TII TLS1.3

1.1

Generated by Doxygen 1.9.1

1 Description 1

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

2 Data Structure Index 7

2.1 Data Structures 7

3 File Index 9

3.1 File List 9

4 Data Structure Documentation 11

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_resp Struct Reference 14

4.3.1 Detailed Description 14

4.4 octad Struct Reference 14

4.4.1 Detailed Description 15

4.4.2 Field Documentation 15

4.4.2.1 len 15

4.4.2.2 max 15
4.4.2.3 val 15
4.5 pktype Struct Reference 15
4.5.1 Detailed Description 16
4.5.2 Field Documentation 16
4.5.2.1 type 16
4.5.2.2 hash 16
4.5.2.3 curve 16
4.6 ret Struct Reference 16
4.6.1 Detailed Description 17
4.6.2 Field Documentation 17
4.6.2.1 val 17
4.6.2.2 err 17
4.7 Socket Class Reference 17
4.7.1 Detailed Description 18
4.8 ticket Struct Reference 18
4.8.1 Detailed Description 18
4.8.2 Field Documentation 18
4.8.2.1 tick 18
4.8.2.2 nonce 19
4.8.2.3 TICK 19
4.8.2.4 NONCE 19
4.8.2.5 lifetime 19
4.8.2.6 age_obfuscator 19
4.8.2.7 max_early_data 19
4.8.2.8 birth 19
4.8.2.9 cipher_suite 19
4.8.2.10 favourite_group 20
4.9 unihash Struct Reference 20
4.9.1 Detailed Description 20
4.9.2 Field Documentation 20
4.9.2.1 state 20
4.9.2.2 htype 20

5 File Documentation **21**
5.1 tls1_3.h File Reference 21
5.1.1 Detailed Description 23
5.1.2 Macro Definition Documentation 24
5.1.2.1 IO_NONE 24
5.1.2.2 IO_APPLICATION 24
5.1.2.3 IO_PROTOCOL 24
5.1.2.4 IO_DEBUG 24

5.1.2.5 IO_WIRE 24
5.1.2.6 VERBOSITY 24
5.1.2.7 THIS_YEAR 24
5.1.2.8 HAVE_A_CLIENT_CERT 25
5.1.2.9 TLS_PROTOCOL 25
5.1.2.10 TLS_AES_128 25
5.1.2.11 TLS_AES_256 25
5.1.2.12 TLS_CHA_256 25
5.1.2.13 TLS_MAX_HASH_STATE 25
5.1.2.14 TLS_MAX_HASH 25
5.1.2.15 TLS_MAX_KEY 25
5.1.2.16 TLS_X509_MAX_FIELD 26
5.1.2.17 TLS_MAX_ROOT_CERT_SIZE 26
5.1.2.18 TLS_MAX_ROOT_CERT_B64 26
5.1.2.19 TLS_MAX_MYCERT_SIZE 26
5.1.2.20 TLS_MAX_MYCERT_B64 26
5.1.2.21 TLS_MAX_CLIENT_HELLO 26
5.1.2.22 TLS_MAX_EXT_LABEL 26
5.1.2.23 TLS_MAX_TICKET_SIZE 26
5.1.2.24 TLS_MAX_EXTENSIONS 27
5.1.2.25 TLS_MAX_IO_SIZE 27
5.1.2.26 TLS_MAX_SIGNATURE_SIZE 27
5.1.2.27 TLS_MAX_PUB_KEY_SIZE 27
5.1.2.28 TLS_MAX_SECRET_KEY_SIZE 27
5.1.2.29 TLS_MAX_ECC_FIELD 27
5.1.2.30 TLS_IV_SIZE 27
5.1.2.31 TLS_TAG_SIZE 28
5.1.2.32 TLS_MAX_COOKIE 28
5.1.2.33 TLS_MAX_SERVER_NAME 28
5.1.2.34 TLS_MAX_SUPPORTED_GROUPS 28
5.1.2.35 TLS_MAX_SUPPORTED_SIGS 28
5.1.2.36 TLS_MAX_PSK_MODES 28
5.1.2.37 TLS_MAX_CIPHER_SUITES 28
5.1.2.38 TLS_AES_128_GCM_SHA256 28
5.1.2.39 TLS_AES_256_GCM_SHA384 29
5.1.2.40 TLS_CHACHA20_POLY1305_SHA256 29
5.1.2.41 X25519 29
5.1.2.42 SECP256R1 29
5.1.2.43 SECP384R1 29
5.1.2.44 ECDSA_SECP256R1_SHA256 29
5.1.2.45 ECDSA_SECP384R1_SHA384 29
5.1.2.46 RSA_PSS_RSAE_SHA256 29

5.1.2.47 RSA_PSS_RSAE_SHA384 30
5.1.2.48 RSA_PSS_RSAE_SHA512 30
5.1.2.49 RSA_PKCS1_SHA256 30
5.1.2.50 RSA_PKCS1_SHA384 30
5.1.2.51 RSA_PKCS1_SHA512 30
5.1.2.52 ED25519 30
5.1.2.53 PSKOK 30
5.1.2.54 PSKWECDHE 30
5.1.2.55 TLS1_0 31
5.1.2.56 TLS1_2 31
5.1.2.57 TLS1_3 31
5.1.2.58 SERVER_NAME 31
5.1.2.59 SUPPORTED_GROUPS 31
5.1.2.60 SIG_ALGS 31
5.1.2.61 SIG_ALGS_CERT 31
5.1.2.62 KEY_SHARE 31
5.1.2.63 PSK_MODE 32
5.1.2.64 PRESHARED_KEY 32
5.1.2.65 TLS_VER 32
5.1.2.66 COOKIE 32
5.1.2.67 EARLY_DATA 32
5.1.2.68 MAX_FRAG_LENGTH 32
5.1.2.69 PADDING 32
5.1.2.70 APP_PROTOCOL 32
5.1.2.71 HSHAKE 33
5.1.2.72 APPLICATION 33
5.1.2.73 ALERT 33
5.1.2.74 CHANGE_CIPHER 33
5.1.2.75 TIME_OUT 33
5.1.2.76 HANDSHAKE_RETRY 33
5.1.2.77 STRANGE_EXTENSION 33
5.1.2.78 CLIENT_HELLO 34
5.1.2.79 SERVER_HELLO 34
5.1.2.80 CERTIFICATE 34
5.1.2.81 CERT_REQUEST 34
5.1.2.82 CERT_VERIFY 34
5.1.2.83 FINISHED 34
5.1.2.84 ENCRYPTED_EXTENSIONS 34
5.1.2.85 TICKET 34
5.1.2.86 KEY_UPDATE 35
5.1.2.87 MESSAGE_HASH 35
5.1.2.88 END_OF_EARLY_DATA 35

