TiigerTLS

1.2

Generated by Doxygen 1.9.5

Chapter 1

Blogs

These blogs are the completely unfiltered thoughts that passed through the developers head while developing this software. In hindsight much of the commentary turned out to be wrong, misplaced, ill-informed. It may help explain some of the design decisions.

Chapter 2

TIIGER TLS C++

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

2.1 Building

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

Private keys, server/client certificate chains, and CA root stores are all fixed in the code. Therefore as it stands the code must be recompiled for each target.
Ideally keys, chains and key stores should be kept in external files, but in an IoT setting there may not be a file system. In this C++ code the client private key and certificate are stored in the source code file *tls_client_cert.cpp*, and the certificate store is stored in the file *tls_cacert.cpp*. However when using secure hardware, the client private key may not embedded in the source code, rather it exists in secure on-board memory.

2.1.1 Miracl

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

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

2.1.2 Miracl + LibSodium

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

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

2.2 Try it out

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

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

```
./client -s
```

To connect to a Website

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

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

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

Shared Secret= 99A5F3B6F8BE0938AB6D74A99E8FD42DEFD71F25445BD703F0D429DA6CC4AA12
Handshake Secret= 093388E25C3F8468DF3A0544683036CBACF5157874CE995C080807559834CBCA
Client handshake traffic secret= 5B383ED973C7324E267B16A1A7507C380846FFB5397B41E3199C305C23A2C430
Server handshake traffic secret= 71A23E7184F1AA8F228504D3FA735EC8E70FFEC54E0922D553A64800A32C2853
Warning - ALPN extension NOT acknowledged by server
Server Name NOT acknowledged
Max frag length request NOT acknowledged
Encrypted Extensions Processed
Certificate Chain Length= 2458
Parsing Server certificate
Signature is 0A5C155DB6DD9F7F6ABE005D351D6E3FF9DEBA799F7479BD33E1C784B63CB4CA695A76815C9B666C24B6E989EE85009A6
RSA signature of length 2048
Public key= E2AB76AE1A676E3268E39BB9B8AE9CA19DD8BC0BFED0A4275E13C191D716794B48F47766A6B6AD17F19764F48D459E8271
RSA public key of length 2048
Issuer is  R3/Let's Encrypt/
Subject is swifttls.org//
Parsing Intermediate certificate
Signature is D94CE0C9F584883731DBBB13E2B3FC8B6B62126C58B7497E3C02B7A81F2861EBCEE02E73EF49077A35841F1DAD68F0D8F
RSA signature of length 2048
Public key= BB021528CCF6A094D30F12EC8D5592C3F882F199A67A4288A75D26AAB52BB9C54CB1AF8E6BF975C8A3D70F479414553557
RSA public key of length 2048
Issuer is  DST Root CA X3/Digital Signature Trust Co./
Subject is R3/Let's Encrypt/
Signature  = 0A5C155DB6DD9F7F6ABE005D351D6E3FF9DEBA799F7479BD33E1C784B63CB4CA695A76815C9B666C24B6E989EE85009A6
Public key = BB021528CCF6A094D30F12EC8D5592C3F882F199A67A4288A75D26AAB52BB9C54CB1AF8E6BF975C8A3D70F47941455355
Checking Signature on Cert
Signature Algorithm is RSA_PKCS1_SHA256
Cert Signature Verification succeeded
Intermediate Certificate Chain sig is OK

Public Key from root cert= DFAFE99750088357B4CC6265F69082ECC7D32C6B30CA5BECD9C37DC740C118148BE0E83376492AE33F2
Signature  = D94CE0C9F584883731DBBB13E2B3FC8B6B62126C58B7497E3C02B7A81F2861EBCEE02E73EF49077A35841F1DAD68F0D8F
Public key = DFAFE99750088357B4CC6265F69082ECC7D32C6B30CA5BECD9C37DC740C118148BE0E83376492AE33F214993AC4E0EAF3
Checking Signature on Cert
Signature Algorithm is RSA_PKCS1_SHA256
Cert Signature Verification succeeded
Root Certificate sig is OK
```

```
Certificate Chain is valid
Transcript Hash (CH+SH+EE+CT) = 7CECF69D794C20FB7551BA5C4B986E1F501011328225CDD740A8EB54B728E31B
Transcript Hash (CH+SH+EE+SCT+SCV) = 8EC0EE587717BAEB401992622E3F31CBE151CC6C489104E68B5A83E96284E1E7
Server Certificate Signature= B5B74CF6026CF16FA866BA7E7562C53F67A74949FF040319B0BD2149CF4EF97CAD482463F1746D2C
Signature Algorithm is RSA_PSS_RSAE_SHA256
Server Cert Verification OK

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

GET / HTTP/1.1
Host: swifttls.org


Waiting for Server input
Got a ticket
Waiting for Server input
Receiving application data (truncated HTML) = 485454502F312E3120323030204F4B0D0A5365727665723A205377696674544C
Alert sent to Server - Close notify
Connection closed
```

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

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

The output should look something like

