TII TLS1.3

0.1

Generated by Doxygen 1.9.1

Chapter 1

Description

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.

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.

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. 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
g++ -O2 client.cpp tls.a core.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
ip= 109.74.204.5
Private key= 0x0170a7e6c297fc8026ae8072c62596273bfa792879716e3d9f9c518384efae97
Client Public key= 0x402a2d7a1ca22eac2a3ab843c7a12a12343e85ce545c190a50fe8b5a1dc4ec15
Client to Server -> 16030100bf010000bb0303b39355382dcd121e82437b9e0f1072f0f3698fba2281672931b0a94be2653be72004
Client Hello sent
Server Random= cf21ad74e59a6111be1d8c021e65b891c2a211167abb8c5e079e09e2c8a8339c
Handshake Retry request!                                                    <--- Server does not s
Cipher suite= 1301
tls version= 0304
Key Share = 0017
Server HelloRetryRequest= 88 020000540303cf21ad74e59a6111be1d8c021e65b891c2a211167abb8c5e079e09e2c8a8339c20040
Client to Server -> 16030300e0010000dc0303a53891fa90aeac927e3686c11ffdcc643c56f116720d91e8af32b1cf245c538220cc
Server Random= 25b719a5c29b7d78cfc7bf9ec6e17cdb1cc8e30e7ec9c771c4db0fd8c7bb5a7e
Cipher suite= 1301
Key Share = 0017
Server Public Key= 04c00a53ca002cfed1d72c727800da9497d34ac44bee21543fed03425137c7929666ce15c3f74fef4a8de949fb9
```

```
tls version= 0304
Server Hello= 155 02000097030325b719a5c29b7d78cfc7bf9ec6e17cdb1cc8e30e7ec9c771c4db0fd8c7bb5a7e20ccf155d37563cf
SECP256R1 Key Exchange
Shared Secret= 8036b974c2bcf5130db585ad55c3f66aa83785c9775e22bdcdf3332d34ec1dc0
Handshake Secret= da45dee93ee938d89a2dddfbd40bfbc88cf909bd8641c6e508829f57794f2eae
Client handshake traffic secret= cba5cb38a4d397bae7213b195363f22ebec56ae0408e4a2b9df823f6c2760473
Server handshake traffic secret= 1131f1572fe474c7e636223a7a169d79c771f9bbb3c28a8a7f5bcfe3fc49ee19
Server fragment authenticates 19
12 padding bytes removed
Length of Encrypted Extension= 0
Server fragment authenticates 2570
12 padding bytes removed
1. Transcript Hash= b67be3f4f3afc7e0073d299e9cf45ad2fa81e3bad96cfde4d2096b3ec0072c4f

Server certificate
Signature is 4384d802a2bef1d6c17cce6d89996421015bc7dce12ee85dac5868ffc21d537f062fbb8e6fba1f4a5838ec3125f09414(
RSA signature of length 2048
Public key= 256 c31a51649a342aeccaf7dd9475d01a634fb482990d1326ff301ff1fdd0afff5c781355c01bd47899299b74c598be6c
RSA public key of length 2048
Issuer is  Let's Encrypt Authority X3
Subject is swifttls.org

st.curve= 2048
SIG= 256
4384d802a2bef1d6c17cce6d89996421015bc7dce12ee85dac5868ffc21d537f062fbb8e6fba1f4a5838ec3125f0941403baf420f3aa50

RSA PUBLIC KEY= 256
9cd30cf05ae52e47b7725d3783b3686330ead735261925e1bdbe35f170922fb7b84b4105aba99e350858ecb12ac468870ba3e375e4e6f3
Checking RSA Signature on Cert 32
RSA Signature/Verification succeeded
Intermediate Certificate Chain sig is OK

Intermediate Certificate
Signature is dd33d711f3635838dd1815fb0955be7656b97048a56947277bc2240892f15a1f4a1229372474511c6268b8cd957067e5f
RSA signature of length 2048
Public key= 256 9cd30cf05ae52e47b7725d3783b3686330ead735261925e1bdbe35f170922fb7b84b4105aba99e350858ecb12ac468
RSA public key of length 2048
Issuer is  DST Root CA X3
Subject is Let's Encrypt Authority X3


Public Key from root CA cert= dfafe99750088357b4cc6265f69082ecc7d32c6b30ca5becd9c37dc740c118148be0e83376492ae3
st.curve= 2048
SIG= 256
dd33d711f3635838dd1815fb0955be7656b97048a56947277bc2240892f15a1f4a1229372474511c6268b8cd957067e5f7a4bc4e2851cc

RSA PUBLIC KEY= 256
dfafe99750088357b4cc6265f69082ecc7d32c6b30ca5becd9c37dc740c118148be0e83376492ae33f214993ac4e0eaf3e48cb65eefcd3
Checking RSA Signature on Cert 32
RSA Signature/Verification succeeded
Root Certificate sig is OK!!!!
Certificate Chain is valid
Server fragment authenticates 277
12 padding bytes removed
2. Transcript Hash= 4066ae522e25e1f3390035572764a45c915bc91d9160c54ee207e074dbfecd2b
Signature Algorithm= 0804
Server Certificate Signature= 256 088f09c7f83826d0bbd5f22d101d5b7afb6dd0de4f055aad59339837bd59d0b8d25d9c028d94
Server Cert Verification OK
Server fragment authenticates 49
12 padding bytes removed
3. Transcript Hash= a0630db59fb4293004025289f57c5a962e84011fa37586d6a48fb382b4ac4ba5
Server Data is verified
Client Verify Data= b820d6c371057bfa7614be2b3e1a81715b224e399ba06e2d17e1b8299ca365d5
Client to Server -> 17030300358392348b3e61be337bcf591045e4bda4211e7b776fb5eca881b94eda44b4a07968a5ef1808d0c2e
Client application traffic secret= 60fc7bb14ccc33d606a17070310c36c7a12740c1402b174acfc82eac8c89ae42
Server application traffic secret= 1c5107e3696b9bf963b9536f8969d6975b06a7d14ac867ca441bc41b89c34af3
Sending Application Message

GET / HTTP/1.1
Host: swifttls.org

Client to Server -> 1703030037b9439dae723e8424b35a3bdcfec7eb591c91e13c93ec79857de56a7c0c23cce1cc0f03eb6dce3e5b
Waiting for Server input
```

```
Server fragment authenticates 71
12 padding bytes removed
Got a ticket
Waiting for Server input
Server fragment authenticates 439
12 padding bytes removed
Application data (truncated HTML) = 485454502f312e3120323030204f4b0d0a5365727665723a205377696674544c530d0a5374
Waiting for Server input
TIMEOUT
Connection closed

Connection re-opened - attempting resumption
Ticket= 00000e106744dbcb010000200abe1a546444a65a14c7830692fae3ebce299bc64867bd51267aff337ca92a870008002a000400

