D218C96DFF67D7FA5D31E69A099B32A333
Server application traffic secret= B427AC41917E466E85049DBCEA36B98098F748D224D743D32FDD3E05602327B4
Resumption Handshake succeeded
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 list.txt for some websites that work OK and test different functionality.

## 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 secp384r1) -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!



## 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 psk
```

### 1.2.1 How to use it

#### 1.2.1.1 Localhost 4433

This is our own server, using TLSSwift (localhost:4433)

```
./client
```

#### 1.2.1.2 Just Host

```
./client tls13.ld.pw
```

#### 1.2.1.3 Host and port

```
./client localhost:1234
```

#### 1.2.1.4 AF\_UNIX Socket

```
./client af_unix /tmp/somesocket
```

## 1.2.2 Building the client application on an Arduino board (like ESP32)

1. Create working directory directory with name NAME
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 the files config.py, client.cpp and tls\*.\* from this directory to the working directory
6. Edit the file core.h to define CORE\_ARDUINO
7. Edit the file [tls1\\_3.h](#) to define POPULAR\_ROOT\_CERTS and TLS\_ARDUINO
8. Edit the file client.cpp to use your wifi SSID and password (near line 170)
9. Run py config.py, and select options 2,3,8,41 and 43
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 library directory off that.)
12. Open the Arduino app, and look in File->Examples->NAME, and look for the example "client"
13. Upload to the board and run it! Tools->Serial Monitor to see the output



## Chapter 2

# Data Structure Index

### 2.1 Data Structures

Here are the data structures with brief descriptions:

<a href="#">capabilities</a>	Cryptographic capabilities of the client . . . . .	11
<a href="#">crypto</a>	Crypto context structure . . . . .	13
<a href="#">ee_expt</a>	Server encrypted extensions expectations . . . . .	14
<a href="#">ee_resp</a>	Server encrypted extensions responses . . . . .	15
<a href="#">octad</a>	Safe representation of an octad . . . . .	16
<a href="#">pktype</a>	Public key type . . . . .	17
<a href="#">ret</a>	Function return structure . . . . .	18
<a href="#">Socket</a>	<a href="#">Socket</a> instance . . . . .	19
<a href="#">ticket</a>	Ticket context structure . . . . .	20
<a href="#">unihash</a>	Universal Hash structure . . . . .	22



## Chapter 3

# File Index

### 3.1 File List

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

<a href="#">tls1_3.h</a>	Main TLS 1.3 Header File for constants and structures . . . . .	23
<a href="#">tls_cacerts.h</a>	Certificate Authority root certificate store . . . . .	41
<a href="#">tls_cert_chain.h</a>	Process Certificate Chain . . . . .	42
<a href="#">tls_client_rcv.h</a>	Process Input received from the Server . . . . .	45
<a href="#">tls_client_send.h</a>	Process Output to be sent to the Server . . . . .	55
<a href="#">tls_keys_calc.h</a>	TLS 1.3 crypto support functions . . . . .	66
<a href="#">tls_logger.h</a>	TLS 1.3 logging . . . . .	74
<a href="#">tls_octads.h</a>	Octad handling routines - octads don't overflow, they truncate . . . . .	80
<a href="#">tls_protocol.h</a>	TLS 1.3 main client-side protocol functions . . . . .	87
<a href="#">tls_sal.h</a>	Security Abstraction Layer for TLS . . . . .	88
<a href="#">tls_sockets.h</a>	Set up sockets for reading and writing . . . . .	98
<a href="#">tls_tickets.h</a>	TLS 1.3 process resumption tickets . . . . .	103
<a href="#">tls_wifi.h</a>	Define <a href="#">Socket</a> structure depending on processor context . . . . .	104
<a href="#">tls_x509.h</a>	X509 function Header File . . . . .	105



## Chapter 4

# Data Structure Documentation

### 4.1 capabilities Struct Reference

Cryptographic capabilities of the client.

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

#### Data Fields

- int [nsg](#)
- int [supportedGroups](#) [TLS\_MAX\_SUPPORTED\_GROUPS]
- int [nsc](#)
- int [ciphers](#) [TLS\_MAX\_CIPHER\_SUITES]
- int [nsa](#)
- int [sigAlgs](#) [TLS\_MAX\_SUPPORTED\_SIGS]
- int [nsac](#)
- int [sigAlgsCert](#) [TLS\_MAX\_SUPPORTED\_SIGS]

#### 4.1.1 Detailed Description

Cryptographic capabilities of the client.

#### 4.1.2 Field Documentation

##### 4.1.2.1 nsg

```
int capabilities::nsg
```

Number of supported groups

#### 4.1.2.2 supportedGroups

```
int capabilities::supportedGroups[TLS_MAX_SUPPORTED_GROUPS]
```

Supported groups

#### 4.1.2.3 nsc

```
int capabilities::nsc
```

Number of supported cipher suites

#### 4.1.2.4 ciphers

```
int capabilities::ciphers[TLS_MAX_CIPHER_SUITES]
```

Supported cipher suites

#### 4.1.2.5 nsa

```
int capabilities::nsa
```

Number of supported signature algorithms for TLS 1.3

#### 4.1.2.6 sigAlgs

```
int capabilities::sigAlgs[TLS_MAX_SUPPORTED_SIGS]
```

Supported signature algorithms for TLS1.3

#### 4.1.2.7 nsac

```
int capabilities::nsac
```

Number of supported signature algorithms for Certificates

#### 4.1.2.8 sigAlgsCert

```
int capabilities::sigAlgsCert[TLS_MAX_SUPPORTED_SIGS]
```

Supported signature algorithms for Certificates

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

- [tls1\\_3.h](#)



## 4.2 crypto Struct Reference

crypto context structure

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

### Data Fields

- char [k](#) [[TLS\\_MAX\\_KEY](#)]
- char [iv](#) [12]
- [octad](#) [K](#)
- [octad](#) [IV](#)
- [unsign32](#) [record](#)
- int [suite](#)

### 4.2.1 Detailed Description

crypto context structure

### 4.2.2 Field Documentation

#### 4.2.2.1 k

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

AEAD cryptographic Key bytes

#### 4.2.2.2 iv

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

AEAD cryptographic IV bytes

#### 4.2.2.3 K

```
octad crypto::K
```

Key as octad

#### 4.2.2.4 IV

```
octad crypto::IV
```

IV as octad

#### 4.2.2.5 record

```
unsigned crypto::record
```

current record number - to be incremented

#### 4.2.2.6 suite

```
int crypto::suite
```

Cipher Suite

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

- [tls1\\_3.h](#)

### 4.3 ee\_expt Struct Reference

server encrypted extensions expectations

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

#### Data Fields

- bool [early\\_data](#)
- bool [alpn](#)
- bool [server\\_name](#)
- bool [max\\_frag\\_length](#)

#### 4.3.1 Detailed Description

server encrypted extensions expectations

#### 4.3.2 Field Documentation

##### 4.3.2.1 early\_data

```
bool ee_expt::early_data
```

true if early data acceptance expected

#### 4.3.2.2 alpn

```
bool ee_expt::alpn
```

true if ALPN response expected

#### 4.3.2.3 server\_name

```
bool ee_expt::server_name
```

true if server name extension response expected

#### 4.3.2.4 max\_frag\_length

```
bool ee_expt::max_frag_length
```

true if max frag length request made

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

- [tls1\\_3.h](#)

## 4.4 ee\_resp Struct Reference

server encrypted extensions responses

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

### Data Fields

- bool [early\\_data](#)
- bool [alpn](#)
- bool [server\\_name](#)
- bool [max\\_frag\\_length](#)

#### 4.4.1 Detailed Description

server encrypted extensions responses

#### 4.4.2 Field Documentation

#### 4.4.2.1 early\_data

```
bool ee_resp::early_data
```

true if early data accepted

#### 4.4.2.2 alpn

```
bool ee_resp::alpn
```

true if ALPN accepted

#### 4.4.2.3 server\_name

```
bool ee_resp::server_name
```

true if server name accepted

#### 4.4.2.4 max\_frag\_length

```
bool ee_resp::max_frag_length
```

true if max frag length respected

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

- [tls1\\_3.h](#)

## 4.5 octad Struct Reference

Safe representation of an octad.

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

### Data Fields

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

#### 4.5.1 Detailed Description

Safe representation of an octad.

## 4.5.2 Field Documentation

### 4.5.2.1 len

```
int octad::len
```

length in bytes

### 4.5.2.2 max

```
int octad::max
```

max length allowed - enforce truncation

### 4.5.2.3 val

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

byte array

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

- [tls\\_octads.h](#)

## 4.6 pktype Struct Reference

Public key type.

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

### Data Fields

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

### 4.6.1 Detailed Description

Public key type.

## 4.6.2 Field Documentation

### 4.6.2.1 type

`int pktype::type`

signature type (ECC or RSA)

### 4.6.2.2 hash

`int pktype::hash`

hash type

### 4.6.2.3 curve

`int pktype::curve`

elliptic curve used or RSA key length in bits

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

- [tls\\_x509.h](#)

## 4.7 ret Struct Reference

function return structure

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

### Data Fields

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

### 4.7.1 Detailed Description

function return structure

### 4.7.2 Field Documentation

#### 4.7.2.1 val