```
Attempting resumption
Hostname= swifttls.org

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

PSK= 41301AAD7DAADCF43D700CD71E1198DD2C8DFF5C61B91BEA35116B96762C8B7E
Binder Key= 3CC796B38A7FEB226D9B0CD6B6BB4994253298DDF9FF43060C5C30834D75EE79
Early Secret= 610B9D95E512F6E199046C93E600D5CE10BB98517F9A81096E653C13B2D0F17D
Private key= 7373AF7D060E0E80959254DC071A06905E067B07367C49D86B48A10F3923CC49
Client Public key= 04EA04CDA74C1A1942BB8C56C0BD8AE1A4CB9D9B76B5AC64C24CFE7C367B46FA6F06037D945835019D3F122080
Ticket age= feff
obfuscated age = 447e2e62
Client Hello sent
BND= 258FA2CE9D69253C83646641266B2A81FCEED47348D60E0C7BBB27D2557D1BD2
Sending Binders
Client Early Traffic Secret= CF7D980E8213205CFD35C2194FB75F6D1E98215860BB1F7FA5CFDC8DAE48E9F5
Sending some early data
Sending Application Message

GET / HTTP/1.1
Host: swifttls.org


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

serverHello= 0200009D0303268C69B38026464DFFE72A496662627EC35798DA3F98437042E39CAF404C888520557742FB051C8ADC0A4
```

```
Shared Secret= 8C7784C539C0144B8FADCBF065637418F190C49995E79660919E204F05287C2D
Handshake Secret= 4025A7EE2C1B634C9FC83FDF5CFB2FCB5498EA3F5D019EEDC6D3C1D751C87C47
Client handshake traffic secret= 5FC1307F4E7ED84B4196B83EA19D69724812C25A571061FB53B5B6E9FD7FCABE
Server handshake traffic secret= 1E84FEBA7F8D75F756408906C608925F9A6445292BA614BB398E634CF5854B2A
Early Data Accepted
Warning - ALPN extension NOT acknowledged by server
Server Name NOT acknowledged
Max frag length request NOT acknowledged
Transcript Hash (CH+SH+EE) = DCB73D7B5416D91546EF7D625FBB6A84105CCCE5F054D753275325A822D394E9
Send End of Early Data
Transcript Hash (CH+SH+EE+SF+ED) = FE1FADC8085B3B41A9146647FC9A40F6F2A303533B237112564A2F51F82B64C4
Server Data is verified
Client Verify Data= 350E968A15D36F16BC20D80789E9DB2792A2975765F9BE537407165F7E7366B8
Client application traffic secret= 536F912C98CF4C2D9672DEA57AC8136519607014EFEBBA289FCED97929EA9633
Server application traffic secret= 6B797DBC7FB2D9F75A877F1D34EE7CACC6D65C847C085331F8941C81F2884E83
Resumption Handshake concluded
Early data was accepted
Waiting for Server input
Receiving application data (truncated HTML) = 485454502F312E3120323030204F4B0D0A5365727665723A205377696674544C
Alert sent to Server - Close notify
Connection closed
```

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

Also try

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

Try it a few times - it randomly asks for a HelloRetryRequest and a Key Update, testing this code (but it does not allow resumption)

Resumption tickets can be deleted by

```
./client -r
```

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

2.2.1 Client side Authentication

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

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

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

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

then for example

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

acts as a normal Website, while

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

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

```
./client localhost
```

2.2.2 Testing Pre-shared keys

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

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

and connect via

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

An important setting in *tls1_3.h* is CRYPTO_SETTING. For the above tests is is assumed that this is set to the default TYPICAL, which allows interaction with standard websites. However it may also be set to TINY_ECC, POST_QUANTUM and HYBRID. These last three support interaction with our own rust server. This setting impacts code size and memory resource allocation. It also controls the type of the self-signed certificate provided by the client if it is asked to authenticate.

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

Chapter 3

Configure the Arduino Nano RP2040

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

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

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

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

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

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

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

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

10. Run py config.py, and select options 2, 8, 41 and 43. This creates the default SAL (in this case using miracl + ECC608A hardware).
11. Drop the working directory into where the Arduino IDE expects it.
12. (In the IDE select File->Preferences and find the Sketchbook location - its the libraries directory off that.)
13. Open the Arduino app, and look in File->Examples->tiitls, and look for the example "client"
14. Upload to the board and run it. Open Tools->Serial Monitor to see the output.
15. Enter URL (e.g. www.bbc.co.uk) when prompted, and press return. A full TLS1.3 handshake followed by a resumption is attempted.
16. Click on Clear Output and Send to repeat for a different URL (or click Send again to see SAL capabilities).

or before executing step 10, search for $*$*$*$* in config.py (around line 1020) and make changes as indicated. If using miracl alone, without hardware support, option 3 must be selected as well. If using assembly language code for X25519, copy x25519.S from https://github.com/pornin/x25519-cm0/blob/main/src/x25519-cm0.S into working directory and remove option 2. This creates the SAL (in this case using miracl + ECC608A hardware + Pornin's x25519). If experimenting with post-quantum primitives, also select options 45 and 46, for Kyber and Dilithium support.

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

The client can also be run in conjunction with our Rust server. Make sure that the CRYPTO_SETTING parameter is the same for both client and server. In our experimental set-up, the rust server runs from Windows, looking for connections on port 4433. Run ipconfig to get the IP address of the server on the local network, which might look something like 192.168.1.186. Then run the client from the Arduino IDE, and when prompted enter for example 192.168.1.186:4433. The client should now connect to the server.

Chapter 4

Class Index

4.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 5

File Index

5.1 File List

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

Chapter 6

Class Documentation

6.1 crypto Struct Reference

crypto context structure

`#include <tls1_3.h>`

Public Attributes

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

6.1.1 Detailed Description

crypto context structure

6.1.2 Member Data Documentation

6.1.2.1 active

`bool crypto::active`

Indicates if encryption has been activated

6.1.2.2 iv

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

AEAD cryptographic IV bytes

6.1.2.3 IV