5.1.2.89 NOT_TLS1_3 . . . 35
5.1.2.90 BAD_CERT_CHAIN . . . 35
5.1.2.91 ID_MISMATCH . . . 35
5.1.2.92 UNRECOGNIZED_EXT . . . 35
5.1.2.93 BAD_HELLO . . . 35
5.1.2.94 WRONG_MESSAGE . . . 36
5.1.2.95 MISSING_REQUEST_CONTEXT . . . 36
5.1.2.96 AUTHENTICATION_FAILURE . . . 36
5.1.2.97 BAD_RECORD . . . 36
5.1.2.98 BAD_TICKET . . . 36
5.1.2.99 ILLEGAL_PARAMETER . . . 36
5.1.2.100 UNEXPECTED_MESSAGE . . . 36
5.1.2.101 DECRYPT_ERROR . . . 36
5.1.2.102 BAD_CERTIFICATE . . . 37
5.1.2.103 UNSUPPORTED_EXTENSION . . . 37
5.1.3 Typedef Documentation . . . 37
5.1.3.1 byte . . . 37
5.1.3.2 sign8 . . . 37
5.1.3.3 sign16 . . . 37
5.1.3.4 sign32 . . . 37
5.1.3.5 sign64 . . . 37
5.1.3.6 unsign32 . . . 38
5.1.3.7 unsign64 . . . 38
5.2 tls_cacerts.h File Reference . . . 38
5.2.1 Detailed Description . . . 38
5.2.2 Variable Documentation . . . 38
5.2.2.1 myprivate . . . 38
5.2.2.2 mycert . . . 39
5.2.2.3 cacerts . . . 39
5.3 tls_cert_chain.h File Reference . . . 39
5.3.1 Detailed Description . . . 39
5.3.2 Macro Definition Documentation . . . 40
5.3.2.1 TLS_SHA256 . . . 40
5.3.2.2 TLS_SHA384 . . . 40
5.3.2.3 TLS_SHA512 . . . 40
5.3.3 Function Documentation . . . 40
5.3.3.1 CHECK_CERT_CHAIN() . . . 40
5.3.3.2 IS_SERVER_CERT_VERIFY() . . . 41
5.3.3.3 GET_CLIENT_KEY_AND_CERTCHAIN() . . . 41
5.3.3.4 CREATE_CLIENT_CERT_VERIFIER() . . . 41
5.4 tls_client_recv.h File Reference . . . 42
5.4.1 Detailed Description . . . 43

5.4.2 Function Documentation . . . 43
5.4.2.1 parseoctad() . . . 43
5.4.2.2 parseInt16() . . . 44
5.4.2.3 parseInt24() . . . 44
5.4.2.4 parseInt32() . . . 44
5.4.2.5 parseByte() . . . 45
5.4.2.6 parseoctadptr() . . . 45
5.4.2.7 getServerFragment() . . . 46
5.4.2.8 parseByteorPull() . . . 46
5.4.2.9 parseInt32orPull() . . . 47
5.4.2.10 parseInt24orPull() . . . 47
5.4.2.11 parseInt16orPull() . . . 47
5.4.2.12 parseoctadorPull() . . . 48
5.4.2.13 parseoctadorPullptr() . . . 48
5.4.2.14 getWhatsNext() . . . 49
5.4.2.15 getServerEncryptedExtensions() . . . 49
5.4.2.16 getServerCertVerify() . . . 50
5.4.2.17 getServerFinished() . . . 51
5.4.2.18 getServerHello() . . . 51
5.4.2.19 getCheckServerCertificateChain() . . . 52
5.4.2.20 getCertificateRequest() . . . 52
5.5 tls_client_send.h File Reference . . . 53
5.5.1 Detailed Description . . . 54
5.5.2 Function Documentation . . . 54
5.5.2.1 sendCCCS() . . . 54
5.5.2.2 addPreSharedKeyExt() . . . 55
5.5.2.3 addServerNameExt() . . . 55
5.5.2.4 addSupportedGroupsExt() . . . 55
5.5.2.5 addSigAlgsExt() . . . 56
5.5.2.6 addSigAlgsCertExt() . . . 56
5.5.2.7 addKeyShareExt() . . . 56
5.5.2.8 addALPNExt() . . . 57
5.5.2.9 addMFLExt() . . . 57
5.5.2.10 addPSKModesExt() . . . 57
5.5.2.11 addVersionExt() . . . 58
5.5.2.12 addPadding() . . . 58
5.5.2.13 addCookieExt() . . . 58
5.5.2.14 addEarlyDataExt() . . . 58
5.5.2.15 clientRandom() . . . 59
5.5.2.16 sessionID() . . . 59
5.5.2.17 cipherSuites() . . . 59
5.5.2.18 sendClientMessage() . . . 60

5.5.2.19 sendBinder() 60
5.5.2.20 sendClientHello() 61
5.5.2.21 sendClientAlert() 61
5.5.2.22 sendClientFinish() 62
5.5.2.23 sendClientCertificateChain() 62
5.5.2.24 sendClientCertVerify() 63
5.5.2.25 sendEndOfEarlyData() 63
5.5.2.26 alert_from_cause() 63
5.6 tls_keys_calc.h File Reference 64
5.6.1 Detailed Description 65
5.6.2 Function Documentation 65
5.6.2.1 running_hash() 65
5.6.2.2 transcript_hash() 65
5.6.2.3 running_syn_hash() 65
5.6.2.4 init_crypto_context() 66
5.6.2.5 create_crypto_context() 66
5.6.2.6 increment_crypto_context() 66
5.6.2.7 GET_KEY_AND_IV() 67
5.6.2.8 RECOVER_PSK() 67
5.6.2.9 GET_EARLY_SECRET() 67
5.6.2.10 GET_LATER_SECRETS() 68
5.6.2.11 GET_HANDSHAKE_SECRETS() 68
5.6.2.12 GET_APPLICATION_SECRETS() 69
5.6.2.13 UPDATE_KEYS() 69
5.6.2.14 IS_VERIFY_DATA() 70
5.6.2.15 VERIFY_DATA() 70
5.7 tls_logger.h File Reference 71
5.7.1 Detailed Description 71
5.7.2 Function Documentation 71
5.7.2.1 myprintf() 72
5.7.2.2 logger() 73
5.7.2.3 logServerHello() 73
5.7.2.4 logTicket() 73
5.7.2.5 logEncExt() 74
5.7.2.6 logCert() 74
5.7.2.7 logCertDetails() 74
5.7.2.8 logServerResponse() 75
5.7.2.9 logAlert() 75
5.7.2.10 logCipherSuite() 76
5.7.2.11 logKeyExchange() 76
5.7.2.12 logSigAlg() 76
5.8 tls_octads.h File Reference 76

5.8.1 Detailed Description 77
5.8.2 Function Documentation 77
5.8.2.1 OCT_append_int() 77
5.8.2.2 OCT_append_octad() 78
5.8.2.3 OCT_compare() 78
5.8.2.4 OCT_shift_left() 78
5.8.2.5 OCT_kill() 79
5.8.2.6 OCT_from_hex() 79
5.8.2.7 OCT_append_string() 79
5.8.2.8 OCT_append_byte() 80
5.8.2.9 OCT_append_bytes() 80
5.8.2.10 OCT_from_base64() 80
5.8.2.11 OCT_reverse() 81
5.8.2.12 OCT_truncate() 81
5.8.2.13 OCT_copy() 81
5.8.2.14 OCT_output_hex() 81
5.8.2.15 OCT_output_string() 82
5.8.2.16 OCT_output_base64() 82
5.9 tls_protocol.h File Reference 83
5.9.1 Detailed Description 83
5.9.2 Function Documentation 83
5.9.2.1 TLS13_full() 83
5.9.2.2 TLS13_resume() 84
5.10 tls_sal.h File Reference 84
5.10.1 Detailed Description 86
5.10.2 Function Documentation 86
5.10.2.1 TLS_SAL_CIPHERS() 86
5.10.2.2 TLS_SAL_GROUPS() 86
5.10.2.3 TLS_SAL_SIGS() 87
5.10.2.4 TLS_SAL_SIGCERTS() 87
5.10.2.5 TLS_SAL_INITLIB() 87
5.10.2.6 TLS_SAL_HASHTYPE() 87
5.10.2.7 TLS_SAL_HASHLEN() 88
5.10.2.8 TLS_RANDOM_BYTE() 88
5.10.2.9 TLS_RANDOM_OCTAD() 88
5.10.2.10 TLS_HKDF_Extract() 89
5.10.2.11 TLS_HKDF_Expand_Label() 89
5.10.2.12 TLS_HMAC() 90
5.10.2.13 TLS_HASH_NULL() 90
5.10.2.14 Hash_Init() 90
5.10.2.15 Hash_Process() 91
5.10.2.16 Hash_Output() 91