```
unsigned32 ret::val
```

return value

#### 4.7.2.2 err

```
int ret::err
```

error return

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

- [tls1\\_3.h](#)

## 4.8 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** ()

#### 4.8.1 Detailed Description

[Socket](#) instance.

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

- [tls\\_sockets.h](#)

## 4.9 ticket Struct Reference

ticket context structure

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

### Data Fields

- char [tick](#) [TLS\_MAX\_TICKET\_SIZE]
- char [nonce](#) [TLS\_MAX\_KEY]
- octad [TICK](#)
- octad [NONCE](#)
- int [lifetime](#)
- [unsign32](#) [age\\_obfuscator](#)
- [unsign32](#) [max\\_early\\_data](#)
- [unsign32](#) [birth](#)
- int [cipher\\_suite](#)
- int [favourite\\_group](#)

### 4.9.1 Detailed Description

ticket context structure

### 4.9.2 Field Documentation

#### 4.9.2.1 tick

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

Ticket bytes

#### 4.9.2.2 nonce

```
char ticket::nonce[TLS_MAX_KEY]
```

32-byte nonce

#### 4.9.2.3 TICK

```
octad ticket::TICK
```

Ticket or external PSK as octad



#### 4.9.2.4 NONCE

```
octad ticket::NONCE
```

Nonce or external PSK label as octad

#### 4.9.2.5 lifetime

```
int ticket::lifetime
```

ticket lifetime

#### 4.9.2.6 age\_obfuscator

```
unsign32 ticket::age_obfuscator
```

ticket age obfuscator - 0 for external PSK

#### 4.9.2.7 max\_early\_data

```
unsign32 ticket::max_early_data
```

Maximum early data allowed for this ticket

#### 4.9.2.8 birth

```
unsign32 ticket::birth
```

Birth time of this ticket

#### 4.9.2.9 cipher\_suite

```
int ticket::cipher_suite
```

Cipher suite used

#### 4.9.2.10 favourite\_group

```
int ticket::favourite_group
```

the server's favourite group

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

- [tls1\\_3.h](#)

## 4.10 unihash Struct Reference

Universal Hash structure.

```
#include <tls_sal.h>
```

### Data Fields

- char [state](#) [[TLS\\_MAX\\_HASH\\_STATE](#)]
- int [htype](#)

### 4.10.1 Detailed Description

Universal Hash structure.

### 4.10.2 Field Documentation

#### 4.10.2.1 state

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

hash function state

#### 4.10.2.2 htype

```
int unihash::htype
```

The hash type (typically SHA256)

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

- [tls\\_sal.h](#)

## Chapter 5

# File Documentation

### 5.1 tls1\_3.h File Reference

Main TLS 1.3 Header File for constants and structures.

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

#### Data Structures

- struct [ret](#)  
*function return structure*
- struct [ee\\_resp](#)  
*server encrypted extensions responses*
- struct [ee\\_expt](#)  
*server encrypted extensions expectations*
- struct [crypto](#)  
*crypto context structure*
- struct [ticket](#)  
*ticket context structure*
- struct [capabilities](#)  
*Cryptographic capabilities of the client.*

#### Macros

- #define [IO\\_NONE](#) 0
- #define [IO\\_APPLICATION](#) 1
- #define [IO\\_PROTOCOL](#) 2
- #define [IO\\_DEBUG](#) 3
- #define [IO\\_WIRE](#) 4
- #define [TLS\\_HTTP\\_PROTOCOL](#) 1
- #define [VERBOSITY](#) [IO\\_PROTOCOL](#)
- #define [THIS\\_YEAR](#) 2021
- #define [HAVE\\_A\\_CLIENT\\_CERT](#)

- `#define TLS_PROTOCOL TLS_HTTP_PROTOCOL`
- `#define TLS_AES_128 16`
- `#define TLS_AES_256 32`
- `#define TLS_CHA_256 32`
- `#define TLS_MAX_HASH_STATE 1024`
- `#define TLS_MAX_HASH 64`
- `#define TLS_MAX_KEY 32`
- `#define TLS_X509_MAX_FIELD 256`
- `#define TLS_MAX_ROOT_CERT_SIZE 2048`
- `#define TLS_MAX_ROOT_CERT_B64 2800`
- `#define TLS_MAX_MYCERT_SIZE 2048`
- `#define TLS_MAX_MYCERT_B64 2800`
- `#define TLS_MAX_CLIENT_HELLO 256`
- `#define TLS_MAX_EXT_LABEL 256`
- `#define TLS_MAX_TICKET_SIZE 2048`
- `#define TLS_MAX_EXTENSIONS 2048`
- `#define TLS_MAX_IO_SIZE 8192`
- `#define TLS_MAX_SIGNATURE_SIZE 512`
- `#define TLS_MAX_PUB_KEY_SIZE 512`
- `#define TLS_MAX_SECRET_KEY_SIZE 512`
- `#define TLS_MAX_ECC_FIELD 66`
- `#define TLS_IV_SIZE 12`
- `#define TLS_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 X25519 0x001d`
- `#define SECP256R1 0x0017`
- `#define SECP384R1 0x0018`
- `#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 PSKOK 0x00`
- `#define PSKWECDHE 0x01`
- `#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 [PRESHARED\\_KEY](#) 0x0029
- #define [TLS\\_VER](#) 0x002b
- #define [COOKIE](#) 0x002c
- #define [EARLY\\_DATA](#) 0x002a
- #define [MAX\\_FRAG\\_LENGTH](#) 0x0001
- #define [PADDING](#) 0x0015
- #define [APP\\_PROTOCOL](#) 0x0010
- #define [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 [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 [CLOSE\\_NOTIFY](#) 0x00

## Typedefs

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

### 5.1.1 Detailed Description

Main TLS 1.3 Header File for constants and structures.

Author

Mike Scott

### 5.1.2 Macro Definition Documentation

#### 5.1.2.1 IO\_NONE

```
#define IO_NONE 0
```

Run silently

#### 5.1.2.2 IO\_APPLICATION

```
#define IO_APPLICATION 1
```

just print application traffic

#### 5.1.2.3 IO\_PROTOCOL

```
#define IO_PROTOCOL 2
```

print protocol progress + application traffic

#### 5.1.2.4 IO\_DEBUG

```
#define IO_DEBUG 3
```

print lots of debug information + protocol progress + application progress

#### 5.1.2.5 IO\_WIRE

```
#define IO_WIRE 4
```

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

#### 5.1.2.6 TLS\_HTTP\_PROTOCOL

```
#define TLS_HTTP_PROTOCOL 1
```

Supported ALPN protocol

#### 5.1.2.7 VERBOSITY

```
#define VERBOSITY IO_PROTOCOL
```

Set to level of output information desired - see above

#### 5.1.2.8 THIS\_YEAR

```
#define THIS_YEAR 2021
```

Set to this year - crudely used to deprecate old certificates

#### 5.1.2.9 HAVE\_A\_CLIENT\_CERT

```
#define HAVE_A_CLIENT_CERT
```

Indicate willingness to authenticate with a cert plus signing key

#### 5.1.2.10 TLS\_PROTOCOL

```
#define TLS_PROTOCOL TLS_HTTP_PROTOCOL
```

Selected application protocol

#### 5.1.2.11 TLS\_AES\_128

```
#define TLS_AES_128 16
```

AES128 key length in bytes

#### 5.1.2.12 TLS\_AES\_256

```
#define TLS_AES_256 32
```

AES256 key length in bytes

#### 5.1.2.13 TLS\_CHA\_256

```
#define TLS_CHA_256 32
```

IETF CHACHA20 key length in bytes

#### 5.1.2.14 TLS\_MAX\_HASH\_STATE

```
#define TLS_MAX_HASH_STATE 1024
```

Maximum memory required to store hash function state

#### 5.1.2.15 TLS\_MAX\_HASH

```
#define TLS_MAX_HASH 64
```

Maximum hash output length in bytes

#### 5.1.2.16 TLS\_MAX\_KEY

```
#define TLS_MAX_KEY 32
```

Maximum key length in bytes

#### 5.1.2.17 TLS\_X509\_MAX\_FIELD

```
#define TLS_X509_MAX_FIELD 256
```

Maximum X.509 field size

#### 5.1.2.18 TLS\_MAX\_ROOT\_CERT\_SIZE

```
#define TLS_MAX_ROOT_CERT_SIZE 2048
```

I checked - current max for root CAs is 2016

#### 5.1.2.19 TLS\_MAX\_ROOT\_CERT\_B64

```
#define TLS_MAX_ROOT_CERT_B64 2800
```

In base64 - current max for root CAs is 2688

#### 5.1.2.20 TLS\_MAX\_MYCERT\_SIZE

```
#define TLS_MAX_MYCERT_SIZE 2048
```

Max client private key/cert



**5.1.2.21 TLS\_MAX\_MYCERT\_B64**

```
#define TLS_MAX_MYCERT_B64 2800
```

In base64 - Max client private key/cert

**5.1.2.22 TLS\_MAX\_CLIENT\_HELLO**

```
#define TLS_MAX_CLIENT_HELLO 256
```

Max client hello size (less extensions)

**5.1.2.23 TLS\_MAX\_EXT\_LABEL**

```
#define TLS_MAX_EXT_LABEL 256
```

Max external psk label size

**5.1.2.24 TLS\_MAX\_TICKET\_SIZE**

```
#define TLS_MAX_TICKET_SIZE 2048
```

maximum resumption ticket size

**5.1.2.25 TLS\_MAX\_EXTENSIONS**