```
octad crypto::IV
```

IV as octad

6.1.2.4 k

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

AEAD cryptographic Key bytes

6.1.2.5 K

```
octad crypto::K
```

Key as octad

6.1.2.6 record

```
unsign32 crypto::record
```

current record number - to be incremented

6.1.2.7 suite

```
int crypto::suite
```

Cipher Suite

6.1.2.8 taglen

```
int crypto::taglen
```

Tag Length

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

- C:/Users/mscot/TLS1.3/cpp/include/tls1_3.h

6.2 ECCX08Class Class Reference

Public Member Functions

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

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

- C:/Users/mscot/TLS1.3/cpp/src/arduino/ECCX08.h
- C:/Users/mscot/TLS1.3/cpp/src/arduino/ECCX08.cpp

6.3 ee_status Struct Reference

server encrypted extensions expectations/responses

`#include <tls1_3.h>`

Public Attributes

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

6.3.1 Detailed Description

server encrypted extensions expectations/responses

6.3.2 Member Data Documentation

6.3.2.1 alpn

```
bool ee_status::alpn
```

true if ALPN accepted

6.3.2.2 early_data

```
bool ee_status::early_data
```

true if early data accepted

6.3.2.3 max_frag_length

```
bool ee_status::max_frag_length
```

true if max frag length respected

6.3.2.4 server_name

```
bool ee_status::server_name
```

true if server name accepted

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

- C:/Users/mscot/TLS1.3/cpp/include/tls1_3.h

6.4 octad Struct Reference

Safe representation of an octad.

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

Public Attributes

- int len
- int max
- char * val

6.4.1 Detailed Description

Safe representation of an octad.

6.4.2 Member Data Documentation

6.4.2.1 len

```
int octad::len
```

length in bytes

6.4.2.2 max

```
int octad::max
```

max length allowed - enforce truncation

6.4.2.3 val

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

byte array

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

- C:/Users/mscot/TLS1.3/cpp/include/tls_octads.h

6.5 pktype Struct Reference

Public key type.

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

Public Attributes

- int type
- int hash
- int curve

6.5.1 Detailed Description

Public key type.

6.5.2 Member Data Documentation

6.5.2.1 curve

```
int pktype::curve
```

elliptic curve used or RSA key length in bits

6.5.2.2 hash

```
int pktype::hash
```

hash type

6.5.2.3 type

```
int pktype::type
```

signature type (ECC or RSA)

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

- C:/Users/mscot/TLS1.3/cpp/include/tls_x509.h

6.6 ret Struct Reference

function return structure

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

Public Attributes

- unsign32 val
- int err

6.6.1 Detailed Description

function return structure

6.6.2 Member Data Documentation

6.6.2.1 err

```
int ret::err
```

error return

6.6.2.2 val

```
unsign32 ret::val
```

return value

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

- C:/Users/mscot/TLS1.3/cpp/include/tls1_3.h

6.7 Socket Class Reference

Socket instance.

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

Public Member Functions

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

Static Public Member Functions

- static Socket **InetSocket** ()
- static Socket **UnixSocket** ()

6.7.1 Detailed Description

Socket instance.

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

- C:/Users/mscot/TLS1.3/cpp/include/tls_sockets.h

6.8 ticket Struct Reference

ticket context structure

`#include <tls1_3.h>`

Public Attributes

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

6.8.1 Detailed Description

ticket context structure

6.8.2 Member Data Documentation

6.8.2.1 age_obfuscator

```
unsign32 ticket::age_obfuscator
```

ticket age obfuscator - 0 for external PSK

6.8.2.2 birth

```
unsign32 ticket::birth
```

Birth time of this ticket

6.8.2.3 cipher_suite

```
int ticket::cipher_suite
```

Cipher suite used

6.8.2.4 favourite_group

```
int ticket::favourite_group
```

the server's favourite group

6.8.2.5 lifetime