5.10.2.17 AEAD_ENCRYPT() 91
5.10.2.18 AEAD_DECRYPT() 92
5.10.2.19 GENERATE_KEY_PAIR() 92
5.10.2.20 GENERATE_SHARED_SECRET() 93
5.10.2.21 CERT_SIGNATURE_VERIFY() 93
5.10.2.22 TLS_SIGNATURE_VERIFY() 93
5.10.2.23 TLS_SIGNATURE_SIGN() 94
5.11 tls_sockets.h File Reference 94
5.11.1 Detailed Description 95
5.11.2 Function Documentation 95
5.11.2.1 setclientsock() 95
5.11.2.2 getIPaddress() 96
5.11.2.3 sendOctad() 96
5.11.2.4 sendLen() 97
5.11.2.5 getBytes() 97
5.11.2.6 getInt16() 97
5.11.2.7 getInt24() 98
5.11.2.8 getByte() 98
5.11.2.9 getOctad() 98
5.12 tls_tickets.h File Reference 99
5.12.1 Detailed Description 99
5.12.2 Function Documentation 99
5.12.2.1 millis() 99
5.12.2.2 parseTicket() 100
5.12.2.3 init_ticket_context() 100
5.13 tls_wifi.h File Reference 100
5.13.1 Detailed Description 100
5.14 tls_x509.h File Reference 101
5.14.1 Detailed Description 102
5.14.2 Macro Definition Documentation 102
5.14.2.1 X509_ECC 102
5.14.2.2 X509_RSA 102
5.14.2.3 X509_ECD 102
5.14.2.4 X509_H256 102
5.14.2.5 X509_H384 102
5.14.2.6 X509_H512 103
5.14.2.7 USE_NIST256 103
5.14.2.8 USE_C25519 103
5.14.2.9 USE_NIST384 103
5.14.2.10 USE_NIST521 103
5.14.3 Function Documentation 103
5.14.3.1 X509_extract_private_key() 103

5.14.3.2 X509_extract_cert_sig() 104
5.14.3.3 X509_extract_cert() 104
5.14.3.4 X509_extract_public_key() 104
5.14.3.5 X509_find_issuer() 105
5.14.3.6 X509_find_validity() 105
5.14.3.7 X509_find_subject() 105
5.14.3.8 X509_self_signed() 106
5.14.3.9 X509_find_entity_property() 106
5.14.3.10 X509_find_start_date() 107
5.14.3.11 X509_find_expiry_date() 107
5.14.3.12 X509_find_extensions() 107
5.14.3.13 X509_find_extension() 108
5.14.3.14 X509_find_alt_name() 108
5.14.4 Variable Documentation 108
5.14.4.1 X509_CN 108
5.14.4.2 X509_ON 109
5.14.4.3 X509_EN 109
5.14.4.4 X509_LN 109
5.14.4.5 X509_UN 109
5.14.4.6 X509_MN 109
5.14.4.7 X509_SN 109
5.14.4.8 X509_AN 109
5.14.4.9 X509_KU 109
5.14.4.10 X509_BC 109