```
#define TLS_MAX_EXTENSIONS 2048
```

Max extensions size

**5.1.2.26 TLS\_MAX\_IO\_SIZE**

```
#define TLS_MAX_IO_SIZE 8192
```

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

**5.1.2.27 TLS\_MAX\_SIGNATURE\_SIZE**

```
#define TLS_MAX_SIGNATURE_SIZE 512
```

Max digital signature size in bytes

#### 5.1.2.28 TLS\_MAX\_PUB\_KEY\_SIZE

```
#define TLS_MAX_PUB_KEY_SIZE 512
```

Max public key size in bytes

#### 5.1.2.29 TLS\_MAX\_SECRET\_KEY\_SIZE

```
#define TLS_MAX_SECRET_KEY_SIZE 512
```

Max private key size in bytes

#### 5.1.2.30 TLS\_MAX\_ECC\_FIELD

```
#define TLS_MAX_ECC_FIELD 66
```

Max ECC field size in bytes

#### 5.1.2.31 TLS\_IV\_SIZE

```
#define TLS_IV_SIZE 12
```

Max IV size in bytes

#### 5.1.2.32 TLS\_TAG\_SIZE

```
#define TLS_TAG_SIZE 16
```

Max HMAC tag length in bytes

#### 5.1.2.33 TLS\_MAX\_COOKIE

```
#define TLS_MAX_COOKIE 128
```

Max Cookie size

#### 5.1.2.34 TLS\_MAX\_SERVER\_NAME

```
#define TLS_MAX_SERVER_NAME 128
```

Max server name size in bytes

#### 5.1.2.35 TLS\_MAX\_SUPPORTED\_GROUPS

```
#define TLS_MAX_SUPPORTED_GROUPS 5
```

Max number of supported crypto groups

#### 5.1.2.36 TLS\_MAX\_SUPPORTED\_SIGS

```
#define TLS_MAX_SUPPORTED_SIGS 16
```

Max number of supported signature schemes

#### 5.1.2.37 TLS\_MAX\_PSK\_MODES

```
#define TLS_MAX_PSK_MODES 2
```

Max preshared key modes

#### 5.1.2.38 TLS\_MAX\_CIPHER\_SUITES

```
#define TLS_MAX_CIPHER_SUITES 5
```

Max number of supported cipher suites

#### 5.1.2.39 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

#### 5.1.2.40 TLS\_AES\_256\_GCM\_SHA384

```
#define TLS_AES_256_GCM_SHA384 0x1302
```

AES256/SHA384/GCM cipher suite

#### 5.1.2.41 TLS\_CHACHA20\_POLY1305\_SHA256

```
#define TLS_CHACHA20_POLY1305_SHA256 0x1303
```

CHACHA20/SHA256/POLY1305 cipher suite

#### 5.1.2.42 X25519

```
#define X25519 0x001d
```

X25519 elliptic curve key exchange

#### 5.1.2.43 SECP256R1

```
#define SECP256R1 0x0017
```

NIST SECP256R1 elliptic curve key exchange

#### 5.1.2.44 SECP384R1

```
#define SECP384R1 0x0018
```

NIST SECP384R1 elliptic curve key exchange

#### 5.1.2.45 ECDSA\_SECP256R1\_SHA256

```
#define ECDSA_SECP256R1_SHA256 0x0403
```

Supported ECDSA Signature algorithm

#### 5.1.2.46 ECDSA\_SECP384R1\_SHA384

```
#define ECDSA_SECP384R1_SHA384 0x0503
```

Supported ECDSA Signature algorithm

#### 5.1.2.47 RSA\_PSS\_RSAE\_SHA256

```
#define RSA_PSS_RSAE_SHA256 0x0804
```

Supported RSA Signature algorithm

#### 5.1.2.48 RSA\_PSS\_RSAE\_SHA384

```
#define RSA_PSS_RSAE_SHA384 0x0805
```

Supported RSA Signature algorithm

#### 5.1.2.49 RSA\_PSS\_RSAE\_SHA512

```
#define RSA_PSS_RSAE_SHA512 0x0806
```

Supported RSA Signature algorithm

#### 5.1.2.50 RSA\_PKCS1\_SHA256

```
#define RSA_PKCS1_SHA256 0x0401
```

Supported RSA Signature algorithm

#### 5.1.2.51 RSA\_PKCS1\_SHA384

```
#define RSA_PKCS1_SHA384 0x0501
```

Supported RSA Signature algorithm

**5.1.2.52 RSA\_PKCS1\_SHA512**

```
#define RSA_PKCS1_SHA512 0x0601
```

Supported RSA Signature algorithm

**5.1.2.53 ED25519**

```
#define ED25519 0x0807
```

Ed25519 EdDSA Signature algorithm

**5.1.2.54 PSKOK**

```
#define PSKOK 0x00
```

Preshared Key only mode

**5.1.2.55 PSKWECDHE**

```
#define PSKWECDHE 0x01
```

Preshared Key with Diffie-Hellman key exchange mode

**5.1.2.56 TLS1\_0**

```
#define TLS1_0 0x0301
```

TLS 1.0 version

**5.1.2.57 TLS1\_2**

```
#define TLS1_2 0x0303
```

TLS 1.2 version

**5.1.2.58 TLS1\_3**

```
#define TLS1_3 0x0304
```

TLS 1.3 version

**5.1.2.59 SERVER\_NAME**

```
#define SERVER_NAME 0x0000
```

Server Name extension

**5.1.2.60 SUPPORTED\_GROUPS**

```
#define SUPPORTED_GROUPS 0x000a
```

Supported Group extension

**5.1.2.61 SIG\_ALGS**

```
#define SIG_ALGS 0x000d
```

Signature algorithms extension

**5.1.2.62 SIG\_ALGS\_CERT**

```
#define SIG_ALGS_CERT 0x0032
```

Signature algorithms Certificate extension

**5.1.2.63 KEY\_SHARE**

```
#define KEY_SHARE 0x0033
```

Key Share extension

**5.1.2.64 PSK\_MODE**

```
#define PSK_MODE 0x002d
```

Preshared key mode extension

**5.1.2.65 PRESHARED\_KEY**

```
#define PRESHARED_KEY 0x0029
```

Preshared key extension

**5.1.2.66 TLS\_VER**

```
#define TLS_VER 0x002b
```

TLS version extension

**5.1.2.67 COOKIE**

```
#define COOKIE 0x002c
```

Cookie extension

**5.1.2.68 EARLY\_DATA**

```
#define EARLY_DATA 0x002a
```

Early Data extension

**5.1.2.69 MAX\_FRAG\_LENGTH**

```
#define MAX_FRAG_LENGTH 0x0001
```

max fragmentation length extension

**5.1.2.70 PADDING**

```
#define PADDING 0x0015
```

Padding extension

**5.1.2.71 APP\_PROTOCOL**

```
#define APP_PROTOCOL 0x0010
```

Application Layer Protocol Negotiation (ALPN)

**5.1.2.72 HSHAKE**

```
#define HSHAKE 0x16
```

Handshake record

**5.1.2.73 APPLICATION**

```
#define APPLICATION 0x17
```

Application record

**5.1.2.74 ALERT**

```
#define ALERT 0x15
```

Alert record

**5.1.2.75 CHANGE\_CIPHER**

```
#define CHANGE_CIPHER 0x14
```

Change Cipher record

#### 5.1.2.76 TIMED\_OUT

```
#define TIMED_OUT 0x01
```

Time-out

#### 5.1.2.77 CLIENT\_HELLO

```
#define CLIENT_HELLO 0x01
```

Client Hello message

#### 5.1.2.78 SERVER\_HELLO

```
#define SERVER_HELLO 0x02
```

Server Hello message

#### 5.1.2.79 CERTIFICATE

```
#define CERTIFICATE 0x0b
```

Certificate message

#### 5.1.2.80 CERT\_REQUEST

```
#define CERT_REQUEST 0x0d
```

Certificate Request

#### 5.1.2.81 CERT\_VERIFY

```
#define CERT_VERIFY 0x0f
```

Certificate Verify message

#### 5.1.2.82 FINISHED

```
#define FINISHED 0x14
```

Handshae Finished message



### 5.1.2.83 ENCRYPTED\_EXTENSIONS

```
#define ENCRYPTED_EXTENSIONS 0x08
```

Encrypted Extensions message

### 5.1.2.84 TICKET

```
#define TICKET 0x04
```

Ticket message

### 5.1.2.85 KEY\_UPDATE

```
#define KEY_UPDATE 0x18
```

Key Update message

### 5.1.2.86 MESSAGE\_HASH

```
#define MESSAGE_HASH 0xFE
```

Special synthetic message hash message

### 5.1.2.87 END\_OF\_EARLY\_DATA

```
#define END_OF_EARLY_DATA 0x05
```

End of Early Data message

### 5.1.2.88 HANDSHAKE\_RETRY

```
#define HANDSHAKE_RETRY 0x102
```

Handshake retry

### 5.1.2.89 NOT\_TLS1\_3

```
#define NOT_TLS1_3 -2
```

Wrong version error, not TLS1.3

### 5.1.2.90 BAD\_CERT\_CHAIN

```
#define BAD_CERT_CHAIN -3
```

Bad Certificate Chain error

#### 5.1.2.91 ID\_MISMATCH