```
int ticket::lifetime
```

ticket lifetime

6.8.2.6 max_early_data

```
unsign32 ticket::max_early_data
```

Maximum early data allowed for this ticket

6.8.2.7 nonce

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

nonce

6.8.2.8 NONCE

```
octad ticket::NONCE
```

Nonce as octad

6.8.2.9 origin

```
int ticket::origin
```

Origin of initial handshake - Full or PSK?

6.8.2.10 psk

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

pre-shared key

6.8.2.11 PSK

```
octad ticket::PSK
```

PSK as octad

6.8.2.12 tick

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

Ticket bytes

6.8.2.13 TICK

```
octad ticket::TICK
```

Ticket or external PSK label as octad

6.8.2.14 valid

```
bool ticket::valid
```

Is ticket valid?

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

- C:/Users/mscot/TLS1.3/cpp/include/tls1_3.h

6.9 TLS_session Struct Reference

TLS1.3 session state.

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

Public Attributes

- int status
- int max_record
- Socket * sockptr
- char id [32]
- char hostname [TLS_MAX_SERVER_NAME]
- int cipher_suite
- int favourite_group
- crypto K_send
- crypto K_recv
- octad HS
- char hs [TLS_MAX_HASH]
- octad RMS
- char rms [TLS_MAX_HASH]
- octad STS
- char sts [TLS_MAX_HASH]
- octad CTS
- char cts [TLS_MAX_HASH]
- octad IO
- char io [TLS_MAX_IO_SIZE]
- int ptr
- unihash tlshash
- ticket T

6.9.1 Detailed Description

TLS1.3 session state.

6.9.2 Member Data Documentation

6.9.2.1 cipher_suite

```
int TLS_session::cipher_suite
```

agreed cipher suite

6.9.2.2 CTS

```
octad TLS_session::CTS
```

Client Traffic secret

6.9.2.3 cts

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

Client Traffic secret data

6.9.2.4 favourite_group

```
int TLS_session::favourite_group
```

favourite key exchange group - may be changed on handshake retry

6.9.2.5 hostname

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

Server name for connection

6.9.2.6 HS

```
octad TLS_session::HS
```

Handshake secret

6.9.2.7 hs

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

Handshake secret data

6.9.2.8 id

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

Session ID

6.9.2.9 IO

```
octad TLS_session::IO
```

Main IO buffer for this connection

6.9.2.10 io

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

Byte array for main IO buffer for this connection

6.9.2.11 K_recv

```
crypto TLS_session::K_recv
```

Receiving Key

6.9.2.12 K_send

```
crypto TLS_session::K_send
```

Sending Key

6.9.2.13 max_record

```
int TLS_session::max_record
```

max record size I should send

6.9.2.14 ptr

```
int TLS_session::ptr
```

pointer into IO buffer

6.9.2.15 RMS

```
octad TLS_session::RMS
```

Resumption Master Secret

6.9.2.16 rms

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

Resumption Master Secret data

6.9.2.17 sockptr

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

Pointer to socket

6.9.2.18 status

```
int TLS_session::status
```

Connection status

6.9.2.19 STS

```
octad TLS_session::STS
```

Server Traffic secret

6.9.2.20 sts

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

Server Traffic secret data

6.9.2.21 T

```
ticket TLS_session::T
```

resumption ticket

6.9.2.22 tlshash

```
unihash TLS_session::tlshash
```

Transcript hash recorder

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

- C:/Users/mscot/TLS1.3/cpp/include/tls1_3.h

6.10 unihash Struct Reference

Universal Hash Function.

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

Public Attributes

- char state [TLS_MAX_HASH_STATE]
- int htype

6.10.1 Detailed Description

Universal Hash Function.

6.10.2 Member Data Documentation

6.10.2.1 htype