Ticket details
life time t= 60 minutes
Age obfuscator = 6744dbcb
Nonce = 00
Ticket = 32 0abe1a546444a65a14c7830692fae3ebce299bc64867bd51267aff337ca92a87
max_early_data = 262144

PSK= 5fb6ba90e449a650708808bbbab2c74ce98b38466f5ceade2e116b06d1d931dc
Binder Key= 32 829d4638e61c08b104d0b719c3facfc4fe6b86fb6a593a94428d770811ed4c2c
Early Secret= 1e25202ad4fd112ddcd1e26a111ed364750008dcf4212964c10e840d546221c4
Private key= 0x029c2115657b5a952f19832b3219c1c5a7a97837c05c25609de1396e36984a4e
Client Public key= 0x0438ecb199120283ec3b042a1ab73809f16c70ab1ce847de1820c0e9e3cb1a1c5a53db00dd1a437ad179658a3
Ticket age= 5114
obfuscated age = 6744efc5
Client to Server -> 160303011001000012f030390e07db6edc69224d4e994fd35a599c770f669d39e13a35535c7117db12a322b20a2
Truncated Client Hello sent
BND= 31a541997912f40cc9fe8f49025c502703c5b4d696157170131f1769d3eb6cce
Sending Binders
Client to Server -> 160303002300212031a541997912f40cc9fe8f49025c502703c5b4d696157170131f1769d3eb6cce
Client Early Traffic Secret= 97e8449a134b83da2b0bdb0a707e94863c77c1c5e5d875658dd93dbf66b49cc6
Sending some early data
Sending Application Message

GET / HTTP/1.1
Host: swifttls.org
Early-Data: 1

Client to Server -> 1703030046172444fd421667c98331214406aa92aab2aa0881b0ed3f6d6cd8efac39d934916becf0484583d6be
Server Random= 25b719ab8881259c48a8a3daa286c670af6136f8c8f349ee6e4921eaf24c7159
Cipher suite= 1301
Key Share = 0017
Server Public Key= 04dcada67f3bd198125448491ea968e62780f57b3958ac4b83e6660889ffb7a5f72471125886cef75e7a9bda115
tls version= 0304
PSK ID= 0
serverHello= 161 0 0200009d030325b719ab8881259c48a8a3daa286c670af6136f8c8f349ee6e4921eaf24c715920a22864e5d2dfe
SECP256R1 Key Exchange
Shared Secret= c21695de18de72e4fe594ee2181eba8154e0fff1b174923419689a0b9ca125bf
Handshake Secret= df4661dab16a7fc1196c36d5b5fe0401fd0172c155cf2d9792a323217a92063c
Client handshake traffic secret= 60be4bc5d462a698ac6ed5a945475be82031d7bebddb8d42f319097220206343
Server handshake traffic secret= 9d49d960ec385693dfdf4d88d3694860d7c5ea470fccc546ebb93723302cce70
Server fragment authenticates 23
12 padding bytes removed
Length of Encrypted Extension= 4
Early Data Accepted
2. Transcript Hash= d8233d93154ccedae45361a21acac29d6a244725da4cae02fc144dc0e0f2f07d
Server fragment authenticates 49
12 padding bytes removed
SR.len= 0
Send End of Early Data
Client to Server -> 1703030015d38f7638b857d65b4d9effa317871ae933287f46e6
3. Transcript Hash= 9045833c8871f10c44b57bcf910f06848a6096c0d239fb58b4b2f143654b2c75
Server Data is verified
Client Verify Data= 769d0ef3c6f22489319170b2c94aada4ae95aa34b2c88bb66c9ced47714c1d01
Client to Server -> 1703030035 04ea1fea838213f765597c5a571b099e4d48aeefdee74a412af4e8278825d74d08ac28a118d55168
Client application traffic secret= f62386365f4f06dc25d218b7aa86f6b58fc6346742c4bfd6ce108eaa9fcba925
Server application traffic secret= 42cd9298d5d384e1297ee33e12e04601ca6f8eb022d98ec23446d97bb0658243
Waiting for Server input
Server fragment authenticates 740
12 padding bytes removed
```

```
Application data (truncated HTML) = 485454502f312e3120323030204f4b0d0a5365727665723a205377696674544c530d0a5374
Waiting for Server input
Server fragment authenticates 71
12 padding bytes removed
Got a ticket
Waiting for Server input
TIMEOUT
```

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.

1.0.1 How to use it

1.0.1.1 Localhost 4433

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

```
./client
```

1.0.1.2 Just Host

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

1.0.1.3 Host and port

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

1.0.1.4 AF_UNIX Socket

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

Chapter 2

Data Structure Index

2.1 Data Structures

Here are the data structures with brief descriptions:

Chapter 3

File Index

3.1 File List

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

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]

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

4.1.2.6 sigAlgs

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

Supported signature algorithms

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]
- octet K
- octet IV
- unsign32 record

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

```
octet crypto::K
```

Key as octet

4.2.2.4 IV

```
octet crypto::IV
```

IV as octet

4.2.2.5 record

```
unsign32 crypto::record
```

current record number - to be incremented

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

- tls1_3.h

4.3 ret Struct Reference

function return structure

`#include <tls1_3.h>`

Data Fields

- unsign32 val
- int err

4.3.1 Detailed Description

function return structure

4.3.2 Field Documentation

4.3.2.1 val

`unsign32 ret::val`

return value

4.3.2.2 err

`int ret::err`

error return

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

- tls1_3.h

4.4 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** ()

4.4.1 Detailed Description

Socket instance.

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

- tls_sockets.h

4.5 ticket Struct Reference

ticket context structure

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

Data Fields

- char tick [TLS_MAX_TICKET_SIZE]
- char nonce [32]
- octet TICK
- octet NONCE
- int lifetime
- unsign32 age_obfuscator
- unsign32 max_early_data
- unsign32 birth

4.5.1 Detailed Description

ticket context structure

4.5.2 Field Documentation

4.5.2.1 tick

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

Ticket bytes

4.5.2.2 nonce

```
char ticket::nonce[32]
```

32-byte nonce

4.5.2.3 TICK

```
octet ticket::TICK
```

Ticket as octet

4.5.2.4 NONCE

```
octet ticket::NONCE
```

Nonce as octet

4.5.2.5 lifetime

```
int ticket::lifetime
```

ticket lifetime

4.5.2.6 age_obfuscator

```
unsign32 ticket::age_obfuscator
```

ticket age obfuscator

4.5.2.7 max_early_data

```
unsign32 ticket::max_early_data
```

Maximum early data allowed for this ticket

4.5.2.8 birth