```
#define ID_MISMATCH -4
```

Session ID mismatch error

#### 5.1.2.92 UNRECOGNIZED\_EXT

```
#define UNRECOGNIZED_EXT -5
```

Unrecognised extension error

#### 5.1.2.93 BAD\_HELLO

```
#define BAD_HELLO -6
```

badly formed Hello message error

#### 5.1.2.94 WRONG\_MESSAGE

```
#define WRONG_MESSAGE -7
```

Message out-of-order error

#### 5.1.2.95 MISSING\_REQUEST\_CONTEXT

```
#define MISSING_REQUEST_CONTEXT -8
```

Request context missing error

#### 5.1.2.96 AUTHENTICATION\_FAILURE

```
#define AUTHENTICATION_FAILURE -9
```

Authentication error - AEAD Tag incorrect

#### 5.1.2.97 BAD\_RECORD

```
#define BAD_RECORD -10
```

Badly formed Record received

#### 5.1.2.98 BAD\_TICKET

```
#define BAD_TICKET -11
```

Badly formed Ticket received

**5.1.2.99 NOT\_EXPECTED**

```
#define NOT_EXPECTED -12
```

Received ack for something not requested

**5.1.2.100 CA\_NOT\_FOUND**

```
#define CA_NOT_FOUND -13
```

Certificate Authority not found

**5.1.2.101 CERT\_OUTOFDATE**

```
#define CERT_OUTOFDATE -14
```

Certificate Expired

**5.1.2.102 MEM\_OVERFLOW**

```
#define MEM_OVERFLOW -15
```

Memory Overflow

**5.1.2.103 ILLEGAL\_PARAMETER**

```
#define ILLEGAL_PARAMETER 0x2F
```

Illegal parameter alert

**5.1.2.104 UNEXPECTED\_MESSAGE**

```
#define UNEXPECTED_MESSAGE 0x0A
```

Unexpected message alert

**5.1.2.105 DECRYPT\_ERROR**

```
#define DECRYPT_ERROR 0x33
```

Decryption error alert

**5.1.2.106 BAD\_CERTIFICATE**

```
#define BAD_CERTIFICATE 0x2A
```

Bad certificate alert

#### 5.1.2.107 UNSUPPORTED\_EXTENSION

```
#define UNSUPPORTED_EXTENSION 0x6E
```

Unsupported extension alert

#### 5.1.2.108 UNKNOWN\_CA

```
#define UNKNOWN_CA 0x30
```

Unrecognised Certificate Authority

#### 5.1.2.109 CERTIFICATE\_EXPIRED

```
#define CERTIFICATE_EXPIRED 0x2D
```

Certificate Expired

#### 5.1.2.110 PROTOCOL\_VERSION

```
#define PROTOCOL_VERSION 0x46
```

Wrong TLS version

#### 5.1.2.111 DECODE\_ERROR

```
#define DECODE_ERROR 0x32
```

Decode error alert

#### 5.1.2.112 CLOSE\_NOTIFY

```
#define CLOSE_NOTIFY 0x00
```

Orderly shut down of connection

### 5.1.3 Typedef Documentation

#### 5.1.3.1 byte

```
using byte = uint8_t
```

8-bit unsigned integer

### 5.1.3.2 sign8

```
using sign8 = int8_t
```

8-bit signed integer

### 5.1.3.3 sign16

```
using sign16 = int16_t
```

16-bit signed integer

### 5.1.3.4 sign32

```
using sign32 = int32_t
```

32-bit signed integer

### 5.1.3.5 sign64

```
using sign64 = int64_t
```

64-bit signed integer

### 5.1.3.6 unsign32

```
using unsign32 = uint32_t
```

32-bit unsigned integer

### 5.1.3.7 unsign64

```
using unsign64 = uint64_t
```

64-bit unsigned integer

## 5.2 tls\_cacerts.h File Reference

Certificate Authority root certificate store.

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

## Variables

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

### 5.2.1 Detailed Description

Certificate Authority root certificate store.

#### Author

Mike Scott

### 5.2.2 Variable Documentation

#### 5.2.2.1 myprivate

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

Client private key

#### 5.2.2.2 mycert

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

Client certificate

#### 5.2.2.3 cacerts

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

The Root Certificate store

## 5.3 `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_cacerts.h"
```

## Macros

- `#define TLS_SHA256 32`
- `#define TLS_SHA384 48`
- `#define TLS_SHA512 64`

## Functions

- `int checkCertChain (octad *CERTCHAIN, char *hostname, octad *PUBKEY)`  
*Check Certificate Chain.*
- `bool checkServerCertVerifier (int sigalg, octad *SCVSIG, octad *H, octad *CERTPK)`  
*verify Server's signature on protocol transcript*
- `int getClientKeyAndCertchain (int nccsalgs, int *csigAlgs, octad *PRIVKEY, octad *CERTCHAIN)`  
*Get Client private key and Certificate chain from .pem files.*
- `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.*

### 5.3.1 Detailed Description

Process Certificate Chain.

Author

Mike Scott

### 5.3.2 Macro Definition Documentation

#### 5.3.2.1 TLS\_SHA256

```
#define TLS_SHA256 32
```

SHA256 hash length in bytes

#### 5.3.2.2 TLS\_SHA384

```
#define TLS_SHA384 48
```

SHA384 hash length in bytes

#### 5.3.2.3 TLS\_SHA512

```
#define TLS_SHA512 64
```

SHA512 hash length in bytes

### 5.3.3 Function Documentation

#### 5.3.3.1 checkCertChain()

```
int checkCertChain (
    octad * CERTCHAIN,
    char * hostname,
    octad * PUBKEY )
```

Check Certificate Chain.

##### 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

##### Returns

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

#### 5.3.3.2 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



### 5.3.3.3 `getClientKeyAndCertchain()`

```
int getClientKeyAndCertchain (
    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

### 5.3.3.4 `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

## 5.4 `tls_client_rcv.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 parseInt16** (octad \*M, int &ptr)  
*Parse out a 16-bit unsigned integer from a pointer into an octad.*
- **ret parseInt24** (octad \*M, int &ptr)  
*Parse out a 24-bit unsigned integer from a pointer into an octad.*
- **ret parseInt32** (octad \*M, int &ptr)  
*Parse out a 32-bit unsigned integer from a pointer into an octad.*
- **ret parseByte** (octad \*M, int &ptr)  
*Parse out an unsigned byte 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** (Socket &client, crypto \*recv, octad \*IO)  
*Read a record from the Server, a fragment of a full protocol message.*
- **ret parseByteorPull** (Socket &client, octad \*IO, int &ptr, crypto \*recv)  
*Parse out an unsigned byte from a pointer into an octad, if necessary pulling in a new fragment.*
- **ret parseInt32orPull** (Socket &client, octad \*IO, int &ptr, crypto \*recv)  
*Parse out a 32-bit unsigned integer from a pointer into an octad, if necessary pulling in a new fragment.*
- **ret parseInt24orPull** (Socket &client, octad \*IO, int &ptr, crypto \*recv)  
*Parse out a 24-bit unsigned integer from a pointer into an octad, if necessary pulling in a new fragment.*
- **ret parseInt16orPull** (Socket &client, octad \*IO, int &ptr, crypto \*recv)  
*Parse out a 16-bit unsigned integer from a pointer into an octad, if necessary pulling in a new fragment.*
- **ret parseoctadorPull** (Socket &client, octad \*O, int len, octad \*IO, int &ptr, crypto \*recv)  
*Parse out an octad from a pointer into an octad, if necessary pulling in a new fragment.*
- **ret parseoctadorPullptr** (Socket &client, octad \*O, int len, octad \*IO, int &ptr, crypto \*recv)  
*Return a pointer to an octad from a pointer into an octad, if necessary pulling in a new fragment.*
- **bool badResponse** (Socket &client, crypto \*send, ret r)  
*Process response from server input.*
- **ret getWhatsNext** (Socket &client, octad \*IO, crypto \*recv, unihash \*trans\_hash)  
*Identify type of message.*
- **ret getServerEncryptedExtensions** (Socket &client, octad \*IO, crypto \*recv, unihash \*trans\_hash, ee\_expt \*enc\_ext\_expt, ee\_resp \*enc\_ext\_resp)  
*Receive and parse Server Encrypted Extensions.*
- **ret getServerCertVerify** (Socket &client, octad \*IO, crypto \*recv, unihash \*trans\_hash, octad \*SCVSIG, int &sigalg)  
*Get Server proof that he owns the Certificate, by receiving and verifying its signature on transcript hash.*
- **ret getServerFinished** (Socket &client, octad \*IO, crypto \*recv, unihash \*trans\_hash, octad \*HFIN)  
*Get final handshake message from Server, a HMAC on the transcript hash.*
- **ret getServerHello** (Socket &client, octad \*SH, int &cipher, int &kex, octad \*CID, octad \*CK, octad \*PK, int &pskid)  
*Receive and parse initial Server Hello.*
- **ret getCheckServerCertificateChain** (Socket &client, octad \*IO, crypto \*recv, unihash \*trans\_hash, char \*hostname, octad \*PUBKEY)  
*Receive and check certificate chain.*
- **ret getCertificateRequest** (Socket &client, octad \*IO, crypto \*recv, unihash \*trans\_hash, int &nalgs, int \*sigalgs)  
*process a Certificate Request*

### 5.4.1 Detailed Description

Process Input received from the Server.

Author

Mike Scott

### 5.4.2 Function Documentation

#### 5.4.2.1 `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

#### 5.4.2.2 `parseInt16()`