```
int unihash::htype
```

The hash type (typically SHA256)

6.10.2.2 state

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

hash function state

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

- C:/Users/mscot/TLS1.3/cpp/include/tls1_3.h

Chapter 7

File Documentation

7.1 C:/Users/mscot/TLS1.3/cpp/include/tls1_3.h File Reference

Main TLS 1.3 Header File for constants and structures.

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

Classes

- struct ret
  function return structure
- struct ee_status
  server encrypted extensions expectations/responses
- struct crypto
  crypto context structure
- struct ticket
  ticket context structure
- struct unihash
  Universal Hash Function.
- struct TLS_session
  TLS1.3 session state.

Macros

- #define IO_NONE 0
- #define IO_APPLICATION 1
- #define IO_PROTOCOL 2
- #define IO_DEBUG 3
- #define IO_WIRE 4
- #define TINY_ECC 0
- #define TYPICAL 1
- #define POST_QUANTUM 2

- #define HYBRID 3
- #define NOCERT 0
- #define RSA_SS 1
- #define ECC_SS 2
- #define DLT_SS 3
- #define HYB_SS 6
- #define HW_1 4
- #define HW_2 5
- #define VERBOSITY IO_PROTOCOL
- #define THIS_YEAR 2022
- #define CLIENT_CERT ECC_SS
- #define CRYPTO_SETTING TYPICAL
- #define TLS_APPLICATION_PROTOCOL (char *)("http/1.1")
- #define ALLOW_SELF_SIGNED
- #define TRY_EARLY_DATA
- #define TLS_SHA256_T 1
- #define TLS_SHA384_T 2
- #define TLS_SHA512_T 3
- #define TLS_MAX_HASH_STATE 768
- #define TLS_MAX_HASH 64
- #define TLS_MAX_KEY 32
- #define TLS_X509_MAX_FIELD 256
- #define TLS_MAX_EXT_LABEL 256
- #define TLS_MAX_FRAG 4
- #define TLS_MAX_IO_SIZE (16384+256)
- #define TLS_MAX_PLAIN_FRAG 16384
- #define TLS_MAX_CIPHER_FRAG (16384+256)
- #define TLS_MAX_CERT_SIZE 2048
- #define TLS_MAX_CERT_B64 2800
- #define TLS_MAX_HELLO 1024
- #define TLS_MAX_SIG_PUB_KEY_SIZE 512
- #define TLS_MAX_SIG_SECRET_KEY_SIZE 512
- #define TLS_MAX_SIGNATURE_SIZE 512
- #define TLS_MAX_KEX_PUB_KEY_SIZE 97
- #define TLS_MAX_KEX_CIPHERTEXT_SIZE 97
- #define TLS_MAX_KEX_SECRET_KEY_SIZE 48
- #define TLS_MAX_SERVER_CHAIN_LEN 2
- #define TLS_MAX_SERVER_CHAIN_SIZE (TLS_MAX_SERVER_CHAIN_LEN*TLS_MAX_CERT_SIZE)
- #define TLS_MAX_CLIENT_CHAIN_LEN 1
- #define TLS_MAX_CLIENT_CHAIN_SIZE (TLS_MAX_CLIENT_CHAIN_LEN*TLS_MAX_CERT_SIZE)
- #define TLS_MAX_SHARED_SECRET_SIZE 256
- #define TLS_MAX_TICKET_SIZE 1024
- #define TLS_MAX_EXTENSIONS 2048
- #define TLS_MAX_ECC_FIELD 66
- #define TLS_MAX_IV_SIZE 12
- #define TLS_MAX_TAG_SIZE 16
- #define TLS_MAX_COOKIE 128
- #define TLS_MAX_SERVER_NAME 128
- #define TLS_MAX_SUPPORTED_GROUPS 10
- #define TLS_MAX_SUPPORTED_SIGS 16
- #define TLS_MAX_PSK_MODES 2
- #define TLS_MAX_CIPHER_SUITES 5
- #define TLS_AES_128_GCM_SHA256 0x1301
- #define TLS_AES_256_GCM_SHA384 0x1302
- #define TLS_CHACHA20_POLY1305_SHA256 0x1303

- #define TLS_AES_128_CCM_SHA256 0x1304
- #define TLS_AES_128_CCM_8_SHA256 0x1305
- #define X25519 0x001d
- #define SECP256R1 0x0017
- #define SECP384R1 0x0018
- #define SECP521R1 0x0019
- #define X448 0x001e
- #define KYBER768 0x4242
- #define HYBRID_KX 0x421d
- #define ECDSA_SECP256R1_SHA256 0x0403
- #define ECDSA_SECP256R1_SHA384 0x0413
- #define ECDSA_SECP384R1_SHA384 0x0503
- #define RSA_PSS_RSAE_SHA256 0x0804
- #define RSA_PSS_RSAE_SHA384 0x0805
- #define RSA_PSS_RSAE_SHA512 0x0806
- #define RSA_PKCS1_SHA256 0x0401
- #define RSA_PKCS1_SHA384 0x0501
- #define RSA_PKCS1_SHA512 0x0601
- #define ED25519 0x0807
- #define DILITHIUM2 0x0902
- #define DILITHIUM3 0x0903
- #define DILITHIUM2_P256 0x09F2
- #define PSKOK 0x00
- #define PSKWECDHE 0x01
- #define TLS_FULL_HANDSHAKE 1
- #define TLS_EXTERNAL_PSK 2
- #define TLS1_0 0x0301
- #define TLS1_2 0x0303
- #define TLS1_3 0x0304
- #define SERVER_NAME 0x0000
- #define SUPPORTED_GROUPS 0x000a
- #define SIG_ALGS 0x000d
- #define SIG_ALGS_CERT 0x0032
- #define KEY_SHARE 0x0033
- #define PSK_MODE 0x002d
- #define PRESHARED_KEY 0x0029
- #define TLS_VER 0x002b
- #define COOKIE 0x002c
- #define EARLY_DATA 0x002a
- #define MAX_FRAG_LENGTH 0x0001
- #define PADDING 0x0015
- #define APP_PROTOCOL 0x0010
- #define RECORD_SIZE_LIMIT 0x001c
- #define HSHAKE 0x16
- #define APPLICATION 0x17
- #define ALERT 0x15
- #define CHANGE_CIPHER 0x14
- #define TIMED_OUT 0x01
- #define CLIENT_HELLO 0x01
- #define SERVER_HELLO 0x02
- #define CERTIFICATE 0x0b
- #define CERT_REQUEST 0x0d
- #define CERT_VERIFY 0x0f
- #define FINISHED 0x14
- #define ENCRYPTED_EXTENSIONS 0x08

- #define TICKET 0x04
- #define KEY_UPDATE 0x18
- #define MESSAGE_HASH 0xFE
- #define END_OF_EARLY_DATA 0x05
- #define HANDSHAKE_RETRY 0x102
- #define NOT_TLS1_3 -2
- #define BAD_CERT_CHAIN -3
- #define ID_MISMATCH -4
- #define UNRECOGNIZED_EXT -5
- #define BAD_HELLO -6
- #define WRONG_MESSAGE -7
- #define MISSING_REQUEST_CONTEXT -8
- #define AUTHENTICATION_FAILURE -9
- #define BAD_RECORD -10
- #define BAD_TICKET -11
- #define NOT_EXPECTED -12
- #define CA_NOT_FOUND -13
- #define CERT_OUTOFDATE -14
- #define MEM_OVERFLOW -15
- #define FORBIDDEN_EXTENSION -16
- #define MAX_EXCEEDED -17
- #define EMPTY_CERT_CHAIN -18
- #define SELF_SIGNED_CERT -20
- #define BAD_MESSAGE -23
- #define ILLEGAL_PARAMETER 0x2F
- #define UNEXPECTED_MESSAGE 0x0A
- #define DECRYPT_ERROR 0x33
- #define BAD_CERTIFICATE 0x2A
- #define UNSUPPORTED_EXTENSION 0x6E
- #define UNKNOWN_CA 0x30
- #define CERTIFICATE_EXPIRED 0x2D
- #define PROTOCOL_VERSION 0x46
- #define DECODE_ERROR 0x32
- #define RECORD_OVERFLOW 0x16
- #define CLOSE_NOTIFY 0x00
- #define LOG_OUTPUT_TRUNCATION 256
- #define TLS13_DISCONNECTED 0
- #define TLS13_CONNECTED 1
- #define TLS_FAILURE 0
- #define TLS_SUCCESS 1
- #define TLS_RESUMPTION_REQUIRED 2
- #define TLS_EARLY_DATA_ACCEPTED 3

Typedefs

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

7.1.1 Detailed Description

Main TLS 1.3 Header File for constants and structures.

Author

Mike Scott

7.1.2 Macro Definition Documentation

7.1.2.1 ALERT