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+TIIcrypto

```
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/`

Then 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 to Server -> 16030100E5010000E1030348219C47B76BC8AD19E17DDB260CAA45108FBDFA75D982E04644AB1A88CDA0FF2055
Client Hello sent
Handshake Retry Request
Cipher Suite is TLS_AES_128_GCM_SHA256
Server HelloRetryRequest= 020000540303CF21AD74E59A6111BE1D8C021E65B891C2A211167ABB8C5E079E09E2C8A8339C20557742
Client to Server -> 1603030106010001020303364BA9C0C2B702B16F320C386D9E10F7619183314D2C09F36F97C8D24FBF2973720A0
Server Hello= 020000970303267374C6F6ABC13209CF0317E0D4C54C0B53769668B0471C9F0C99E26292657720A0E658C6A5BB912768
Parsing serverHello
Cipher Suite is TLS_AES_128_GCM_SHA256
Key Exchange Group is SECP256R1
Server Public Key= 04DBDAC433E340CCE52AF4AEE4D92C5D02A1CA3A0AC0A4A936000CD07E574D71120ED94E76B00ADAD0527CA1C55

Shared Secret= D2F2C6A7168BD9148F4F9EF8AED6A98167C6E4FBB622DD33A3BE74145A1B6F07
Handshake Secret= 2437B991C1DC51AAA35ED8A49B7FA01F110C59A54B379A1E37A080C04EBEF9D4
Client handshake traffic secret= B183E2BD906E2CE4AC4B50CF36A616637D30BFE0288006E25AC456C11940ABF5
Server handshake traffic secret= ABEC73EF21349C6139D033565330EE8CFBF5602DF3E7D2FEC1509F8B4DE19728
Warning - ALPN extension NOT acknowledged by server
Server Name NOT Acknowledged
Encrypted Extensions Processed
Certificate Chain Length= 2458
Parsing Server certificate
Signature is 0A5C155DB6DD9F7F6ABE005D351D6E3FF9DEBA799F7479BD33E1C784B63CB4CA695A76815C9B666C24B6E989EE85009A6
RSA signature of length 2048
Public key= E2AB76AE1A676E3268E39BB9B8AE9CA19DD8BC0BFED0A4275E13C191D716794B48F47766A6B6AD17F19764F48D459E8271
RSA public key of length 2048
Issuer is  R3/Let's Encrypt/
Subject is swifttls.org//
Parsing Intermediate certificate
Signature is D94CE0C9F584883731DBBB13E2B3FC8B6B62126C58B7497E3C02B7A81F2861EBCEE02E73EF49077A35841F1DAD68F0D8F
RSA signature of length 2048
Public key= BB021528CCF6A094D30F12EC8D5592C3F882F199A67A4288A75D26AAB52BB9C54CB1AF8E6BF975C8A3D70F479414553557
RSA public key of length 2048
Issuer is  DST Root CA X3/Digital Signature Trust Co./
Subject is R3/Let's Encrypt/
Signature  = 0A5C155DB6DD9F7F6ABE005D351D6E3FF9DEBA799F7479BD33E1C784B63CB4CA695A76815C9B666C24B6E989EE85009A6
Public key = BB021528CCF6A094D30F12EC8D5592C3F882F199A67A4288A75D26AAB52BB9C54CB1AF8E6BF975C8A3D70F47941455355
Checking Signature on Cert
Signature Algorithm is RSA_PKCS1_SHA256
Cert Signature Verification succeeded
Intermediate Certificate Chain sig is OK

Public Key from root cert= DFAFE99750088357B4CC6265F69082ECC7D32C6B30CA5BECD9C37DC740C118148BE0E83376492AE33F2
```

```
Signature  = D94CE0C9F584883731DBBB13E2B3FC8B6B62126C58B7497E3C02B7A81F2861EBCEE02E73EF49077A35841F1DAD68F0D8F
Public key = DFAFE99750088357B4CC6265F69082ECC7D32C6B30CA5BECD9C37DC740C118148BE0E83376492AE33F214993AC4E0EAF3
Checking Signature on Cert
Signature Algorithm is RSA_PKCS1_SHA256
Cert Signature Verification succeeded
Root Certificate sig is OK
Certificate Chain is valid
Transcript Hash= 009914583333AEBD14E04F9960BC9E4F1DA264B283D13AE5A830D816B9E0FE4E
Transcript Hash= 20FE6702F64323FC8D2F6FA56E4F02B3EC89EF1063278C378DE887E64A52023B
Server Certificate Signature= 8FA8AAE331DEB9EAEC746CA31ABD293C4BABBFE70745A9F491AC6E96B96FF3DB3942BBA990052B50
Signature Algorithm is RSA_PSS_RSAE_SHA256
Server Cert Verification OK

Server Data is verified
Transcript Hash= 23542FEAA48D469273FCF0F73CF3D62C026CCFC9B5AF03A5DC3E4F17425A45E6
Client Verify Data= 356518A7CE91F351D4E51E2CCF7BAE6F45DDE18710F11F5B82D3D37D278D7A29
Client to Server -> 17030300449A2F8BA35986761B0C1C5ADAEE9C78FC2D5DCF96FD8844BC84665CED01AB732D017C7F931362D31E
Client application traffic secret= 75FF1BDEE81A66C4DF287CAE3C7CF64D2A66A664FC423DC4392D03BBC27CE707
Server application traffic secret= 2E437FA0058054C34953568BD48F3CD869CB750894ADDE6407A9105DF8B87E83
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
Connection closed

Attempting resumption

Parsing Ticket
life time in minutes = 60
Age obfuscator = e01e88b5
Nonce = 00
Ticket = B5C413F5D46D4FAB1EA31CED3F342B9887425F5E4E33C4E9F71CA410DD1C0B93
max_early_data = 40960

PSK= B7AE97009D6FB27BE6FE91AA46B40FDA05B109F12911BF878CCF71AB79EE93B3
Binder Key= 5E1361389B77C42A9E8C1D9FC0FC01E5FB8B6CBC7C963679390BA06858E64941
Early Secret= DF69A43C7507B123F59ED427926A33D6A559167AC0B6201E1924B36C519081A1
Private key= B2BE24A6B02F166305D4B3A5CC644BAFD31EADDDF28EA783EA5850FB046D230E
Client Public key= 04DD8091A6A7134225F56F520450B3773A8B689F8E9090399D06C916DFFC4179F236F1ACA3F97B5794D6D5E7ADA
Ticket age= 34
obfuscated age = e01e88e9
Client to Server -> 16030301140100013303030C20004D98AF9D915CE6807F3766E804D3F3EA72A6FDE3883C72786C21507CEEC2077
Client Hello sent
BND= 62311EA9BBD18AA7929D1F103F37BB49756E8FA97F4F2F918E27656D005BCCA3
Sending Binders
Client to Server -> 160303002300212062311EA9BBD18AA7929D1F103F37BB49756E8FA97F4F2F918E27656D005BCCA3
Client Early Traffic Secret= 096D42B97067974660668C971BA2E3890A7BCDD10DEC6721354CB8D4FECD2648
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= 045AD14B24FBEB2A56E4D9FF050A6372ECB329A9907FE7739C56CE6BEF09FDAD51623E23B3A2DFA739274D4FDF1

serverHello= 0200009D0303267374C78BD760AA7A7C97B9B0C2513805B09653E45A3CDB76E1EC53FDF7E6DD2077898C8276DD669FF94
Shared Secret= 724F76AAA8157BBC6FC072B70BF26054F62EE1E831E4FBBC02765EDD087B6011
Handshake Secret= AD049FAE721797DD0078436FD92139172D2C5570C52CF991D3EF795D472B701E
Client handshake traffic secret= 243DDFF0AD49EBA19D7FD2EECE3718B3B8FDD172CF833F705EABFACC6545A38A
Server handshake traffic secret= 829E8F8A1EA13C7C7F603E4DD61C728D7B99DC21AD91F2FA55B7A2C6661A9268
```

```
Early Data Accepted
Warning - ALPN extension NOT acknowledged by server
Server Name NOT Acknowledged
Transcript Hash= FA775FF04D6FEE07DD9935D2A4BCB60B717ED0C6CB1E34CC40306570A45C2BA4
Send End of Early Data
Client to Server -> 170303002044B3C7F3A5964D67212A693DE6E04951DF5F0CB0EAA7A59937B62B08B1B6C8B0
Transcript Hash= 4B287F921879798321E8DC1886FD8C5898BE3DA7E2DE18E13353100E03671EB3
Server Data is verified
Client Verify Data= 71A0D71AA782C671D34DD261CD6E49B6C40742308DF688A7637688EC00782503
Client to Server -> 170303003908D00F304E8B7113C41BA532F4BE7D139F8681FEFE9A9C814AAE7B3C8986B5406DE4E02FA80ED582
Client application traffic secret= C6A112E7C0873BE7C80AB08E5B0CFCCB9B8DFEB3309ED185A34B75A20C0FE9F0
Server application traffic secret= 86488942A19801E5A010E75BB935DDB06C9DCE5CDA4B704D5F8943EE0C28BB58
Resumption Handshake succeeded
Early data was accepted
Waiting for Server input
Receiving application data (truncated HTML) = 485454502F312E3120323030204F4B0D0A5365727665723A205377696674544C
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 quite fragile (only tested and debugged aginst a dozen websites or so!), and would be expected to often fail. In a small number of cases it will fail due to receiving a malformed certificate chain from the Server.

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.

1.1 Client side Authentication

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

```
openssl req -x509 -nodes -days 3650 -newkey ec:<(openssl ecparam -name secp384r1) -keyout mykey.pem -out mycer
```

Another 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 tls13.1d.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

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 2,3,8,40 and 42
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:

capabilities
Cryptographic capabilities of the client 11
crypto
Crypto context structure 13
ee_expt
Server encrypted extensions expectations . . . . . **??**
ee_resp
Server encrypted extensions responses 14
octad
Safe representation of an octad 14
pktype
Public key type 15
ret
Function return structure 16
Socket
Socket instance 17
ticket
Ticket context structure 18
unihash
Universal Hash structure 20

Chapter 3

File Index

3.1 File List

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

tls1_3.h
Main TLS 1.3 Header File for constants and structures 21
tls_cacerts.h
Certificate Authority root certificate store 38
tls_cert_chain.h
Process Certificate Chain 39
tls_client_recv.h
Process Input received from the Server 42
tls_client_send.h
Process Output to be sent to the Server 53
tls_keys_calc.h
TLS 1.3 crypto support functions 64
tls_logger.h
TLS 1.3 logging 71
tls_octads.h
Octad handling routines - octads don't overflow, they truncate 76
tls_protocol.h
TLS 1.3 main client-side protocol functions 83
tls_sal.h
Security Abstraction Layer for TLS 84
tls_sockets.h
Set up sockets for reading and writing 94
tls_tickets.h
TLS 1.3 process resumption tickets 99
tls_wifi.h
Define Socket structure depending on processor context 100
tls_x509.h
X509 function Header File 101

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 Certicates

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

`unsign32 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