```
ret parseInt16 (
    octad * M,
    int & ptr )
```

Parse out a 16-bit unsigned integer from a pointer into an octad.

Parameters

<i>M</i>	the input octad
<i>ptr</i>	a pointer into M, which advances after use

**Returns**

the 16-bit integer value, and an error flag

**5.4.2.3 parseInt24()**

```
ret parseInt24 (
    octad * M,
    int & ptr )
```

Parse out a 24-bit unsigned integer from a pointer into an octad.

**Parameters**

<i>M</i>	the input octad
<i>ptr</i>	a pointer into M, which advances after use

**Returns**

the 24-bit integer value, and an error flag

**5.4.2.4 parseInt32()**

```
ret parseInt32 (
    octad * M,
    int & ptr )
```

Parse out a 32-bit unsigned integer from a pointer into an octad.

**Parameters**

<i>M</i>	the input octad
<i>ptr</i>	a pointer into M, which advances after use

**Returns**

the 32-bit integer value, and an error flag

**5.4.2.5 parseByte()**

```
ret parseByte (
    octad * M,
    int & ptr )
```

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

## Parameters

<i>M</i>	the input octad
<i>ptr</i>	a pointer into <i>M</i> , which advances after use

## Returns

the unsigned byte, and an error flag

**5.4.2.6 `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 <i>M</i>
<i>len</i>	the expected length of the octad <i>E</i>
<i>M</i>	the input octad
<i>ptr</i>	a pointer into <i>M</i> , which advances after use

## Returns

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

**5.4.2.7 `getServerFragment()`**

```
int getServerFragment (
    Socket & client,
    crypto * recv,
    octad * IO )
```

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

## Parameters

<i>client</i>	the socket connection to the Server
<i>recv</i>	the cryptographic key under which the fragment is encrypted, or NULL if not encrypted
<i>IO</i>	the received record, a protocol message fragment

**Returns**

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

**5.4.2.8 parseByteorPull()**

```
ret parseByteorPull (
    Socket & client,
    octad * IO,
    int & ptr,
    crypto * recv )
```

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

**Parameters**

<i>client</i>	the socket connection to the Server
<i>IO</i>	the input octad
<i>ptr</i>	a pointer into IO, which advances after use
<i>recv</i>	the cryptographic key under which the fragment is encrypted, or NULL if not encrypted

**Returns**

the unsigned byte, and an error flag

**5.4.2.9 parseInt32orPull()**

```
ret parseInt32orPull (
    Socket & client,
    octad * IO,
    int & ptr,
    crypto * recv )
```

Parse out a 32-bit unsigned integer from a pointer into an octad, if necessary pulling in a new fragment.

**Parameters**

<i>client</i>	the socket connection to the Server
<i>IO</i>	the input octad
<i>ptr</i>	a pointer into IO, which advances after use
<i>recv</i>	the cryptographic key under which the fragment is encrypted, or NULL if not encrypted

**Returns**

the 32-bit integer value, and an error flag

#### 5.4.2.10 `parseInt24orPull()`

```
ret parseInt24orPull (
    Socket & client,
    octad * IO,
    int & ptr,
    crypto * rcv )
```

Parse out a 24-bit unsigned integer from a pointer into an octad, if necessary pulling in a new fragment.

##### Parameters

<i>client</i>	the socket connection to the Server
<i>IO</i>	the input octad
<i>ptr</i>	a pointer into IO, which advances after use
<i>rcv</i>	the cryptographic key under which the fragment is encrypted, or NULL if not encrypted

##### Returns

the 24-bit integer value, and an error flag

#### 5.4.2.11 `parseInt16orPull()`

```
ret parseInt16orPull (
    Socket & client,
    octad * IO,
    int & ptr,
    crypto * rcv )
```

Parse out a 16-bit unsigned integer from a pointer into an octad, if necessary pulling in a new fragment.

##### Parameters

<i>client</i>	the socket connection to the Server
<i>IO</i>	the input octad
<i>ptr</i>	a pointer into IO, which advances after use
<i>rcv</i>	the cryptographic key under which the fragment is encrypted, or NULL if not encrypted

##### Returns

the 16-bit integer value, and an error flag

#### 5.4.2.12 `parseoctadorPull()`

```
ret parseoctadorPull (
    Socket & client,
```

```

    octad * O,
    int len,
    octad * IO,
    int & ptr,
    crypto * recv )

```

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

#### Parameters

<i>client</i>	the socket connection to the Server
<i>O</i>	the output octad
<i>len</i>	the expected length of the output octad O
<i>IO</i>	the input octad
<i>ptr</i>	a pointer into IO, which advances after use
<i>recv</i>	the cryptographic key under which the fragment is encrypted, or NULL if not encrypted

#### Returns

the actual length of O extracted, and an error flag

#### 5.4.2.13 parseoctadorPullptr()

```

ret parseoctadorPullptr (
    Socket & client,
    octad * O,
    int len,
    octad * IO,
    int & ptr,
    crypto * recv )

```

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

#### Parameters

<i>client</i>	the socket connection to the Server
<i>O</i>	a pointer to an octad contained within an octad IO
<i>len</i>	the expected length of the octad O
<i>IO</i>	the input octad
<i>ptr</i>	a pointer into IO, which advances after use
<i>recv</i>	the cryptographic key under which the fragment is encrypted, or NULL if not encrypted

#### Returns

the actual length of O extracted, and an error flag



#### 5.4.2.14 badResponse()

```
bool badResponse (
    Socket & client,
    crypto * send,
    ret r )
```

Process response from server input.

##### Parameters

<i>client</i>	the socket connection to the Server
<i>send</i>	the cryptographic key under which an outgoing alert may be encrypted
<i>r</i>	return value to be processed

##### Returns

true, if its a bad response requiring an abort

#### 5.4.2.15 getWhatsNext()

```
ret getWhatsNext (
    Socket & client,
    octad * IO,
    crypto * recv,
    unihash * trans_hash )
```

Identify type of message.

##### Parameters

<i>client</i>	the socket connection to the Server
<i>IO</i>	an octad to accept input
<i>recv</i>	the cryptographic key under which communications are encrypted
<i>trans_hash</i>	the current and updated transcript hash

##### Returns

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

#### 5.4.2.16 getServerEncryptedExtensions()

```
ret getServerEncryptedExtensions (
    Socket & client,
    octad * IO,
```

```

    crypto * recv,
    unihash * trans_hash,
    ee_expt * enc_ext_expt,
    ee_resp * enc_ext_resp )

```

Receive and parse Server Encrypted Extensions.

#### Parameters

<i>client</i>	the socket connection to the Server
<i>IO</i>	an octad to accept input
<i>recv</i>	the cryptographic key under which the extensions are encrypted
<i>trans_hash</i>	the current and updated transcript hash
<i>enc_ext_expt</i>	ext structure containing server expectations
<i>enc_ext_resp</i>	ext structure containing server responses

#### Returns

response structure

#### 5.4.2.17 getServerCertVerify()

```

ret getServerCertVerify (
    Socket & client,
    octad * IO,
    crypto * recv,
    unihash * trans_hash,
    octad * SCVSiG,
    int & sigalg )

```

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

#### Parameters

<i>client</i>	the socket connection to the Server
<i>IO</i>	an octad to accept server input
<i>recv</i>	the cryptographic key under which the server response is encrypted
<i>trans_hash</i>	the current and updated transcript hash
<i>SCVSiG</i>	the received signature on the transcript hash
<i>sigalg</i>	the type of the received signature

#### Returns

response structure

#### 5.4.2.18 getServerFinished()

```
ret getServerFinished (
    Socket & client,
    octad * IO,
    crypto * recv,
    unihash * trans_hash,
    octad * HFIN )
```

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

##### Parameters

<i>client</i>	the socket connection to the Server
<i>IO</i>	an octad to accept input
<i>recv</i>	the cryptographic key under which the server response is encrypted
<i>trans_hash</i>	the current and updated transcript hash
<i>HFIN</i>	an octad containing HMAC on transcript as calculated by Server

##### Returns

response structure

#### 5.4.2.19 getServerHello()

```
ret getServerHello (
    Socket & client,
    octad * SH,
    int & cipher,
    int & kex,
    octad * CID,
    octad * CK,
    octad * PK,
    int & pskid )
```

Receive and parse initial Server Hello.

##### Parameters

<i>client</i>	the socket connection to the Server
<i>SH</i>	an octad to accept server input
<i>cipher</i>	the agreed cipher suite
<i>kex</i>	key exchange data
<i>CID</i>	random session identity
<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

**5.4.2.20 getCheckServerCertificateChain()**

```
ret getCheckServerCertificateChain (
    Socket & client,
    octad * IO,
    crypto * recv,
    unihash * trans_hash,
    char * hostname,
    octad * PUBKEY )
```

Receive and check certificate chain.

**Parameters**

<i>client</i>	the socket connection to the Server
<i>IO</i>	an octad to accept server supplied certificate chain
<i>recv</i>	the cryptographic key under which the server response is encrypted
<i>trans_hash</i>	the current and updated transcript hash
<i>hostname</i>	the Server name which the client wants confirmed by Server Certificate
<i>PUBKEY</i>	the public key extracted from the Server certificate

**Returns**

response structure

**5.4.2.21 getCertificateRequest()**