```
unsign32 ticket::birth
```

Birth time of this ticket

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

- tls1_3.h

4.6 unihash Struct Reference

Universal Hash structure.

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

Data Fields

- hash256 sh32
- hash512 sh64
- int hlen

4.6.1 Detailed Description

Universal Hash structure.

4.6.2 Field Documentation

4.6.2.1 sh32

```
hash256 unihash::sh32
```

A SHA256 instance

4.6.2.2 sh64

```
hash512 unihash::sh64
```

A SHA384/512 instance

4.6.2.3 hlen

```
int unihash::hlen
```

The length of the SHA output in bytes (32/48/64)

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

- tls1_3.h

Chapter 5

File Documentation

5.1 tls1_3.h File Reference

Main TLS 1.3 Header File for constants and structures.

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

Data Structures

- struct ret
  function return structure
- struct crypto
  crypto context structure
- struct ticket
  ticket context structure
- struct capabilities
  Cryptographic capabilities of the client.
- struct unihash
  Universal Hash structure.

Macros

- #define IO_NONE 0
- #define IO_APPLICATION 1
- #define IO_PROTOCOL 2
- #define IO_DEBUG 3
- #define IO_WIRE 4
- #define POPULAR_ROOT_CERTS
- #define VERBOSITY IO_PROTOCOL
- #define THIS_YEAR 2021
- #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_TICKET_SIZE 512
- #define TLS_MAX_CLIENT_HELLO 256
- #define TLS_MAX_EXTENSIONS 512
- #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 12
- #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 RSA_PSS_RSAE_SHA256 0x0804
- #define RSA_PKCS1_SHA256 0x0401
- #define ECDSA_SECP384R1_SHA384 0x0503
- #define RSA_PSS_RSAE_SHA384 0x0805
- #define RSA_PKCS1_SHA384 0x0501
- #define RSA_PSS_RSAE_SHA512 0x0806
- #define RSA_PKCS1_SHA512 0x0601
- #define RSA_PKCS1_SHA1 0x0201
- #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 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 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_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 ILLEGAL_PARAMETER 0x2F
- #define UNEXPECTED_MESSAGE 0x0A
- #define DECRYPT_ERROR 0x33
- #define BAD_CERTIFICATE 0x2A
- #define UNSUPPORTED_EXTENSION 0x6E

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 POPULAR_ROOT_CERTS

```
#define POPULAR_ROOT_CERTS
```

Define this to limit root CAs to most popular only

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 TLS_MAX_HASH

```
#define TLS_MAX_HASH 64
```

Maximum hash output length in bytes

5.1.2.10 TLS_MAX_KEY

```
#define TLS_MAX_KEY 32
```

Maximum key length in bytes

5.1.2.11 TLS_X509_MAX_FIELD

```
#define TLS_X509_MAX_FIELD 256
```

Maximum X.509 field size

5.1.2.12 TLS_MAX_ROOT_CERT_SIZE

```
#define TLS_MAX_ROOT_CERT_SIZE 2048
```

I checked - current max for root CAs is 2016

5.1.2.13 TLS_MAX_ROOT_CERT_B64

```
#define TLS_MAX_ROOT_CERT_B64 2800
```

In base64 - current max for root CAs is 2688

5.1.2.14 TLS_MAX_TICKET_SIZE

```
#define TLS_MAX_TICKET_SIZE 512
```

maximum resumption ticket size

5.1.2.15 TLS_MAX_CLIENT_HELLO

```
#define TLS_MAX_CLIENT_HELLO 256
```

Max client hello size (less extensions)

5.1.2.16 TLS_MAX_EXTENSIONS

```
#define TLS_MAX_EXTENSIONS 512
```

Max extensions size

5.1.2.17 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

5.1.2.18 TLS_MAX_SIGNATURE_SIZE

```
#define TLS_MAX_SIGNATURE_SIZE 512
```

Max digital signature size in bytes

5.1.2.19 TLS_MAX_PUB_KEY_SIZE

```
#define TLS_MAX_PUB_KEY_SIZE 512
```

Max public key size in bytes

5.1.2.20 TLS_MAX_SECRET_KEY_SIZE

```
#define TLS_MAX_SECRET_KEY_SIZE 512
```

Max private key size in bytes

5.1.2.21 TLS_MAX_ECC_FIELD

```
#define TLS_MAX_ECC_FIELD 66
```

Max ECC field size in bytes

5.1.2.22 TLS_IV_SIZE

```
#define TLS_IV_SIZE 12
```

Max IV size in bytes

5.1.2.23 TLS_TAG_SIZE

```
#define TLS_TAG_SIZE 16
```

Max HMAC tag length in bytes

5.1.2.24 TLS_MAX_COOKIE

```
#define TLS_MAX_COOKIE 128
```

Max Cookie size

5.1.2.25 TLS_MAX_SERVER_NAME

```
#define TLS_MAX_SERVER_NAME 128
```

Max server name size in bytes

5.1.2.26 TLS_MAX_SUPPORTED_GROUPS

```
#define TLS_MAX_SUPPORTED_GROUPS 5
```

Max number of supported crypto groups

5.1.2.27 TLS_MAX_SUPPORTED_SIGS

```
#define TLS_MAX_SUPPORTED_SIGS 12
```

Max number of supported signature schemes

5.1.2.28 TLS_MAX_PSK_MODES

```
#define TLS_MAX_PSK_MODES 2
```

Max preshared key modes

5.1.2.29 TLS_MAX_CIPHER_SUITES

```
#define TLS_MAX_CIPHER_SUITES 5
```

Max number of supported cipher suites

5.1.2.30 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

5.1.2.31 TLS_AES_256_GCM_SHA384

```
#define TLS_AES_256_GCM_SHA384 0x1302
```

AES256/SHA384/GCM cipher suite

5.1.2.32 TLS_CHACHA20_POLY1305_SHA256

```
#define TLS_CHACHA20_POLY1305_SHA256 0x1303
```

CHACHA20/SHA256/POLY1305 cipher suite

5.1.2.33 X25519

```
#define X25519 0x001d
```

X25519 elliptic curve key exchange

5.1.2.34 SECP256R1

```
#define SECP256R1 0x0017
```

NIST SECP256R1 elliptic curve key exchange

5.1.2.35 SECP384R1

```
#define SECP384R1 0x0018
```

NIST SECP384R1 elliptic curve key exchange

5.1.2.36 ECDSA_SECP256R1_SHA256

```
#define ECDSA_SECP256R1_SHA256 0x0403
```

Supported ECDSA Signature algorithm

5.1.2.37 RSA_PSS_RSAE_SHA256

```
#define RSA_PSS_RSAE_SHA256 0x0804
```

Supported RSA Signature algorithm

5.1.2.38 RSA_PKCS1_SHA256

```
#define RSA_PKCS1_SHA256 0x0401
```

Supported RSA Signature algorithm

5.1.2.39 ECDSA_SECP384R1_SHA384