```
#define ALERT 0x15
```

Alert record

7.1.2.2 ALLOW_SELF_SIGNED

```
#define ALLOW_SELF_SIGNED
```

allow self-signed server cert

7.1.2.3 APP_PROTOCOL

```
#define APP_PROTOCOL 0x0010
```

Application Layer Protocol Negotiation (ALPN)

7.1.2.4 APPLICATION

```
#define APPLICATION 0x17
```

Application record

7.1.2.5 AUTHENTICATION_FAILURE

```
#define AUTHENTICATION_FAILURE -9
```

Authentication error - AEAD Tag incorrect

7.1.2.6 BAD_CERT_CHAIN

```
#define BAD_CERT_CHAIN -3
```

Bad Certificate Chain error

7.1.2.7 BAD_CERTIFICATE

```
#define BAD_CERTIFICATE 0x2A
```

Bad certificate alert

7.1.2.8 BAD_HELLO

```
#define BAD_HELLO -6
```

badly formed Hello message error

7.1.2.9 BAD_MESSAGE

```
#define BAD_MESSAGE -23
```

Badly formed message

7.1.2.10 BAD_RECORD

```
#define BAD_RECORD -10
```

Badly formed Record received

7.1.2.11 BAD_TICKET

```
#define BAD_TICKET -11
```

Badly formed Ticket received

7.1.2.12 CA_NOT_FOUND

```
#define CA_NOT_FOUND -13
```

Certificate Authority not found

7.1.2.13 CERT_OUTOFDATE

```
#define CERT_OUTOFDATE -14
```

Certificate Expired

7.1.2.14 CERT_REQUEST

```
#define CERT_REQUEST 0x0d
```

Certificate Request

7.1.2.15 CERT_VERIFY

```
#define CERT_VERIFY 0x0f
```

Certificate Verify message

7.1.2.16 CERTIFICATE

```
#define CERTIFICATE 0x0b
```

Certificate message

7.1.2.17 CERTIFICATE_EXPIRED

```
#define CERTIFICATE_EXPIRED 0x2D
```

Certificate Expired

7.1.2.18 CHANGE_CIPHER