```
unsign32 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 TIME_OUT 0x01
- #define HANDSHAKE_RETRY 0x02
- #define STRANGE_EXTENSION 0x03
- #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 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 ILLEGAL_PARAMETER 0x2F
- #define UNEXPECTED_MESSAGE 0x0A
- #define DECRYPT_ERROR 0x33
- #define BAD_CERTIFICATE 0x2A
- #define UNSUPPORTED_EXTENSION 0x6E
- #define UNKNOWN_CA 0x30

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 traffic

5.1.2.5 IO_WIRE

```
#define IO_WIRE 4
```

print lots of debug information + protocol progress + application traffic + 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 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 TIME_OUT

```
#define TIME_OUT 0x01
```

Time-out

5.1.2.77 HANDSHAKE_RETRY

```
#define HANDSHAKE_RETRY 0x02
```

Handshake retry

5.1.2.78 STRANGE_EXTENSION

```
#define STRANGE_EXTENSION 0x03
```

Strange extension

5.1.2.79 CLIENT_HELLO

```
#define CLIENT_HELLO 0x01
```

Client Hello message

5.1.2.80 SERVER_HELLO

```
#define SERVER_HELLO 0x02
```

Server Hello message

5.1.2.81 CERTIFICATE

```
#define CERTIFICATE 0x0b
```

Certificate message

5.1.2.82 CERT_REQUEST

```
#define CERT_REQUEST 0x0d
```

Certificate Request

5.1.2.83 CERT_VERIFY

```
#define CERT_VERIFY 0x0f
```

Certificate Verify message

5.1.2.84 FINISHED

```
#define FINISHED 0x14
```

Handshae Finished message

5.1.2.85 ENCRYPTED_EXTENSIONS

```
#define ENCRYPTED_EXTENSIONS 0x08
```

Encrypted Extensions message

5.1.2.86 TICKET

```
#define TICKET 0x04
```

Ticket message

5.1.2.87 KEY_UPDATE

```
#define KEY_UPDATE 0x18
```

Key Update message

5.1.2.88 MESSAGE_HASH

```
#define MESSAGE_HASH 0xFE
```

Special synthetic message hash message

5.1.2.89 END_OF_EARLY_DATA

```
#define END_OF_EARLY_DATA 0x05
```

End of Early Data message

5.1.2.90 NOT_TLS1_3

```
#define NOT_TLS1_3 -2
```

Wrong version error, not TLS1.3

5.1.2.91 BAD_CERT_CHAIN

```
#define BAD_CERT_CHAIN -3
```

Bad Certificate Chain error

5.1.2.92 ID_MISMATCH

```
#define ID_MISMATCH -4
```

Session ID mismatch error

5.1.2.93 UNRECOGNIZED_EXT

```
#define UNRECOGNIZED_EXT -5
```

Unrecognised extension error

5.1.2.94 BAD_HELLO

```
#define BAD_HELLO -6
```

badly formed Hello message error

5.1.2.95 WRONG_MESSAGE

```
#define WRONG_MESSAGE -7
```

Message out-of-order error

5.1.2.96 MISSING_REQUEST_CONTEXT

```
#define MISSING_REQUEST_CONTEXT -8
```

Request context missing error

5.1.2.97 AUTHENTICATION_FAILURE

```
#define AUTHENTICATION_FAILURE -9
```

Authentication error - AEAD Tag incorrect

5.1.2.98 BAD_RECORD

```
#define BAD_RECORD -10
```

Badly formed Record received

5.1.2.99 BAD_TICKET

```
#define BAD_TICKET -11
```

Badly formed Ticket received

5.1.2.100 NOT_EXPECTED

```
#define NOT_EXPECTED -12
```

Received ack for something not requested

5.1.2.101 CA_NOT_FOUND

```
#define CA_NOT_FOUND -13
```

Certificate Authority not found

5.1.2.102 ILLEGAL_PARAMETER

```
#define ILLEGAL_PARAMETER 0x2F
```

Illegal parameter alert

5.1.2.103 UNEXPECTED_MESSAGE

```
#define UNEXPECTED_MESSAGE 0x0A
```

Unexpected message alert

5.1.2.104 DECRYPT_ERROR

```
#define DECRYPT_ERROR 0x33
```

Decryption error alert

5.1.2.105 BAD_CERTIFICATE

```
#define BAD_CERTIFICATE 0x2A
```

Bad certificate alert

5.1.2.106 UNSUPPORTED_EXTENSION

```
#define UNSUPPORTED_EXTENSION 0x6E
```

Unsupported extension alert

5.1.2.107 UNKNOWN_CA

```
#define UNKNOWN_CA 0x30
```

Unrecognised Certificate Authority

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 CHECK_CERT_CHAIN (octad ∗CERTCHAIN, char ∗hostname, octad ∗PUBKEY)

  Check Certificate Chain.
- bool IS_SERVER_CERT_VERIFY (int sigalg, octad ∗SCVSIG, octad ∗H, octad ∗CERTPK)

  verify Server's signature on protocol transcript
- int GET_CLIENT_KEY_AND_CERTCHAIN (int nccsalgs, int ∗csigAlgs, octad ∗PRIVKEY, octad ∗CERTCHAIN)

  Get Client private key and Certificate chain from .pem files.
- void CREATE_CLIENT_CERT_VERIFIER (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 CHECK_CERT_CHAIN()

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

Check Certificate Chain.

Parameters

CERTCHAIN	the input certificate chain
hostname	the input Server name associated with the Certificate chain
PUBKEY	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 IS_SERVER_CERT_VERIFY()

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

verify Server's signature on protocol transcript

Parameters

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

Returns

true if signature is verified, else returns false

5.3.3.3 GET_CLIENT_KEY_AND_CERTCHAIN()

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

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

Parameters

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

Returns

type of private key, ECC or RSA

5.3.3.4 CREATE_CLIENT_CERT_VERIFIER()

```
void CREATE_CLIENT_CERT_VERIFIER (
             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

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

5.4 tls_client_recv.h File Reference

Process Input received from the Server.

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

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.
- int getWhatsNext (Socket &client, octad ∗IO, crypto ∗recv, unihash ∗trans_hash)

  Identify type of message.
- int 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.
- int 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.
- int getServerFinished (Socket &client, octad ∗IO, crypto ∗recv, unihash ∗trans_hash, octad ∗HFIN)

  Get final handshake message from Server, a HMAC on the transcript hash.
- int getServerHello (Socket &client, octad ∗SH, int &cipher, int &kex, octad ∗CID, octad ∗CK, octad ∗PK, int &pskid)

  Receive and parse initial Server Hello.

- int getCheckServerCertificateChain (Socket &client, octad ∗IO, crypto ∗recv, unihash ∗trans_hash, char ∗hostname, octad ∗PUBKEY)

  Receive and check certificate chain.
- int 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

E	the output octad copied out from the octad M
len	the expected length of the output octad E
M	the input octad
ptr	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

M	the input octad
ptr	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

M	the input octad
ptr	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

M	the input octad
ptr	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

M	the input octad
ptr	a pointer into M, 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

E	a pointer to an octad contained within an octad M
len	the expected length of the octad E
M	the input octad
ptr	a pointer into M, 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

client	the socket connection to the Server
recv	the cryptographic key under which the fragment is encrypted, or NULL if not encrypted
IO	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

client	the socket connection to the Server
IO	the input octad
ptr	a pointer into IO, which advances after use
recv	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

client	the socket connection to the Server
IO	the input octad
ptr	a pointer into IO, which advances after use
recv	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 * recv )
```

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

Parameters

client	the socket connection to the Server
IO	the input octad
ptr	a pointer into IO, which advances after use
recv	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 * recv )
```

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

Parameters

client	the socket connection to the Server
IO	the input octad
ptr	a pointer into IO, which advances after use
recv	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

client	the socket connection to the Server
O	the output octad
len	the expected length of the output octad O
IO	the input octad
ptr	a pointer into IO, which advances after use
recv	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

client	the socket connection to the Server
O	a pointer to an octad contained within an octad IO
len	the expected length of the octad O
IO	the input octad
ptr	a pointer into IO, which advances after use
recv	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 getWhatsNext()

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

Identify type of message.

Parameters

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

Returns

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

5.4.2.15 getServerEncryptedExtensions()

```
int 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

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

Returns

negative error, zero for OK, or positive for informative response

5.4.2.16 getServerCertVerify()

```
int 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

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

Returns

negative error, zero for OK, or positive for informative response

5.4.2.17 getServerFinished()

```
int 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

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

Returns

negative error, zero for OK, or positive for informative response

5.4.2.18 getServerHello()

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

Receive and parse initial Server Hello.