```
ret getCertificateRequest (
    Socket & client,
    octad * IO,
    crypto * recv,
    unihash * trans_hash,
    int & nalgs,
    int * sigalgs )
```

process a Certificate Request

**Parameters**

<i>client</i>	the socket connection to the Server
<i>IO</i>	an octad to accept server supplied certificate request
<i>recv</i>	the cryptographic key under which the server response is encrypted
<i>trans_hash</i>	the current and updated transcript hash
<i>nalgs</i>	the number of acceptable signature algorithms
<i>sigalgs</i>	an array of nalgs signature algorithms

## Returns

response structure

## 5.5 `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` (`Socket` &`client`)  
*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 `nsa`, 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 `nsa`, 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 `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 `sessionID` (`octad` \*`SI`)  
*Create 32-byte random session ID octad.*
- int `cipherSuites` (`octad` \*`CS`, int `ncs`, int \*`ciphers`)  
*Build a cipher-suites octad from supported ciphers.*

- void `sendClientMessage` (`Socket` &client, int rectype, int version, `crypto` \*send, `octad` \*CM, `octad` \*EXT, `octad` \*IO)  
*Send a generic client message (as a single record) to the Server.*
- void `sendBinder` (`Socket` &client, `octad` \*B, `octad` \*BND, `octad` \*IO)  
*Send a preshared key binder message to the Server.*
- void `sendClientHello` (`Socket` &client, int version, `octad` \*CH, int nsc, int \*ciphers, `octad` \*CID, `octad` \*EXTENSIONS, int extra, `octad` \*IO)  
*Prepare and send Client Hello message to the Server, appending prepared extensions.*
- void `sendClientAlert` (`Socket` &client, int type, `crypto` \*send)  
*Prepare and send an Alert message to the Server.*
- void `sendClientFinish` (`Socket` &client, `crypto` \*send, `unihash` \*h, `octad` \*CHF, `octad` \*IO)  
*Prepare and send a final handshake Verification message to the Server.*
- void `sendClientCertificateChain` (`Socket` &client, `crypto` \*send, `unihash` \*h, `octad` \*CERTCHAIN, `octad` \*IO)  
*Prepare and send client certificate message to the Server.*
- void `sendClientCertVerify` (`Socket` &client, `crypto` \*send, `unihash` \*h, int sigAlg, `octad` \*CCVSIG, `octad` \*IO)  
*Send client Certificate Verify message to the Server.*
- void `sendEndOfEarlyData` (`Socket` &client, `crypto` \*send, `unihash` \*h, `octad` \*IO)  
*Indicate End of Early Data in message to the Server.*
- int `alert_from_cause` (int rtn)  
*Maps problem cause to Alert.*

### 5.5.1 Detailed Description

Process Output to be sent to the Server.

Author

Mike Scott

### 5.5.2 Function Documentation

#### 5.5.2.1 sendCCCS()

```
void sendCCCS (
    Socket & client )
```

Send Change Cipher Suite message.

Parameters

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

### 5.5.2.2 addPreSharedKeyExt()

```
int addPreSharedKeyExt (
    octad * EXT,
    unsigned 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

### 5.5.2.3 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

### 5.5.2.4 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

### 5.5.2.5 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

### 5.5.2.6 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

### 5.5.2.7 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



### 5.5.2.8 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

### 5.5.2.9 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

### 5.5.2.10 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

### 5.5.2.11 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

### 5.5.2.12 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

### 5.5.2.13 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

### 5.5.2.14 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
------------	---

**5.5.2.15 `clientRandom()`**

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

Generate 32-byte random octad.

**Parameters**

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

**Returns**

length of output octad

**5.5.2.16 `sessionID()`**

```
int sessionID (
    octad * SI )
```

Create 32-byte random session ID octad.

**Parameters**

<i>SI</i>	the output random octad
-----------	-------------------------

**Returns**

length of output octad

**5.5.2.17 `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

**5.5.2.18 sendClientMessage()**

```
void sendClientMessage (
    Socket & client,
    int rectype,
    int version,
    crypto * send,
    octad * CM,
    octad * EXT,
    octad * IO )
```

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

## Parameters

<i>client</i>	the socket connection to the Server
<i>rectype</i>	the record type
<i>version</i>	TLS version indication
<i>send</i>	the cryptographic key under which the message is encrypted (or NULL if no encryption)
<i>CM</i>	the client message to be sent
<i>EXT</i>	extensions to be added (or NULL if there are none)
<i>IO</i>	the workspace octad in which to construct the encrypted message

**5.5.2.19 sendBinder()**

```
void sendBinder (
    Socket & client,
    octad * B,
    octad * BND,
    octad * IO )
```

Send a preshared key binder message to the Server.

## Parameters

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

## Parameters

<i>B</i>	workspace octad in which to construct binder message
<i>BND</i>	binding HMAC of truncated transcript hash
<i>IO</i>	the workspace octad in which to construct the overall message

## 5.5.2.20 sendClientHello()

```
void sendClientHello (
    Socket & client,
    int version,
    octad * CH,
    int nsc,
    int * ciphers,
    octad * CID,
    octad * EXTENSIONS,
    int extra,
    octad * IO )
```

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

## Parameters

<i>client</i>	the socket connection to the Server
<i>version</i>	TLS version indication
<i>CH</i>	workspace octad in which to build client Hello
<i>nsc</i>	the number of supported cipher-suites
<i>ciphers</i>	an array of supported cipher-suites
<i>CID</i>	random session ID (generated and used internally, and output here)
<i>EXTENSIONS</i>	pre-prepared extensions
<i>extra</i>	length of preshared key binder to be sent later
<i>IO</i>	the workspace octad in which to construct the overall message

## 5.5.2.21 sendClientAlert()

```
void sendClientAlert (
    Socket & client,
    int type,
    crypto * send )
```

Prepare and send an Alert message to the Server.

## Parameters

<i>client</i>	the socket connection to the Server
<i>type</i>	the type of the Alert
<i>send</i>	the cryptographic key under which the alert message is encrypted (or NULL if no encryption)

### 5.5.2.22 sendClientFinish()

```
void sendClientFinish (
    Socket & client,
    crypto * send,
    unihash * h,
    octad * CHF,
    octad * IO )
```

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

#### Parameters

<i>client</i>	the socket connection to the Server
<i>send</i>	the cryptographic key under which the verification message is encrypted
<i>h</i>	the current transcript hash up to this point
<i>CHF</i>	the client verify data HMAC
<i>IO</i>	the workspace octad in which to construct the overall message

### 5.5.2.23 sendClientCertificateChain()

```
void sendClientCertificateChain (
    Socket & client,
    crypto * send,
    unihash * h,
    octad * CERTCHAIN,
    octad * IO )
```

Prepare and send client certificate message to the Server.

#### Parameters

<i>client</i>	the socket connection to the Server
<i>send</i>	the cryptographic key under which the certificate message is encrypted
<i>h</i>	the current transcript hash up to this point
<i>CERTCHAIN</i>	the client certificate chain
<i>IO</i>	the workspace octad in which to construct the overall message

### 5.5.2.24 sendClientCertVerify()

```
void sendClientCertVerify (
    Socket & client,
```

```
crypto * send,  
unihash * h,  
int sigAlg,  
octad * CCVSIG,  
octad * IO )
```

Send client Certificate Verify message to the Server.

#### Parameters

<i>client</i>	the socket connection to the Server
<i>send</i>	the cryptographic key under which the certificate message is encrypted
<i>h</i>	the current transcript hash up to this point
<i>sigAlg</i>	the client's digital signature algorithm
<i>CCVSIG</i>	the client's signature
<i>IO</i>	the workspace octad in which to construct the overall message

#### 5.5.2.25 sendEndOfEarlyData()

```
void sendEndOfEarlyData (  
    Socket & client,  
    crypto * send,  
    unihash * h,  
    octad * IO )
```

Indicate End of Early Data in message to the Server.

#### Parameters

<i>client</i>	the socket connection to the Server
<i>send</i>	the cryptographic key under which the message is encrypted
<i>h</i>	the current transcript hash up to this point
<i>IO</i>	the workspace octad in which to construct the overall message

#### 5.5.2.26 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

## 5.6 tls\_keys\_calc.h File Reference

TLS 1.3 crypto support functions.

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

### Functions

- void [runningHash](#) (octad \*O, unihash \*h)  
*Accumulate octad into ongoing hashing.*
- void [transcriptHash](#) (unihash \*h, octad \*O)  
*Output current hash value.*
- void [runningSyntheticHash](#) (octad \*O, octad \*E, unihash \*h)  
*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\_suite, octad \*TS, crypto \*context)  
*Build a crypto context from an input raw Secret and an agreed cipher\_suite.*
- void [recoverPSK](#) (int htype, octad \*RMS, octad \*NONCE, octad \*PSK)  
*Recover a pre-shared key from Resumption Master Secret and a nonce.*
- 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](#) (int htype, octad \*SS, octad \*ES, octad \*H, octad \*HS, octad \*CHTS, octad \*SHTS)  
*Extract Handshake Secret from Shared Secret and Early Secret. Use Handshake Secret to extract Client and Server Handshake Traffic secrets.*
- void [deriveApplicationSecrets](#) (int htype, octad \*HS, octad \*SFH, octad \*CFH, octad \*CTS, octad \*STS, octad \*EMS, octad \*RMS)  
*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.*



## 5.6.1 Detailed Description

TLS 1.3 crypto support functions.