```
#define ECDSA_SECP384R1_SHA384 0x0503
```

Supported ECDSA Signature algorithm

5.1.2.40 RSA_PSS_RSAE_SHA384

```
#define RSA_PSS_RSAE_SHA384 0x0805
```

Supported RSA Signature algorithm

5.1.2.41 RSA_PKCS1_SHA384

```
#define RSA_PKCS1_SHA384 0x0501
```

Supported RSA Signature algorithm

5.1.2.42 RSA_PSS_RSAE_SHA512

```
#define RSA_PSS_RSAE_SHA512 0x0806
```

Supported RSA Signature algorithm

5.1.2.43 RSA_PKCS1_SHA512

```
#define RSA_PKCS1_SHA512 0x0601
```

Supported RSA Signature algorithm

5.1.2.44 RSA_PKCS1_SHA1

```
#define RSA_PKCS1_SHA1 0x0201
```

Supported (but deprecated!) RSA Signature algorithm

5.1.2.45 PSKOK

```
#define PSKOK 0x00
```

Preshared Key only mode

5.1.2.46 PSKWECDHE

```
#define PSKWECDHE 0x01
```

Preshared Key with Diffie-Hellman key exchange mode

5.1.2.47 TLS1_0

```
#define TLS1_0 0x0301
```

TLS 1.0 version

5.1.2.48 TLS1_2

```
#define TLS1_2 0x0303
```

TLS 1.2 version

5.1.2.49 TLS1_3

```
#define TLS1_3 0x0304
```

TLS 1.3 version

5.1.2.50 SERVER_NAME

```
#define SERVER_NAME 0x0000
```

Server Name extension

5.1.2.51 SUPPORTED_GROUPS

```
#define SUPPORTED_GROUPS 0x000a
```

Supported Group extension

5.1.2.52 SIG_ALGS

```
#define SIG_ALGS 0x000d
```

Signature algorithms extension

5.1.2.53 KEY_SHARE

```
#define KEY_SHARE 0x0033
```

Key Share extension

5.1.2.54 PSK_MODE

```
#define PSK_MODE 0x002d
```

Preshared key mode extension

5.1.2.55 PRESHARED_KEY

```
#define PRESHARED_KEY 0x0029
```

Preshared key extension

5.1.2.56 TLS_VER

```
#define TLS_VER 0x002b
```

TLS version extension

5.1.2.57 COOKIE

```
#define COOKIE 0x002c
```

Cookie extension

5.1.2.58 EARLY_DATA

```
#define EARLY_DATA 0x002a
```

Early Data extension

5.1.2.59 MAX_FRAG_LENGTH

```
#define MAX_FRAG_LENGTH 0x0001
```

max fragmentation length extension

5.1.2.60 PADDING

```
#define PADDING 0x0015
```

Padding extension

5.1.2.61 HSHAKE

```
#define HSHAKE 0x16
```

Handshake record

5.1.2.62 APPLICATION

```
#define APPLICATION 0x17
```

Application record

5.1.2.63 ALERT

```
#define ALERT 0x15
```

Alert record

5.1.2.64 CHANGE_CIPHER

```
#define CHANGE_CIPHER 0x14
```

Change Cipher record

5.1.2.65 TIME_OUT

```
#define TIME_OUT 0x01
```

Time-out

5.1.2.66 HANDSHAKE_RETRY

```
#define HANDSHAKE_RETRY 0x02
```

Handshake retry

5.1.2.67 STRANGE_EXTENSION

```
#define STRANGE_EXTENSION 0x03
```

Strange extension

5.1.2.68 CLIENT_HELLO

```
#define CLIENT_HELLO 0x01
```

Client Hello message

5.1.2.69 SERVER_HELLO

```
#define SERVER_HELLO 0x02
```

Server Hello message

5.1.2.70 CERTIFICATE

```
#define CERTIFICATE 0x0b
```

Certificate message

5.1.2.71 CERT_VERIFY

```
#define CERT_VERIFY 0x0f
```

Certificate Verify message

5.1.2.72 FINISHED

```
#define FINISHED 0x14
```

Handshae Finished message

5.1.2.73 ENCRYPTED_EXTENSIONS

```
#define ENCRYPTED_EXTENSIONS 0x08
```

Encrypted Extensions message

5.1.2.74 TICKET

```
#define TICKET 0x04
```

Ticket message

5.1.2.75 KEY_UPDATE

```
#define KEY_UPDATE 0x18
```

Key Update message

5.1.2.76 MESSAGE_HASH

```
#define MESSAGE_HASH 0xFE
```

Special synthetic message hash message

5.1.2.77 END_OF_EARLY_DATA

```
#define END_OF_EARLY_DATA 0x05
```

End of Early Data message

5.1.2.78 NOT_TLS1_3

```
#define NOT_TLS1_3 -2
```

Wrong version error, not TLS1.3

5.1.2.79 BAD_CERT_CHAIN

```
#define BAD_CERT_CHAIN -3
```

Bad Certificate Chain error

5.1.2.80 ID_MISMATCH

```
#define ID_MISMATCH -4
```

Session ID mismatch error

5.1.2.81 UNRECOGNIZED_EXT

```
#define UNRECOGNIZED_EXT -5
```

Unrecognised extension error

5.1.2.82 BAD_HELLO

```
#define BAD_HELLO -6
```

badly formed Hello message error

5.1.2.83 WRONG_MESSAGE

```
#define WRONG_MESSAGE -7
```

Message out-of-order error

5.1.2.84 MISSING_REQUEST_CONTEXT

```
#define MISSING_REQUEST_CONTEXT -8
```

Request context missing error

5.1.2.85 AUTHENTICATION_FAILURE

```
#define AUTHENTICATION_FAILURE -9
```

Authentication error - AEAD Tag incorrect

5.1.2.86 BAD_RECORD

```
#define BAD_RECORD -10
```

Badly formed Record received

5.1.2.87 BAD_TICKET

```
#define BAD_TICKET -11
```

Badly formed Ticket received

5.1.2.88 ILLEGAL_PARAMETER

```
#define ILLEGAL_PARAMETER 0x2F
```

Illegal parameter alert from Server

5.1.2.89 UNEXPECTED_MESSAGE

```
#define UNEXPECTED_MESSAGE 0x0A
```

Unexpected message alert from Server

5.1.2.90 DECRYPT_ERROR

```
#define DECRYPT_ERROR 0x33
```

Decryption error alert from Server

5.1.2.91 BAD_CERTIFICATE

```
#define BAD_CERTIFICATE 0x2A
```

Bad certificate alert from Server

5.1.2.92 UNSUPPORTED_EXTENSION

```
#define UNSUPPORTED_EXTENSION 0x6E
```

Unsupported extension alert from Server

5.2 tls_cacerts.h File Reference

Certificate Authority root certificate store.

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

Variables

- 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 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 "core.h"
#include "x509.h"
#include "ecdh_NIST256.h"
#include "ecdh_NIST384.h"
#include "rsa_RSA2048.h"
#include "rsa_RSA4096.h"
```

Functions

- bool CHECK_CERT_CHAIN (octet *CERTCHAIN, char *hostname, octet *PUBKEY)

  Check Certificate Chain.
- bool IS_SERVER_CERT_VERIFY (int sigalg, octet *SCVSIG, octet *H, octet *CERTPK)

  verify Server's signature on protocol transcript

5.3.1 Detailed Description

Process Certificate Chain.