Parameters

client	the socket connection to the Server
SH	an octad to accept server input
cipher	the agreed cipher suite
kex	key exchange data
CID	random session identity
CK	an output Cookie
PK	the key exchange public value supplied by the Server
pskid	indicates if a pre-shared key was accepted, otherwise -1

Returns

negative error, zero for OK, or positive for informative response

5.4.2.19 getCheckServerCertificateChain()

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

Receive and check certificate chain.

Parameters

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

Returns

negative error, zero for OK, or positive for informative response

5.4.2.20 getCertificateRequest()

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

process a Certificate Request

Parameters

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

Returns

negative error, zero for OK, or positive for informative response

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 nsg, int ∗supportedGroups)

  Add Supported Groups extension to under-construction Extensions Octet.
- void addSigAlgsExt (octad ∗EXT, int nsa, int ∗sigAlgs)

  Add Supported TLS1.3 Signature algorithms to under-construction Extensions Octet.
- void addSigAlgsCertExt (octad ∗EXT, int nsac, int ∗sigAlgsCert)

  Add Supported X.509 Certificate Signature algorithms to under-construction Extensions Octet.
- void addKeyShareExt (octad ∗EXT, int alg, octad ∗PK)

  Add Key Share extension to under-construction Extensions Octet.
- void addALPNExt (octad ∗EXT, octad ∗AP)

  Add Application Layer Protocol Negotiation (ALPN) extension to under-construction Extensions Octet.
- void addMFLExt (octad ∗EXT, int mode)

  Add Maximum Fragment Length extension to under-construction Extensions Octet.
- void 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, octad ∗IO)

  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

client	the socket connection to the Server

5.5.2.2 addPreSharedKeyExt()

```
int addPreSharedKeyExt (
             octad * EXT,
             unsign32 age,
             octad * IDS,
             int sha )
```

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

Parameters

EXT	the extensions octad which is being built
age	the obfuscated age of the preshared key
IDS	the proposed preshared key identity
sha	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

EXT	the extensions octad which is being built
servername	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

EXT	the extensions octad which is being built
nsg	Number of supported groups
supportedGroups	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

EXT	the extensions octad which is being built
nsa	Number of supported signature algorithms
sigAlgs	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

EXT	the extensions octad which is being built
nsac	Number of supported signature algorithms
sigAlgsCert	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

EXT	the extensions octad which is being built
alg	the suggested key exchange algorithm
PK	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

EXT	the extensions octad which is being built
AP	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

EXT	the extensions octad which is being built
mode	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

EXT	the extensions octad which is being built
mode	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

EXT	the extensions octad which is being built
version	the supported TLS version

5.5.2.12 addPadding()

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

Add padding extension to under-construction Extensions Octet.

Parameters

EXT	the extensions octad which is being built
n	the zero padding length

5.5.2.13 addCookieExt()

```
void addCookieExt (
             octad * EXT,
             octad * CK )
```

Add Cookie extension to under-construction Extensions Octet.

Parameters

EXT	the extensions octad which is being built
CK	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

EXT	the extensions octad which is being built

5.5.2.15 clientRandom()

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

Generate 32-byte random octad.

Parameters

RN	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

SI	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

CS	the output cipher-suite octad
ncs	the number of supported cipher-suites
ciphers	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

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

client	the socket connection to the Server
B	workspace octad in which to construct binder message
BND	binding HMAC of truncated transcript hash
IO	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

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

5.5.2.21 sendClientAlert()

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

Prepare and send an Alert message to the Server.

Parameters

client	the socket connection to the Server
type	the type of the Alert
send	the cryptographic key under which the alert message is encrypted (or NULL if no encryption)
IO	the workspace octad in which to construct the overall message

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

client	the socket connection to the Server
send	the cryptographic key under which the verification message is encrypted
h	the current transcript hash up to this point
CHF	the client verify data HMAC
IO	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

client	the socket connection to the Server
send	the cryptographic key under which the certificate message is encrypted
h	the current transcript hash up to this point
CERTCHAIN	the client certificate chain
IO	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

client	the socket connection to the Server
send	the cryptographic key under which the certificate message is encrypted
h	the current transcript hash up to this point
sigAlg	the client's digital signature algorithm
CCVSIG	the client's signature
IO	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

client	the socket connection to the Server
send	the cryptographic key under which the message is encrypted
h	the current transcript hash up to this point
IO	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

rtn	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 running_hash (octad ∗O, unihash ∗h)

  Accumulate octad into ongoing hashing.
- void transcript_hash (unihash ∗h, octad ∗O)

  Output current hash value.
- void running_syn_hash (octad ∗O, octad ∗E, unihash ∗h)

  Calculate special synthetic hash calculation for first clientHello after retry request (RFC 8446 section 4.4.1)
- void init_crypto_context (crypto ∗C)

  Initiate a Crypto Context.
- void create_crypto_context (crypto ∗C, octad ∗K, octad ∗IV)

  Build a Crypto Context.
- void increment_crypto_context (crypto ∗C)

  Increment a Crypto Context for the next record, updating IV.
- void GET_KEY_AND_IV (int cipher_suite, octad ∗TS, crypto ∗context)

  Build a crypto context from an input raw Secret and an agreed cipher_suite.
- void RECOVER_PSK (int htype, octad ∗RMS, octad ∗NONCE, octad ∗PSK)

  Recover a pre-shared key from Resumption Master Secret and a nonce.
- void GET_EARLY_SECRET (int htype, octad ∗PSK, octad ∗ES, octad ∗BKE, octad ∗BKR)

  Extract Early Secret Key and Binder Key from Preshared Key (External or Resumption)
- void GET_LATER_SECRETS (int htype, octad ∗H, octad ∗ES, octad ∗CETS, octad ∗EEMS)

  Extract more secrets from Early Secret.
- void GET_HANDSHAKE_SECRETS (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 GET_APPLICATION_SECRETS (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 UPDATE_KEYS (crypto ∗context, octad ∗TS)

  Perform a Key Update on a crypto context.
- bool IS_VERIFY_DATA (int htype, octad ∗SF, octad ∗STS, octad ∗H)

  Test if data from Server is verified using server traffic secret and a transcript hash.
- void VERIFY_DATA (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 running_hash()

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

Accumulate octad into ongoing hashing.

Parameters

O	an octad to be included in hash
h	a hashing context

5.6.2.2 transcript_hash()

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

Output current hash value.