Author

Mike Scott

## 5.6.2 Function Documentation

### 5.6.2.1 runningHash()

```
void runningHash (
    octad * O,
    unihash * h )
```

Accumulate octad into ongoing hashing.

Parameters

<i>O</i>	an octad to be included in hash
<i>h</i>	a hashing context

### 5.6.2.2 transcriptHash()

```
void transcriptHash (
    unihash * h,
    octad * O )
```

Output current hash value.

Parameters

<i>h</i>	a hashing context
<i>O</i>	an output octad containing current hash

### 5.6.2.3 runningSyntheticHash()

```
void runningSyntheticHash (
    octad * O,
```

```

    octad * E,
    unihash * h )

```

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

#### Parameters

<i>O</i>	an octad containing clientHello
<i>E</i>	an octad containing clientHello extensions
<i>h</i>	a hashing context

#### 5.6.2.4 initCryptoContext()

```

void initCryptoContext (
    crypto * C )

```

Initiate a Crypto Context.

#### Parameters

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

#### 5.6.2.5 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

#### 5.6.2.6 incrementCryptoContext()

```

void incrementCryptoContext (
    crypto * C )

```

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

## Parameters

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

### 5.6.2.7 createCryptoContext()

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

Build a crypto context from an input raw Secret and an agreed cipher\_suite.

## Parameters

<i>cipher_suite</i>	the chosen cipher suite
<i>TS</i>	the input raw secret
<i>context</i>	an AEAD encryption context

### 5.6.2.8 recoverPSK()

```
void recoverPSK (
    int htype,
    octad * RMS,
    octad * NONCE,
    octad * PSK )
```

Recover a pre-shared key from Resumption Master Secret and a nonce.

## Parameters

<i>htype</i>	hash algorithm
<i>RMS</i>	the input resumption master secret
<i>NONCE</i>	the input nonce
<i>PSK</i>	the output pre-shared key

### 5.6.2.9 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)

#### 5.6.2.10 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)

#### 5.6.2.11 deriveHandshakeSecrets()

```

void deriveHandshakeSecrets (
    int htype,
    octad * SS,
    octad * ES,
    octad * H,
    octad * HS,
    octad * CHTS,
    octad * SHTS )

```

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

## Parameters

<i>htype</i>	hash algorithm
<i>SS</i>	input Shared Secret
<i>ES</i>	the input early secret key
<i>H</i>	a partial transcript hash
<i>HS</i>	the output Handshake Secret
<i>CHTS</i>	the output Client Handshake Traffic Secret
<i>SHTS</i>	the output Server Handshake Traffic Secret

## 5.6.2.12 deriveApplicationSecrets()

```
void deriveApplicationSecrets (
    int htype,
    octad * HS,
    octad * SFH,
    octad * CFH,
    octad * CTS,
    octad * STS,
    octad * EMS,
    octad * RMS )
```

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

## Parameters

<i>htype</i>	hash algorithm
<i>HS</i>	input Handshake Secret
<i>SFH</i>	an input partial transcript hash
<i>CFH</i>	an input partial transcript hash
<i>CTS</i>	the output Client Application Traffic Secret
<i>STS</i>	the output Server Application Traffic Secret
<i>EMS</i>	the output External Master Secret (or NULL if not required)
<i>RMS</i>	the output Resumption Master Secret (or NULL if not required)

## 5.6.2.13 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

**5.6.2.14 checkVeriferData()**

```
bool checkVeriferData (
    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

**5.6.2.15 deriveVeriferData()**

```
void deriveVeriferData (
    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

## 5.7 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 [logger](#) (char \*preamble, char \*string, [unsign32](#) info, [octad](#) \*O)  
*basic logging function*
- void [logServerHello](#) (int cipher\_suite, int kex, int pskid, [octad](#) \*PK, [octad](#) \*CK)  
*logging the Server hello*
- void [logTicket](#) (int lifetime, [unsign32](#) age\_obfuscator, [unsign32](#) max\_early\_data, [octad](#) \*NONCE, [octad](#) \*ETICK)  
*logging a resumption ticket*
- void [logEncExt](#) ([ee\\_expt](#) \*e, [ee\\_resp](#) \*r)  
*logging server extended extensions responses vs expectations*
- void [logCert](#) ([octad](#) \*CERT)  
*logging a Certificate in standard base 64 format*
- void [logCertDetails](#) (char \*txt, [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*

### 5.7.1 Detailed Description

TLS 1.3 logging.

Author

Mike Scott

### 5.7.2 Function Documentation

### 5.7.2.1 myprintf()

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

internal printf function - all output funnels through this function



## Parameters

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

## 5.7.2.2 logger()

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

basic logging function

## Parameters

<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)

## 5.7.2.3 logServerHello()

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

logging the Server hello

## Parameters

<i>cipher_suite</i>	the chosen cipher suite
<i>kex</i>	the chosen key exchange algorithm
<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)

## 5.7.2.4 logTicket()

```
void logTicket (
```

```

int lifetime,
unsigned32 age_obfuscator,
unsigned32 max_early_data,
octad * NONCE,
octad * ETICK )

```

logging a resumption ticket

#### Parameters

<i>lifetime</i>	the ticket lifetime in minutes
<i>age_obfuscator</i>	the ticket age obfuscator
<i>max_early_data</i>	the maximum amount of permitted early data
<i>NONCE</i>	the Ticket nonce
<i>ETICK</i>	the Ticket octad

#### 5.7.2.5 logEncExt()

```

void logEncExt (
    ee_expt * e,
    ee_resp * r )

```

logging server extended extensions responses vs expectations

#### Parameters

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

#### 5.7.2.6 logCert()

```

void logCert (
    octad * CERT )

```

logging a Certificate in standard base 64 format

#### Parameters

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

#### 5.7.2.7 logCertDetails()

```

void logCertDetails (

```

```
char * txt,  
octad * PUBKEY,  
pktype pk,  
octad * SIG,  
pktype sg,  
octad * ISSUER,  
octad * SUBJECT )
```

logging Certificate details

#### Parameters

<i>txt</i>	preamble text
<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

#### 5.7.2.8 logServerResponse()

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

log client processing of a Server response

#### Parameters

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

#### 5.7.2.9 logAlert()

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

log Server Alert

#### Parameters

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

#### 5.7.2.10 logCipherSuite()

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

log Cipher Suite

##### Parameters

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

#### 5.7.2.11 logKeyExchange()

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

log Key Exchange Group

##### Parameters

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

#### 5.7.2.12 logSigAlg()

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

log Signature Algorithm

##### Parameters

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

## 5.8 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.*

### 5.8.1 Detailed Description

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

Author

Mike Scott

### 5.8.2 Function Documentation

#### 5.8.2.1 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 <i>O</i>
<i>len</i>	number of bytes in <i>m</i>

**5.8.2.2 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 <i>O</i>

**5.8.2.3 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

**5.8.2.4 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

**5.8.2.5 `OCT_kill()`**

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

Wipe clean an octad.

## Parameters

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

**5.8.2.6 `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

**5.8.2.7 `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

### 5.8.2.8 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

### 5.8.2.9 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

### 5.8.2.10 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



#### 5.8.2.11 `OCT_reverse()`

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

Reverse bytes in an octad.

##### Parameters

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

#### 5.8.2.12 `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

#### 5.8.2.13 `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

#### 5.8.2.14 `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

#### 5.8.2.15 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

#### 5.8.2.16 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

## 5.9 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

- int **TLS13\_full** (**Socket** &client, char \*hostname, **octad** &IO, **octad** &RMS, **crypto** &K\_send, **crypto** &K\_recv, **octad** &STS, **capabilities** &CPB, int &cipher\_suite, int &favourite\_group)  
*TLS 1.3 full handshake.*
- int **TLS13\_resume** (**Socket** &client, char \*hostname, **octad** &IO, **octad** &RMS, **crypto** &K\_send, **crypto** &K\_recv, **octad** &STS, **ticket** &T, **octad** &EARLY)  
*TLS 1.3 resumption handshake.*

### 5.9.1 Detailed Description

TLS 1.3 main client-side protocol functions.

Author

Mike Scott

### 5.9.2 Function Documentation

#### 5.9.2.1 TLS13\_full()

```
int TLS13_full (
    Socket & client,
    char * hostname,
    octad & IO,
    octad & RMS,
    crypto & K_send,
    crypto & K_recv,
    octad & STS,
    capabilities & CPB,
    int & cipher_suite,
    int & favourite_group )
```

TLS 1.3 full handshake.

## Parameters

<i>client</i>	the socket connection to the Server
<i>hostname</i>	the host name (URL) of the server
<i>IO</i>	a workspace octad to buffer Server input
<i>RMS</i>	a returned Resumption Master secret
<i>K_send</i>	a crypto context for encrypting application traffic to the server
<i>K_recv</i>	a crypto context for decrypting application traffic from the server
<i>STS</i>	server application traffic secret - may be updated
<i>CPB</i>	the client capabilities structure
<i>cipher_suite</i>	the cipher_suite used for the handshake
<i>favourite_group</i>	our preferred group, which may be updated on a handshake retry

## 5.9.2.2 TLS13\_resume()

```
int TLS13_resume (
    Socket & client,
    char * hostname,
    octad & IO,
    octad & RMS,
    crypto & K_send,
    crypto & K_recv,
    octad & STS,
    ticket & T,
    octad & EARLY )
```

TLS 1.3 resumption handshake.