Author

Mike Scott

5.3.2 Function Documentation

5.3.2.1 CHECK_CERT_CHAIN()

```
bool CHECK_CERT_CHAIN (
            octet * CERTCHAIN,
            char * hostname,
            octet * 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

true if certificate chain is OK, else returns false

5.3.2.2 IS_SERVER_CERT_VERIFY()

```
bool IS_SERVER_CERT_VERIFY (
            int sigalg,
            octet * SCVSIG,
            octet * H,
            octet * 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

5.4 tls_client_recv.h File Reference

Process Input received from the Server.

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

Functions

- ret parseOctet (octet *E, int len, octet *M, int &ptr)

  Parse out an Octet from a pointer into an Octet.

- ret parseInt16 (octet *M, int &ptr)

  Parse out a 16-bit unsigned integer from a pointer into an Octet.
- ret parseInt24 (octet *M, int &ptr)

  Parse out a 24-bit unsigned integer from a pointer into an Octet.
- ret parseInt32 (octet *M, int &ptr)

  Parse out a 32-bit unsigned integer from a pointer into an Octet.
- ret parseByte (octet *M, int &ptr)

  Parse out an unsigned byte from a pointer into an Octet.
- ret parseOctetptr (octet *E, int len, octet *M, int &ptr)

  Return a pointer to an Octet from a pointer into an Octet.
- int getServerFragment (Socket &client, crypto *recv, octet *IO)

  Read a record from the Server, a fragment of a full protocol message.
- ret parseByteorPull (Socket &client, octet *IO, int &ptr, crypto *recv)

  Parse out an unsigned byte from a pointer into an Octet, if necessary pulling in a new fragment.
- ret parseInt32orPull (Socket &client, octet *IO, int &ptr, crypto *recv)

  Parse out a 32-bit unsigned integer from a pointer into an Octet, if necessary pulling in a new fragment.
- ret parseInt24orPull (Socket &client, octet *IO, int &ptr, crypto *recv)

  Parse out a 24-bit unsigned integer from a pointer into an Octet, if necessary pulling in a new fragment.
- ret parseInt16orPull (Socket &client, octet *IO, int &ptr, crypto *recv)

  Parse out a 16-bit unsigned integer from a pointer into an Octet, if necessary pulling in a new fragment.
- ret parseOctetorPull (Socket &client, octet *O, int len, octet *IO, int &ptr, crypto *recv)

  Parse out an octet from a pointer into an Octet, if necessary pulling in a new fragment.
- ret parseOctetorPullptr (Socket &client, octet *O, int len, octet *IO, int &ptr, crypto *recv)

  Return a pointer to an Octet from a pointer into an Octet, if necessary pulling in a new fragment.
- int getServerEncryptedExtensions (Socket &client, octet *IO, crypto *recv, unihash *trans_hash, bool &early_data_accepted)

  Receive and parse Server Encrypted Extensions.
- int getServerCertVerify (Socket &client, octet *IO, crypto *recv, unihash *trans_hash, octet *SCVSIG, int &sigalg)

  Get Server proof that he owns the Certificate, by receiving and verifying its signature on transcript hash.
- int getServerFinished (Socket &client, octet *IO, crypto *recv, unihash *trans_hash, octet *HFIN)

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

  Receive and parse initial Server Hello.
- int getCheckServerCertificateChain (Socket &client, octet *IO, crypto *recv, unihash *trans_hash, char *hostname, octet *PUBKEY)

  Receive and check certificate chain.

5.4.1 Detailed Description

Process Input received from the Server.

Author

Mike Scott

5.4.2 Function Documentation

5.4.2.1 parseOctet()

```
ret parseOctet (
            octet * E,
            int len,
            octet * M,
            int & ptr )
```

Parse out an Octet from a pointer into an Octet.

Parameters

E	the output octet copied out from the octet M
len	the expected length of the output octet E
M	the input octet
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 (
            octet * M,
            int & ptr )
```

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

Parameters

M	the input octet
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 (
            octet * M,
            int & ptr )
```

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

Parameters

M	the input octet
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 (
            octet * M,
            int & ptr )
```

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

Parameters

M	the input octet
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 (
            octet * M,
            int & ptr )
```

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

Parameters

M	the input octet
ptr	a pointer into M, which advances after use

Returns

the unsigned byte, and an error flag

5.4.2.6 parseOctetptr()

```
ret parseOctetptr (
            octet * E,
            int len,
            octet * M,
            int & ptr )
```

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

Parameters

E	a pointer to an octet contained within an octet M
len	the expected length of the octet E
M	the input octet
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,
            octet * 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,
```

```
octet * IO,
int & ptr,
crypto * recv )
```

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

Parameters

client	the socket connection to the Server
IO	the input octet
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,
            octet * IO,
            int & ptr,
            crypto * recv )
```

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

Parameters

client	the socket connection to the Server
IO	the input octet
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,
            octet * IO,
            int & ptr,
            crypto * recv )
```

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

Parameters

client	the socket connection to the Server
IO	the input octet
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,
            octet * IO,
            int & ptr,
            crypto * recv )
```

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

Parameters

client	the socket connection to the Server
IO	the input octet
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 parseOctetorPull()

```
ret parseOctetorPull (
            Socket & client,
            octet * O,
            int len,
            octet * IO,
            int & ptr,
            crypto * recv )
```

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

Parameters

client	the socket connection to the Server
O	the output octet
len	the expected length of the output octet O
IO	the input octet
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 parseOctetorPullptr()

```
ret parseOctetorPullptr (
            Socket & client,
            octet * O,
            int len,
            octet * IO,
            int & ptr,
            crypto * recv )
```

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

Parameters

client	the socket connection to the Server
O	a pointer to an octet contained within an octet IO
len	the expected length of the octet O
IO	the input octet
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 getServerEncryptedExtensions()

```
int getServerEncryptedExtensions (
            Socket & client,
            octet * IO,
            crypto * recv,
            unihash * trans_hash,
            bool & early_data_accepted )
```

Receive and parse Server Encrypted Extensions.