Parameters

h	a hashing context
O	an output octad containing current hash

5.6.2.3 running_syn_hash()

```
void running_syn_hash (
            octad * O,
            octad * E,
            unihash * h )
```

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

Parameters

O	an octad containing clientHello
E	an octad containing clientHello extensions
h	a hashing context

5.6.2.4 init_crypto_context()

```
void init_crypto_context (
             crypto * C )
```

Initiate a Crypto Context.

Parameters

C	an AEAD encryption context

5.6.2.5 create_crypto_context()

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

Build a Crypto Context.

Parameters

C	an AEAD encryption context
K	an encryption key
IV	an encryption Initialisation Vector

5.6.2.6 increment_crypto_context()

```
void increment_crypto_context (
             crypto * C )
```

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

Parameters

C	an AEAD encryption context

5.6.2.7 GET_KEY_AND_IV()

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

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

Parameters

cipher_suite	the chosen cipher suite
TS	the input raw secret
context	an AEAD encryption context

5.6.2.8 RECOVER_PSK()

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

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

Parameters

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

5.6.2.9 GET_EARLY_SECRET()

```
void GET_EARLY_SECRET (
            int htype,
            octad * PSK,
            octad * ES,
            octad * BKE,
            octad * BKR )
```

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

Parameters

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

5.6.2.10 GET_LATER_SECRETS()

```
void GET_LATER_SECRETS (
             int htype,
             octad * H,
             octad * ES,
             octad * CETS,
             octad * EEMS )
```

Extract more secrets from Early Secret.

Parameters

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

5.6.2.11 GET_HANDSHAKE_SECRETS()

```
void GET_HANDSHAKE_SECRETS (
             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

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

5.6.2.12 GET_APPLICATION_SECRETS()

```
void GET_APPLICATION_SECRETS (
            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

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

5.6.2.13 UPDATE_KEYS()

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

Perform a Key Update on a crypto context.

Parameters

context	an AEAD encryption context
TS	the updated Traffic secret

5.6.2.14 IS_VERIFY_DATA()

```
bool IS_VERIFY_DATA (
            int htype,
            octad * SF,
            octad * STS,
            octad * H )
```

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

Parameters

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

Returns

true is data is verified, else false

5.6.2.15 VERIFY_DATA()

```
void VERIFY_DATA (
             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

htype	hash algorithm
SF	the output verification data
CTS	the input Client Traffic Secret
H	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 (int rtn, octad ∗O)

  log the result of client processing of a Server response
- bool logAlert (octad ∗O)

  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

s	a string to be output

5.7.2.2 logger()

```
void logger (
            char * preamble,
            char * string,
            unsign32 info,
            octad * O )
```

basic logging function

Parameters

preamble	a string to be output
string	another string, or a format specifier for info, or NULL
info	an integer to be output
O	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

cipher_suite	the chosen cipher suite
kex	the chosen key exchange algorithm
pskid	the chosen preshared key (or -1 if none)
PK	the Server Public Key
CK	a Cookie (if any)

5.7.2.4 logTicket()

```
void logTicket (
            int lifetime,
            unsign32 age_obfuscator,
            unsign32 max_early_data,
            octad * NONCE,
            octad * ETICK )
```

logging a resumption ticket

Parameters

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

5.7.2.5 logEncExt()

```
void logEncExt (
            ee_expt * e,
            ee_resp * r )
```

logging server extended extensions responses vs expectations

Parameters

e	structure containing server expectations
r	structure containing server responses

5.7.2.6 logCert()

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

logging a Certificate in standard base 64 format

Parameters

CERT	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

txt	preamble text
PUBKEY	the certificate public key octad
pk	the public key type
SIG	the signature on the certificate
sg	the signature type
ISSUER	the (composite) certificate issuer
SUBJECT	the (composite) certificate subject

5.7.2.8 logServerResponse()

```
void logServerResponse (
             int rtn,
             octad * O )
```

log the result of client processing of a Server response

Parameters

rtn	the return value from Server response function processing
O	the server's raw response, might include alert indication

5.7.2.9 logAlert()

```
bool logAlert (
             octad * O )
```

log Server Alert

Parameters

O	the server's alert code

5.7.2.10 logCipherSuite()

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

log Cipher Suite

Parameters

cipher_suite	the Cipher Suite to be logged

5.7.2.11 logKeyExchange()

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

log Key Exchange Group

Parameters

kex	the Key Exchange Group to be logged

5.7.2.12 logSigAlg()

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

log Signature Algorithm

Parameters

sigAlg	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

O	octad to be appended to
x	integer to be appended to O
len	number of bytes in m

5.8.2.2 OCT_append_octad()

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

Join one octad to the end of another.

Parameters

O	octad to be appended to
P	octad to be joined to the end of O

5.8.2.3 OCT_compare()

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

Compare two octads.

Parameters

O	first octad to be compared
P	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

O	octad to be shifted
n	number of bytes to shift

5.8.2.5 OCT_kill()

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

Wipe clean an octad.

Parameters

O	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

O	octad
src	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

O	octad to be written to
s	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

O	octad to be written to
b	byte to be joined to end of octad
n	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

O	octad to be written to
s	byte array to be joined to end of octad
n	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

O	octad to be populated
b	zero terminated base64 string

5.8.2.11 OCT_reverse()

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

Reverse bytes in an octad.

Parameters

O	octad to be reversed

5.8.2.12 OCT_truncate()

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

Reverse bytes in an octad.

Parameters

O	octad to be truncated
n	the new shorter length

5.8.2.13 OCT_copy()

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

Copy one octad into another.

Parameters

O	octad to be copied to
P	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

O	octad to be output
max	the maximum output length
s	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

O	octad to be output
max	the maximum output length
s	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

O	octad to be output
max	the maximum output length
s	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

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

client	the socket connection to the Server
hostname	the host name (URL) of the server
IO	a workspace octad to buffer Server input
RMS	a provided Resumption Master secret
K_send	a crypto context for encrypting application traffic to the server
K_recv	a crypto context for decrypting application traffic from the server
STS	server application traffic secret - may be updated
T	a resumption ticket (or pre-shared key)
EARLY	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 TLS_SAL_CIPHERS (int ∗ciphers)

  Return supported ciphers.
- int TLS_SAL_GROUPS (int ∗groups)

  Return supported groups in preferred order.
- int TLS_SAL_SIGS (int ∗sigAlgs)

  Return supported TLS signature algorithms in preferred order.
- int TLS_SAL_SIGCERTS (int ∗sigAlgsCert)

  Return supported TLS signature algorithms for Certificates in preferred order.
- bool TLS_SAL_INITLIB ()

  Initialise libraries.
- int TLS_SAL_HASHTYPE (int cipher_suite)

  return hash type asspciated with a cipher suite
- int TLS_SAL_HASHLEN (int hash_type)

  return output length of hash function associated with a hash type
- int TLS_RANDOM_BYTE ()

  get a random byte
- void TLS_RANDOM_OCTAD (int len, octad ∗R)

  get a random octad
- void TLS_HKDF_Extract (int sha, octad ∗PRK, octad ∗SALT, octad ∗IKM)

  HKDF Extract function.
- void TLS_HKDF_Expand_Label (int htype, octad ∗OKM, int olen, octad ∗PRK, octad ∗Label, octad ∗CTX)

  Special HKDF Expand function (for TLS)
- void TLS_HMAC (int htype, octad ∗T, octad ∗K, octad ∗M)

  simple HMAC function
- void TLS_HASH_NULL (int sha, octad ∗H)

  simple HASH of nothing function
- void Hash_Init (int hlen, unihash ∗h)

  Initiate Hashing context.
- void Hash_Process (unihash ∗h, int b)

  Hash process a byte.
- int Hash_Output (unihash ∗h, char ∗d)

  Hash finish and output.
- void AEAD_ENCRYPT (crypto ∗send, int hdrlen, char ∗hdr, int ptlen, char ∗pt, octad ∗TAG)

  AEAD encryption.
- int AEAD_DECRYPT (crypto ∗recv, int hdrlen, char ∗hdr, int ctlen, char ∗ct, octad ∗TAG)

  AEAD decryption.
- void GENERATE_KEY_PAIR (int group, octad ∗SK, octad ∗PK)

  generate a public/private key pair in an approved group for a key exchange
- void GENERATE_SHARED_SECRET (int group, octad ∗SK, octad ∗PK, octad ∗SS)

  generate a Diffie-Hellman shared secret
- bool CERT_SIGNATURE_VERIFY (int sigAlg, octad ∗CERT, octad ∗SIG, octad ∗PUBKEY)

  Verify a generic certificate signature.
- bool TLS_SIGNATURE_VERIFY (int sigAlg, octad ∗TRANS, octad ∗SIG, octad ∗PUBKEY)

  Verify a generic TLS transcript signature.
- void TLS_SIGNATURE_SIGN (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 TLS_SAL_CIPHERS()