## Parameters

<i>client</i>	the socket connection to the Server
<i>hostname</i>	the host name (URL) of the server
<i>IO</i>	a workspace octad to buffer Server input
<i>RMS</i>	a provided Resumption Master secret
<i>K_send</i>	a crypto context for encrypting application traffic to the server
<i>K_recv</i>	a crypto context for decrypting application traffic from the server
<i>STS</i>	server application traffic secret - may be updated
<i>T</i>	a resumption ticket (or pre-shared key)
<i>EARLY</i>	early data that can be immediately sent to the server (0-RTT data)

## 5.10 tls\_sal.h File Reference

Security Abstraction Layer for TLS.

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

## Data Structures

- struct [unihash](#)  
*Universal Hash structure.*

## Functions

- 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 libraries.*
- 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\\_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\\_hkdfExpandLabel](#) (int htype, octad \*OKM, int olen, octad \*PRK, octad \*Label, octad \*CTX)  
*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\\_hashProcess](#) (unihash \*h, int b)  
*Hash process a byte.*
- int [SAL\\_hashOutput](#) (unihash \*h, char \*d)  
*Hash finish and output.*
- void [SAL\\_aeadEncrypt](#) (crypto \*send, int hdrlen, char \*hdr, int pten, char \*pt, octad \*TAG)  
*AEAD encryption.*
- int [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\\_certSignatureVerify](#) (int sigAlg, octad \*CERT, octad \*SIG, octad \*PUBKEY)  
*Verify a generic certificate signature.*
- bool [SAL\\_tlsSignatureVerify](#) (int sigAlg, octad \*TRANS, octad \*SIG, octad \*PUBKEY)  
*Verify a generic TLS transcript signature.*
- void [SAL\\_tlsSignature](#) (int sigAlg, octad \*KEY, octad \*TRANS, octad \*SIG)  
*Apply a generic TLS transcript signature.*

### 5.10.1 Detailed Description

Security Abstraction Layer for TLS.

Author

Mike Scott

### 5.10.2 Function Documentation

#### 5.10.2.1 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

#### 5.10.2.2 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

### 5.10.2.3 `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

### 5.10.2.4 `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

### 5.10.2.5 `SAL_initLib()`

```
bool SAL_initLib ( )
```

Initialise libraries.

#### Returns

return true if successful, else false

### 5.10.2.6 `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

**5.10.2.7 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

**5.10.2.8 SAL\_randomByte()**

```
int SAL_randomByte ( )
```

get a random byte

**Returns**

a random byte

**5.10.2.9 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

## 5.10.2.10 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

## 5.10.2.11 SAL\_hkdfExpandLabel()

```
void SAL_hkdfExpandLabel (
    int htype,
    octad * OKM,
    int olen,
    octad * PRK,
    octad * Label,
    octad * CTX )
```

Special HKDF Expand function (for TLS)

## Parameters

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

### 5.10.2.12 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

### 5.10.2.13 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

### 5.10.2.14 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

### 5.10.2.15 SAL\_hashProcess()

```
void SAL_hashProcess (
    unihash * h,
    int b )
```

Hash process a byte.

#### Parameters

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

### 5.10.2.16 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

### 5.10.2.17 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

**Parameters**

<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

**5.10.2.18 SAL\_aeadDecrypt()**

```
int 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**

-1 if tag is wrong, else 0

**5.10.2.19 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

### 5.10.2.20 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

### 5.10.2.21 SAL\_certSignatureVerify()

```
bool SAL_certSignatureVerify (
    int sigAlg,
    octad * CERT,
    octad * SIG,
    octad * PUBKEY )
```

Verify a generic certificate signature.

#### Parameters

<i>sigAlg</i>	the signature type
<i>CERT</i>	the input certificate that was signed
<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

### 5.10.2.22 SAL\_tlsSignatureVerify()

```
bool SAL_tlsSignatureVerify (
    int sigAlg,
```

```

    octad * TRANS,
    octad * SIG,
    octad * PUBKEY )

```

Verify a generic TLS transcript signature.

#### Parameters

<i>sigAlg</i>	the signature type
<i>TRANS</i>	the input transcript hash that was signed
<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

#### 5.10.2.23 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

## 5.11 tls\_sockets.h File Reference

set up sockets for reading and writing

```

#include <string.h>
#include "tls_logger.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 [getBytes](#) ([Socket](#) &client)  
*receive a single byte from a socket*
- int [getOctad](#) ([Socket](#) &client, [octad](#) \*B, int expected)  
*receive an octet from a socket*

### 5.11.1 Detailed Description

set up sockets for reading and writing

Author

Mike Scott

### 5.11.2 Function Documentation

#### 5.11.2.1 `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>to ms</i>	the time-out period in milliseconds

**Returns**

the socket handle

**5.11.2.2 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

**5.11.2.3 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



#### 5.11.2.4 `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

#### 5.11.2.5 `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

#### 5.11.2.6 `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

**5.11.2.7 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

**5.11.2.8 getByte()**

```
int getByte (
    Socket & client )
```

receive a single byte from a socket

**Parameters**

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

**Returns**

a byte

**5.11.2.9 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

## 5.12 `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, int cipher\_suite, int favourite\_group)  
*initialize a ticket structure, include time of creation*

#### 5.12.1 Detailed Description

TLS 1.3 process resumption tickets.

## Author

Mike Scott

#### 5.12.2 Function Documentation

##### 5.12.2.1 `millis()`

```
unsigned long millis ( )
```

read milliseconds from a stop-watch

## Returns

milliseconds read from stop-watch

### 5.12.2.2 parseTicket()

```
int parseTicket (
    octad * TICK,
    uint32_t 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

### 5.12.2.3 initTicketContext()

```
void initTicketContext (
    ticket * T,
    int cipher_suite,
    int favourite_group )
```

initialize a ticket structure, include time of creation

#### Parameters

<i>T</i>	the ticket structure
<i>cipher_suite</i>	the cipher suite currently in use
<i>favourite_group</i>	the server/client agreed group

## 5.13 tls\_wifi.h File Reference

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

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

### 5.13.1 Detailed Description

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

#### Author

Mike Scott

## 5.14 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\\_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](#)

### 5.14.1 Detailed Description

X509 function Header File.

Author

Mike Scott

defines structures declares functions

### 5.14.2 Macro Definition Documentation

#### 5.14.2.1 X509\_ECC

```
#define X509_ECC 1
```

Elliptic Curve data type detected

#### 5.14.2.2 X509\_RSA

```
#define X509_RSA 2
```

RSA data type detected

#### 5.14.2.3 X509\_ECD

```
#define X509_ECD 3
```

Elliptic Curve (Ed25519) detected

#### 5.14.2.4 X509\_H256

```
#define X509_H256 2
```

SHA256 hash algorithm used

#### 5.14.2.5 X509\_H384

```
#define X509_H384 3
```

SHA384 hash algorithm used

#### 5.14.2.6 X509\_H512

```
#define X509_H512 4
```

SHA512 hash algorithm used

#### 5.14.2.7 USE\_NIST256

```
#define USE_NIST256 0
```

For the NIST 256-bit standard curve - WEIERSTRASS only

#### 5.14.2.8 USE\_C25519

```
#define USE_C25519 1
```

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

#### 5.14.2.9 USE\_NIST384

```
#define USE_NIST384 10
```

For the NIST 384-bit standard curve - WEIERSTRASS only

#### 5.14.2.10 USE\_NIST521

```
#define USE_NIST521 12
```

For the NIST 521-bit standard curve - WEIERSTRASS only

### 5.14.3 Function Documentation

#### 5.14.3.1 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)

**5.14.3.2 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)

**5.14.3.3 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

**5.14.3.4 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)

**5.14.3.5 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

**5.14.3.6 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

**5.14.3.7 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

**5.14.3.8 X509\_self\_signed()**

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

**Parameters**

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

**Returns**

true if self-signed, else false

**5.14.3.9 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

#### 5.14.3.10 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

#### 5.14.3.11 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

#### 5.14.3.12 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

#### 5.14.3.13 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

#### 5.14.3.14 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

### 5.14.4 Variable Documentation

#### 5.14.4.1 X509\_CN

```
octad X509_CN [extern]
```

Country Name

#### 5.14.4.2 X509\_ON

`octad` X509\_ON [extern]

organisation Name

#### 5.14.4.3 X509\_EN

`octad` X509\_EN [extern]

email

#### 5.14.4.4 X509\_LN

`octad` X509\_LN [extern]

local name

#### 5.14.4.5 X509\_UN

`octad` X509\_UN [extern]

Unit name (aka Organisation Unit OU)

#### 5.14.4.6 X509\_MN

`octad` X509\_MN [extern]

My Name (aka Common Name)

#### 5.14.4.7 X509\_SN

`octad` X509\_SN [extern]

State Name

#### 5.14.4.8 X509\_AN

`octad` X509\_AN [extern]

Alternate Name

#### 5.14.4.9 X509\_KU

`octad` X509\_KU [extern]

Key Usage

#### 5.14.4.10 X509\_BC

`octad` X509\_BC [extern]

Basic Constraints