Parameters

client	the socket connection to the Server
IO	an Octet to accept input
recv	the cryptographic key under which the extensions are encrypted
trans_hash	the current and updated transcript hash
early_data_accepted	an output boolean indicating if early data was accepted

Returns

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

5.4.2.15 getServerCertVerify()

```
int getServerCertVerify (
            Socket & client,
            octet * IO,
            crypto * recv,
            unihash * trans_hash,
            octet * 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 Octet 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

5.4.2.16 getServerFinished()

```
int getServerFinished (
            Socket & client,
            octet * IO,
            crypto * recv,
            unihash * trans_hash,
            octet * HFIN )
```

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

Parameters

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

Returns

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

5.4.2.17 getServerHello()

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

Receive and parse initial Server Hello.

Parameters

client	the socket connection to the Server
SH	an Octet 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

5.4.2.18 getCheckServerCertificateChain()

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

Receive and check certificate chain.

Parameters

client	the socket connection to the Server
IO	an Octet 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

5.5 tls_client_send.h File Reference

Process Output to be sent to the Server.

```
#include "core.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 (octet *EXT, unsign32 age, octet *IDS, int sha)

  Add PreShared Key extension to under-construction Extensions Octet (omitting binder)
- void addServerNameExt (octet *EXT, char *servername)

  Add Server name extension to under-construction Extensions Octet.
- void addSupportedGroupsExt (octet *EXT, int nsg, int *supportedGroups)

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

  Add Supported Signature algorithms to under-construction Extensions Octet.
- void addKeyShareExt (octet *EXT, int alg, octet *PK)

  Add Key Share extension to under-construction Extensions Octet.
- void addMFLExt (octet *EXT, int mode)

  Add Maximum Fragment Length extension to under-construction Extensions Octet.
- void addPSKExt (octet *EXT, int mode)

  Add Preshared Key exchange modes extension to under-construction Extensions Octet.
- void addVersionExt (octet *EXT, int version)

  Add Version extension to under-construction Extensions Octet.
- void addPadding (octet *EXT, int n)

  Add padding extension to under-construction Extensions Octet.
- void addCookieExt (octet *EXT, octet *CK)

  Add Cookie extension to under-construction Extensions Octet.
- void addEarlyDataExt (octet *EXT)

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

- int clientRandom (octet *RN, csprng *RNG)

  Generate 32-byte random octet.
- int sessionID (octet *SI, csprng *RNG)

  Create 32-byte random session ID octet.
- int cipherSuites (octet *CS, int ncs, int *ciphers)

  Build a cipher-suites octet from supported ciphers.
- void sendClientMessage (Socket &client, csprng *RNG, int rectype, int version, crypto *send, octet *CM, octet *EXT, octet *IO)

  Send a generic client message (as a single record) to the Server.
- void sendBinder (Socket &client, csprng *RNG, octet *B, octet *BND, octet *IO)

  Send a preshared key binder message to the Server.
- void sendClientHello (Socket &client, csprng *RNG, int version, octet *CH, int nsc, int *ciphers, octet *CID, octet *EXTENSIONS, int extra, octet *IO)

  Prepare and send Client Hello message to the Server, appending prepared extensions.
- void sendClientAlert (Socket &client, csprng *RNG, int type, crypto *send, octet *IO)

  Prepare and send an Alert message to the Server.
- void sendClientVerify (Socket &client, csprng *RNG, crypto *send, unihash *h, octet *CHF, octet *IO)

  Prepare and send a final handshake Verification message to the Server.
- void sendEndOfEarlyData (Socket &client, csprng *RNG, crypto *send, unihash *h, octet *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 (
            octet * EXT,
            unsign32 age,
            octet * IDS,
            int sha )
```

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

Parameters

EXT	the extensions octet 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 (
            octet * EXT,
            char * servername )
```

Add Server name extension to under-construction Extensions Octet.

Parameters

EXT	the extensions octet which is being built
servername	the Host name (URL) of the Server

5.5.2.4 addSupportedGroupsExt()

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

Add Supported Groups extension to under-construction Extensions Octet.

Parameters

EXT	the extensions octet which is being built
nsg	Number of supported groups
supportedGroups	an array of supported groups

5.5.2.5 addSigAlgsExt()

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

Add Supported Signature algorithms to under-construction Extensions Octet.

Parameters

EXT	the extensions octet which is being built
nsa	Number of supported signature algorithms
sigAlgs	an array of supported signature algorithms

5.5.2.6 addKeyShareExt()

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

Add Key Share extension to under-construction Extensions Octet.

Parameters

EXT	the extensions octet which is being built
alg	the suggested key exchange algorithm
PK	the key exchange public value to be sent to the Server

5.5.2.7 addMFLExt()

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

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

Parameters

EXT	the extensions octet which is being built
mode	the proposed maximum fragment size

5.5.2.8 addPSKExt()

```
void addPSKExt (
            octet * EXT,
            int mode )
```

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

Parameters

EXT	the extensions octet which is being built
mode	the proposed preshared key mode

5.5.2.9 addVersionExt()

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

Add Version extension to under-construction Extensions Octet.

Parameters

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

5.5.2.10 addPadding()

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

Add padding extension to under-construction Extensions Octet.

Parameters

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

5.5.2.11 addCookieExt()

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

Add Cookie extension to under-construction Extensions Octet.

Parameters

EXT	the extensions octet which is being built
CK	the cookie octet to be added

5.5.2.12 addEarlyDataExt()

```
void addEarlyDataExt (
             octet * EXT )
```

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

Parameters

EXT	the extensions octet which is being built

5.5.2.13 clientRandom()

```
int clientRandom (
             octet * RN,
             csprng * RNG )
```

Generate 32-byte random octet.

Parameters

RN	the output 32-byte octet
RNG	a random number generator

Returns

length of output octet

5.5.2.14 sessionID()

```
int sessionID (
            octet * SI,
            csprng * RNG )
```

Create 32-byte random session ID octet.

Parameters

SI	the output random octet
RNG	a random number generator

Returns

length of output octet

5.5.2.15 cipherSuites()

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

Build a cipher-suites octet from supported ciphers.

Parameters

CS	the output cipher-suite octet
ncs	the number of supported cipher-suites
ciphers	an array of supported cipher-suites

Returns

length of the output octet

5.5.2.16 sendClientMessage()

```
void sendClientMessage (
            Socket & client,
            csprng * RNG,
            int rectype,
            int version,
            crypto * send,
            octet * CM,
```

```
            octet * EXT,
            octet * IO )
```

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

Parameters

client	the socket connection to the Server
RNG	a random number generator
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 octet in which to construct the encrypted message

5.5.2.17 sendBinder()

```
void sendBinder (
            Socket & client,
            csprng * RNG,
            octet * B,
            octet * BND,
            octet * IO )
```

Send a preshared key binder message to the Server.