```
int TLS_SAL_CIPHERS (
            int * ciphers )
```

Return supported ciphers.

Parameters

ciphers	array of supported ciphers in preferred order

Returns

number of supported ciphers

5.10.2.2 TLS_SAL_GROUPS()

```
int TLS_SAL_GROUPS (
            int * groups )
```

Return supported groups in preferred order.

Parameters

groups	array of supported groups

Returns

number of supported groups

5.10.2.3 TLS_SAL_SIGS()

```
int TLS_SAL_SIGS (
            int * sigAlgs )
```

Return supported TLS signature algorithms in preferred order.

Parameters

sigAlgs	array of supported signature algorithms

Returns

number of supported groups

5.10.2.4 TLS_SAL_SIGCERTS()

```
int TLS_SAL_SIGCERTS (
            int * sigAlgsCert )
```

Return supported TLS signature algorithms for Certificates in preferred order.

Parameters

sigAlgsCert	array of supported signature algorithms for Certificates

Returns

number of supported groups

5.10.2.5 TLS_SAL_INITLIB()

```
bool TLS_SAL_INITLIB ( )
```

Initialise libraries.

Returns

return true if successful, else false

5.10.2.6 TLS_SAL_HASHTYPE()

```
int TLS_SAL_HASHTYPE (
            int cipher_suite )
```

return hash type asspciated with a cipher suite

Parameters

cipher_suite	a TLS cipher suite

Returns

hash function output length

5.10.2.7 TLS_SAL_HASHLEN()

```
int TLS_SAL_HASHLEN (
             int hash_type )
```

return output length of hash function associated with a hash type

Parameters

hash_type	a TLS hash type

Returns

hash function output length

5.10.2.8 TLS_RANDOM_BYTE()

```
int TLS_RANDOM_BYTE ( )
```

get a random byte

Returns

a random byte

5.10.2.9 TLS_RANDOM_OCTAD()

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

get a random octad

Parameters

len	number of random bytes
R	octad to be filled with random bytes

5.10.2.10 TLS_HKDF_Extract()

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

HKDF Extract function.

Parameters

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

5.10.2.11 TLS_HKDF_Expand_Label()

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

Special HKDF Expand function (for TLS)

Parameters

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

5.10.2.12 TLS_HMAC()

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

simple HMAC function

Parameters

htype	hash algorithm
T	an output tag
K	an input key, or salt
M	an input message

5.10.2.13 TLS_HASH_NULL()

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

simple HASH of nothing function

Parameters

sha	the SHA2 function output length (32,48 or 64)
H	the output hash

5.10.2.14 Hash_Init()

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

Initiate Hashing context.

Parameters

hlen	length in bytes of SHA2 hashing output
h	a hashing context

5.10.2.15 Hash_Process()

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

Hash process a byte.

Parameters

h	a hashing context
b	the byte to be included in hash

5.10.2.16 Hash_Output()

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

Hash finish and output.

Parameters

h	a hashing context
d	the current output digest of an ongoing hashing operation

Returns

hash output length

5.10.2.17 AEAD_ENCRYPT()

```
void AEAD_ENCRYPT (
            crypto * send,
            int hdrlen,
            char * hdr,
            int ptlen,
            char * pt,
            octad * TAG )
```

AEAD encryption.

Parameters

send	the AES key and IV
hdrlen	the length of the header

Parameters

hdr	the header bytes
ptlen	the plaintext length
pt	the input plaintext and output ciphertext
TAG	the output authentication tag

5.10.2.18 AEAD_DECRYPT()

```
int AEAD_DECRYPT (
            crypto * recv,
            int hdrlen,
            char * hdr,
            int ctlen,
            char * ct,
            octad * TAG )
```

AEAD decryption.

Parameters

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

Returns

-1 if tag is wrong, else 0

5.10.2.19 GENERATE_KEY_PAIR()

```
void GENERATE_KEY_PAIR (
            int group,
            octad * SK,
            octad * PK )
```

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

Parameters

group	the cryptographic group used to generate the key pair
SK	the output Private Key
PK	the output Public Key

5.10.2.20 GENERATE_SHARED_SECRET()

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

generate a Diffie-Hellman shared secret

Parameters

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

5.10.2.21 CERT_SIGNATURE_VERIFY()

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

Verify a generic certificate signature.

Parameters

sigAlg	the signature type
CERT	the input certificate that was signed
SIG	the input signature
PUBKEY	the public key used to verify the signature

Returns

true if signature is valid, else false

5.10.2.22 TLS_SIGNATURE_VERIFY()

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

Verify a generic TLS transcript signature.

Parameters

sigAlg	the signature type
TRANS	the input transcript hash that was signed
SIG	the input signature
PUBKEY	the public key used to verify the signature

Returns

true if signature is valid, else false

5.10.2.23 TLS_SIGNATURE_SIGN()

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

Apply a generic TLS transcript signature.

Parameters

sigAlg	the signature type
KEY	the private key used to form the signature
TRANS	the input transcript hash to be signed
SIG	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 getByte (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

port	the TCP/IP port on which to connect
ip	the IP address with which to connect
toms	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

ip	the IP address
hostname	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

client	the socket connection to the Server
B	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

client	the socket connection to the Server
len	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

client	the socket connection to the Server
b	the received bytes
expected	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

client	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

client	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

client	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

client	the socket connection to the Server
B	the output octet
expected	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 init_ticket_context (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,
            unsign32 birth,
            ticket * T )
```

parse a received ticket octad into a ticket structure

Parameters

TICK	the input ticket octad
T	the output ticket structure
birth	the birth time of the ticket

Returns

bad ticket error, or 0 if ticket is good

5.12.2.3 init_ticket_context()

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

initialize a ticket structure, include time of creation

Parameters

T	the ticket structure
cipher_suite	the cipher suite currently in use
favourite_group	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

c	an X.509 private key
pk	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

c	an X.509 certificate
s	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

sc	a signed certificate
c	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

c	an X.509 certificate
k	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

c	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

c	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

c	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

c	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

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

c	an X.509 certificate
s	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

c	an X.509 certificate
s	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

c	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

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

c	an X.509 certificate
s	is a pointer to certificate extension SubjectAltNames
name	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