```
#define CHANGE_CIPHER 0x14
```

Change Cipher record

7.1.2.19 CLIENT_CERT

```
#define CLIENT_CERT ECC_SS
```

Indicate capability of authenticating with a cert plus signing key

7.1.2.20 CLIENT_HELLO

```
#define CLIENT_HELLO 0x01
```

Client Hello message

7.1.2.21 CLOSE_NOTIFY

```
#define CLOSE_NOTIFY 0x00
```

Orderly shut down of connection

7.1.2.22 COOKIE

```
#define COOKIE 0x002c
```

Cookie extension

7.1.2.23 CRYPTO_SETTING

```
#define CRYPTO_SETTING TYPICAL
```

Determine Cryptography settings

7.1.2.24 DECODE_ERROR

```
#define DECODE_ERROR 0x32
```

Decode error alert

7.1.2.25 DECRYPT_ERROR

```
#define DECRYPT_ERROR 0x33
```

Decryption error alert

7.1.2.26 DILITHIUM2

```
#define DILITHIUM2 0x0902
```

Dilithium2 Signature algorithm

7.1.2.27 DILITHIUM2_P256

```
#define DILITHIUM2_P256 0x09F2
```

Dilithium2+SECP256R1 Signature algorithms - this type can be negotiated, but always implemented seperately by SAL

7.1.2.28 DILITHIUM3

```
#define DILITHIUM3 0x0903
```

Dilithium3 Signature algorithm

7.1.2.29 DLT_SS

```
#define DLT_SS 3
```

self signed Dilithium cert

7.1.2.30 EARLY_DATA

```
#define EARLY_DATA 0x002a
```

Early Data extension

7.1.2.31 ECC_SS

```
#define ECC_SS 2
```

self signed ECC cert

7.1.2.32 ECDSA_SECP256R1_SHA256

```
#define ECDSA_SECP256R1_SHA256 0x0403
```

Supported ECDSA Signature algorithm

7.1.2.33 ECDSA_SECP256R1_SHA384

```
#define ECDSA_SECP256R1_SHA384 0x0413
```

Non-standard ECDSA Signature algorithm

7.1.2.34 ECDSA_SECP384R1_SHA384

```
#define ECDSA_SECP384R1_SHA384 0x0503
```

Supported ECDSA Signature algorithm

7.1.2.35 ED25519

```
#define ED25519 0x0807
```

Ed25519 EdDSA Signature algorithm

7.1.2.36 EMPTY_CERT_CHAIN

```
#define EMPTY_CERT_CHAIN -18
```

Empty Certificate Message

7.1.2.37 ENCRYPTED_EXTENSIONS

```
#define ENCRYPTED_EXTENSIONS 0x08
```

Encrypted Extensions message

7.1.2.38 END_OF_EARLY_DATA

```
#define END_OF_EARLY_DATA 0x05
```

End of Early Data message

7.1.2.39 FINISHED

```
#define FINISHED 0x14
```

Handshake Finished message

7.1.2.40 FORBIDDEN_EXTENSION

```
#define FORBIDDEN_EXTENSION -16
```

Forbidden Encrypted Extension

7.1.2.41 HANDSHAKE_RETRY

```
#define HANDSHAKE_RETRY 0x102
```

Handshake retry

7.1.2.42 HSHAKE

```
#define HSHAKE 0x16
```

Handshake record

7.1.2.43 HW_1

```
#define HW_1 4
```

RP2040 1 Hardware cert

7.1.2.44 HW_2

```
#define HW_2 5
```

RP2040 2 Hardware cert

7.1.2.45 HYB_SS

```
#define HYB_SS 6
```

self signed Hybrid cert (Dilithium+ECC)

7.1.2.46 HYBRID

```
#define HYBRID 3
```

Hybrid, Kyber/Dilithium + X25519

7.1.2.47 HYBRID_KX

```
#define HYBRID_KX 0x421d
```

Hybrid key exchange, Kyber+X25519

7.1.2.48 ID_MISMATCH

```
#define ID_MISMATCH -4
```

Session ID mismatch error

7.1.2.49 ILLEGAL_PARAMETER

```
#define ILLEGAL_PARAMETER 0x2F
```

Illegal parameter alert

7.1.2.50 IO_APPLICATION

```
#define IO_APPLICATION 1
```

just print application traffic

7.1.2.51 IO_DEBUG

```
#define IO_DEBUG 3
```

print lots of debug information + protocol progress + application progress

7.1.2.52 IO_NONE

```
#define IO_NONE 0
```

Run silently

7.1.2.53 IO_PROTOCOL

```
#define IO_PROTOCOL 2
```

print protocol progress + application traffic

7.1.2.54 IO_WIRE

```
#define IO_WIRE 4
```

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

7.1.2.55 KEY_SHARE