Parameters

client	the socket connection to the Server
RNG	a random number generator
B	workspace octet in which to construct binder message
BND	binding HMAC of truncated transcript hash
IO	the workspace octet in which to construct the overall message

5.5.2.18 sendClientHello()

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

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

Parameters

client	the socket connection to the Server
RNG	a random number generator
version	TLS version indication
CH	workspace octet 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 octet in which to construct the overall message

5.5.2.19 sendClientAlert()

```
void sendClientAlert (
            Socket & client,
            csprng * RNG,
            int type,
            crypto * send,
            octet * IO )
```

Prepare and send an Alert message to the Server.

Parameters

client	the socket connection to the Server
RNG	a random number generator
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 octet in which to construct the overall message

5.5.2.20 sendClientVerify()

```
void sendClientVerify (
            Socket & client,
            csprng * RNG,
            crypto * send,
            unihash * h,
            octet * CHF,
            octet * IO )
```

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

Parameters

client	the socket connection to the Server
RNG	a random number generator
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 octet in which to construct the overall message

5.5.2.21 sendEndOfEarlyData()

```
void sendEndOfEarlyData (
            Socket & client,
            csprng * RNG,
            crypto * send,
            unihash * h,
            octet * IO )
```

Indicate End of Early Data in message to the Server.

Parameters

client	the socket connection to the Server
RNG	a random number generator
send	the cryptographic key under which the message is encrypted
h	the current transcript hash up to this point
IO	the workspace octet in which to construct the overall message

5.5.2.22 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 "core.h"
#include "tls1_3.h"
#include "ecdh_NIST256.h"
#include "ecdh_NIST384.h"
#include "ecdh_C25519.h"
```

Functions

- void Hash_Init (int hlen, unihash *h)

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

  Hash process a byte.
- void Hash_Output (unihash *h, char *d)

  Hash output.
- void running_hash (octet *O, unihash *h)

  Accumulate octet into ongoing hashing.
- void transcript_hash (unihash *h, octet *O)

  Output current hash value.
- void running_syn_hash (octet *O, octet *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, octet *K, octet *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, octet *TS, crypto *context)

  Build a crypto context from an input raw Secret.
- void RECOVER_PSK (int sha, octet *RMS, octet *NONCE, octet *PSK)

  Recover a pre-shared key from Resumption Master Secret and a nonce.
- void GET_EARLY_SECRET (int sha, octet *PSK, octet *ES, octet *BKE, octet *BKR)

  Extract Early Secret Key and Binder Key from Preshared Key (External or Resumption)
- void GET_LATER_SECRETS (int sha, octet *H, octet *ES, octet *CETS, octet *EEMS)

  Extract more secrets from Early Secret.
- void GET_HANDSHAKE_SECRETS (int sha, octet *SS, octet *ES, octet *H, octet *HS, octet *CHTS, octet *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 sha, octet *HS, octet *SFH, octet *CFH, octet *CTS, octet *STS, octet *EMS, octet *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, octet *TS)

  Perform a Key Update on a crypto context.
- bool IS_VERIFY_DATA (int sha, octet *SF, octet *STS, octet *H)

  Test if data from Server is verified using server traffic secret and a transcript hash.
- void VERIFY_DATA (int sha, octet *SF, octet *CTS, octet *H)

  Create handshake verification data for Client to send to Server from client traffic secret and a transcript hash.
- void GENERATE_KEY_PAIR (csprng *RNG, int group, octet *SK, octet *PK)

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

  generate a Diffie-Hellman shared secret

5.6.1 Detailed Description

TLS 1.3 crypto support functions.

Author

Mike Scott

5.6.2 Function Documentation

5.6.2.1 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

5.6.2.2 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

5.6.2.3 Hash_Output()

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

Hash output.

Parameters

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

5.6.2.4 running_hash()

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

Accumulate octet into ongoing hashing.

Parameters

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

5.6.2.5 transcript_hash()

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

Output current hash value.

Parameters

h	a hashing context
O	an output octet containing current hash

5.6.2.6 running_syn_hash()

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

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

Parameters

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

5.6.2.7 init_crypto_context()

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

Initiate a Crypto Context.

Parameters

C	an AEAD encryption context

5.6.2.8 create_crypto_context()

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

Build a Crypto Context.

Parameters

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

5.6.2.9 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

5.6.2.10 GET_KEY_AND_IV()

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

Build a crypto context from an input raw Secret.

Parameters

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

5.6.2.11 RECOVER_PSK()

```
void RECOVER_PSK (
            int sha,
            octet * RMS,
            octet * NONCE,
            octet * PSK )
```

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

Parameters

sha	length in bytes of SHA2 hashing output
RMS	the input resumption master secret
NONCE	the input nonce
PSK	the output pre-shared key

5.6.2.12 GET_EARLY_SECRET()

```
void GET_EARLY_SECRET (
            int sha,
            octet * PSK,
            octet * ES,
            octet * BKE,
            octet * BKR )
```

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

Parameters

sha	length in bytes of SHA2 hashing output
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)

5.6.2.13 GET_LATER_SECRETS()

```
void GET_LATER_SECRETS (
            int sha,
            octet * H,
            octet * ES,
            octet * CETS,
            octet * EEMS )
```

Extract more secrets from Early Secret.

Parameters

sha	length in bytes of SHA2 hashing output
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)

5.6.2.14 GET_HANDSHAKE_SECRETS()

```
void GET_HANDSHAKE_SECRETS (
            int sha,
            octet * SS,
            octet * ES,
            octet * H,
```

```
            octet * HS,
            octet * CHTS,
            octet * SHTS )
```

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

Parameters

sha	length in bytes of SHA2 hashing output
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

5.6.2.15 GET_APPLICATION_SECRETS()

```
void GET_APPLICATION_SECRETS (
            int sha,
            octet * HS,
            octet * SFH,
            octet * CFH,
            octet * CTS,
            octet * STS,
            octet * EMS,
            octet * RMS )
```

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

Parameters

sha	length in bytes of SHA2 hashing output
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)

5.6.2.16 UPDATE_KEYS()

```
void UPDATE_KEYS (
```

```
            crypto * context,
            octet * TS )
```

Perform a Key Update on a crypto context.

Parameters

context	an AEAD encryption context
TS	the updated Traffic secret

5.6.2.17 IS_VERIFY_DATA()

```
bool IS_VERIFY_DATA (
            int sha,
            octet * SF,
            octet * STS,
            octet * H )
```

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

Parameters

sha	length in bytes of SHA2 hashing output
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

5.6.2.18 VERIFY_DATA()

```
void VERIFY_DATA (
            int sha,
            octet * SF,
            octet * CTS,
            octet * H )
```

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

Parameters

sha	length in bytes of SHA2 hashing output
SF	the output verification data
CTS	the input Client Traffic Secret
H	the input partial transcript hash

5.6.2.19 GENERATE_KEY_PAIR()

```
void GENERATE_KEY_PAIR (
            csprng * RNG,
            int group,
            octet * SK,
            octet * PK )
```

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

Parameters

RNG	a random number generator
group	the cryptographic group used to generate the key pair
SK	the output Private Key
PK	the output Public Key

5.6.2.20 GENERATE_SHARED_SECRET()

```
void GENERATE_SHARED_SECRET (
            int group,
            octet * SK,
            octet * PK,
            octet * 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

5.7 tls_logger.h File Reference

TLS 1.3 logging.

```
#include <string.h>
#include "tls1_3.h"
#include "x509.h"
```

Functions

- void myprintf (char *s)

  internal printf function - all output funnels through this function
- void logger (char *preamble, char *string, unsign32 info, octet *O)

  basic logging function
- void logServerHello (int cipher_suite, int kex, int pskid, octet *PK, octet *CK)

  logging the Server hello
- void logTicket (int lifetime, unsign32 age_obfuscator, unsign32 max_early_data, octet *NONCE, octet *ETICK)

  logging a resumption ticket
- void logCert (octet *CERT)

  logging a Certificate in standard base 64 format
- void logCertDetails (char *txt, octet *PUBKEY, pktype pk, octet *SIG, pktype sg, octet *ISSUER, octet *SUBJECT)

  logging Certificate details
- void logServerResponse (int rtn, octet *O)

  log the result of client processing of a Server response

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,
            octet * 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 Octet to be output (or NULL)

5.7.2.3 logServerHello()

```
void logServerHello (
            int cipher_suite,
            int kex,
            int pskid,
            octet * PK,
            octet * 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,
            octet * NONCE,
            octet * ETICK )
```

logging a resumption ticket

Parameters

lifetime	the ticket lifetime in minutes

Parameters

age_obfuscator	the ticket age obfuscator
max_early_data	the maximum amount of permitted early data
NONCE	the Ticket nonce
ETICK	the Ticket octet

5.7.2.5 logCert()

```
void logCert (
             octet * CERT )
```

logging a Certificate in standard base 64 format

Parameters

CERT	the certificate to be logged

5.7.2.6 logCertDetails()

```
void logCertDetails (
             char * txt,
             octet * PUBKEY,
             pktype pk,
             octet * SIG,
             pktype sg,
             octet * ISSUER,
             octet * SUBJECT )
```

logging Certificate details

Parameters

txt	preamble text
PUBKEY	the certificate public key octet
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

5.7.2.7 logServerResponse()

```
void logServerResponse (
            int rtn,
            octet * 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

5.8 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, csprng &RNG, int &favourite_group, capabilities &CPB, octet &IO, octet &RMS, ticket &T, crypto &K_send, crypto &K_recv, octet &STS)

  TLS 1.3 full handshake.
- int TLS13_resume (Socket &client, char *hostname, csprng &RNG, int favourite_group, capabilities &CPB, octet &IO, octet &RMS, ticket &T, crypto &K_send, crypto &K_recv, octet &STS, octet &EARLY)

  TLS 1.3 resumption handshake.

5.8.1 Detailed Description

TLS 1.3 main client-side protocol functions.

Author

Mike Scott

5.8.2 Function Documentation

5.8.2.1 TLS13_full()

```
int TLS13_full (
             Socket & client,
             char * hostname,
             csprng & RNG,
             int & favourite_group,
             capabilities & CPB,
             octet & IO,
             octet & RMS,
             ticket & T,
             crypto & K_send,
             crypto & K_recv,
             octet & STS )
```

TLS 1.3 full handshake.

Parameters

client	the socket connection to the Server
hostname	the host name (URL) of the server
RNG	a random number generator
favourite_group	our preferred group, which may be updated on a handshake retry
CPB	the client capabilities structure
IO	a workspace octet to buffer Server input
RMS	a returned Resumption Master secret
T	a returned resumption ticket
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

5.8.2.2 TLS13_resume()

```
int TLS13_resume (
             Socket & client,
             char * hostname,
             csprng & RNG,
             int favourite_group,
             capabilities & CPB,
             octet & IO,
             octet & RMS,
             ticket & T,
             crypto & K_send,
             crypto & K_recv,
             octet & STS,
             octet & EARLY )
```

TLS 1.3 resumption handshake.

Parameters

client	the socket connection to the Server
hostname	the host name (URL) of the server
RNG	a random number generator
favourite_group	our preferred group
CPB	the client capabilities structure
IO	a workspace octet to buffer Server input
RMS	a returned Resumption Master secret
T	a returned resumption ticket
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
EARLY	early data that can be immediately sent to the server (0-RTT data)

5.9 tls_sockets.h File Reference

set up sockets for reading and writing

```
#include <string.h>
#include "core.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 sendOctet (Socket &client, octet *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 getOctet (Socket &client, octet *B, int expected)
  receive an octet from a socket

5.9.1 Detailed Description

set up sockets for reading and writing

Author

Mike Scott

5.9.2 Function Documentation

5.9.2.1 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

5.9.2.2 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

5.9.2.3 sendOctet()

```
void sendOctet (
            Socket & client,
            octet * B )
```

send an octet over a socket

Parameters

client	the socket connection to the Server
B	the octet to be transmitted

5.9.2.4 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

5.9.2.5 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

5.9.2.6 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

5.9.2.7 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

5.9.2.8 getByte()

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

receive a single byte from a socket

Parameters

client	the socket connection to the Server

Returns

a byte

5.9.2.9 getOctet()

```
int getOctet (
            Socket & client,
            octet * 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

5.10 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 (octet *TICK, ticket *T)

  parse a received ticket octet into a ticket structure
- void init_ticket_context (ticket *T, unsign32 birthtime)

  initialize a ticket structure, include time of creation

5.10.1 Detailed Description

TLS 1.3 process resumption tickets.

Author

Mike Scott

5.10.2 Function Documentation

5.10.2.1 millis()

```
unsigned long millis ( )
```

read milliseconds from a stop-watch

Returns

milliseconds read from stop-watch

5.10.2.2 parseTicket()

```
int parseTicket (
            octet * TICK,
            ticket * T )
```

parse a received ticket octet into a ticket structure

Parameters

TICK	the input ticket octet
T	the output ticket structure

Returns

bad ticket error, or 0 if ticket is good

5.10.2.3 init_ticket_context()

```
void init_ticket_context (
            ticket * T,
            unsign32 birthtime )
```

initialize a ticket structure, include time of creation

Parameters

T	the ticket structure
birthtime	the time when the ticket was born

5.11 tls_wifi.h File Reference

define Socket structure depending on processor context

5.11.1 Detailed Description

define Socket structure depending on processor context

Author

Mike Scott

Index