```
#define KEY_SHARE 0x0033
```

Key Share extension

7.1.2.56 KEY_UPDATE

```
#define KEY_UPDATE 0x18
```

Key Update message

7.1.2.57 KYBER768

```
#define KYBER768 0x4242
```

Kyber PQ key exchange - NOTE I just made this up! Not generally recognised!

7.1.2.58 LOG_OUTPUT_TRUNCATION

```
#define LOG_OUTPUT_TRUNCATION 256
```

Output Hex digits before truncation

7.1.2.59 MAX_EXCEEDED

```
#define MAX_EXCEEDED -17
```

Maximum record size exceeded

7.1.2.60 MAX_FRAG_LENGTH

```
#define MAX_FRAG_LENGTH 0x0001
```

max fragmentation length extension

7.1.2.61 MEM_OVERFLOW

```
#define MEM_OVERFLOW -15
```

Memory Overflow

7.1.2.62 MESSAGE_HASH

```
#define MESSAGE_HASH 0xFE
```

Special synthetic message hash message

7.1.2.63 MISSING_REQUEST_CONTEXT

```
#define MISSING_REQUEST_CONTEXT -8
```

Request context missing error

7.1.2.64 NOCERT

```
#define NOCERT 0
```

Don't have a Client Cert

7.1.2.65 NOT_EXPECTED

```
#define NOT_EXPECTED -12
```

Received ack for something not requested

7.1.2.66 NOT_TLS1_3

```
#define NOT_TLS1_3 -2
```

Wrong version error, not TLS1.3

7.1.2.67 PADDING

```
#define PADDING 0x0015
```

Padding extension

7.1.2.68 POST_QUANTUM

```
#define POST_QUANTUM 2
```

Post quantum (Dilithium+Kyber?)

7.1.2.69 PRESHARED_KEY

```
#define PRESHARED_KEY 0x0029
```

Preshared key extension

7.1.2.70 PROTOCOL_VERSION

```
#define PROTOCOL_VERSION 0x46
```

Wrong TLS version

7.1.2.71 PSK_MODE

```
#define PSK_MODE 0x002d
```

Preshared key mode extension

7.1.2.72 PSKOK

```
#define PSKOK 0x00
```

Preshared Key only mode

7.1.2.73 PSKWECDHE

```
#define PSKWECDHE 0x01
```

Preshared Key with Diffie-Hellman key exchange mode

7.1.2.74 RECORD_OVERFLOW

```
#define RECORD_OVERFLOW 0x16
```

Record Overflow

7.1.2.75 RECORD_SIZE_LIMIT

```
#define RECORD_SIZE_LIMIT 0x001c
```

Record Size Limit

7.1.2.76 RSA_PKCS1_SHA256

```
#define RSA_PKCS1_SHA256 0x0401
```

Supported RSA Signature algorithm

7.1.2.77 RSA_PKCS1_SHA384

```
#define RSA_PKCS1_SHA384 0x0501
```

Supported RSA Signature algorithm

7.1.2.78 RSA_PKCS1_SHA512

```
#define RSA_PKCS1_SHA512 0x0601
```

Supported RSA Signature algorithm

7.1.2.79 RSA_PSS_RSAE_SHA256

```
#define RSA_PSS_RSAE_SHA256 0x0804
```

Supported RSA Signature algorithm

7.1.2.80 RSA_PSS_RSAE_SHA384

```
#define RSA_PSS_RSAE_SHA384 0x0805
```

Supported RSA Signature algorithm

7.1.2.81 RSA_PSS_RSAE_SHA512

```
#define RSA_PSS_RSAE_SHA512 0x0806
```

Supported RSA Signature algorithm

7.1.2.82 RSA_SS

```
#define RSA_SS 1
```

self signed RSA cert

7.1.2.83 SECP256R1

```
#define SECP256R1 0x0017
```

NIST SECP256R1 elliptic curve key exchange

7.1.2.84 SECP384R1

```
#define SECP384R1 0x0018
```

NIST SECP384R1 elliptic curve key exchange

7.1.2.85 SECP521R1

```
#define SECP521R1 0x0019
```

NIST SECP521R1 elliptic curve key exchange

7.1.2.86 SELF_SIGNED_CERT

```
#define SELF_SIGNED_CERT -20
```

Self signed certificate

7.1.2.87 SERVER_HELLO

```
#define SERVER_HELLO 0x02
```

Server Hello message

7.1.2.88 SERVER_NAME

```
#define SERVER_NAME 0x0000
```

Server Name extension

7.1.2.89 SIG_ALGS

```
#define SIG_ALGS 0x000d
```

Signature algorithms extension

7.1.2.90 SIG_ALGS_CERT

```
#define SIG_ALGS_CERT 0x0032
```

Signature algorithms Certificate extension

7.1.2.91 SUPPORTED_GROUPS

```
#define SUPPORTED_GROUPS 0x000a
```

Supported Group extension

7.1.2.92 THIS_YEAR

```
#define THIS_YEAR 2022
```

Set to this year - crudely used to deprecate old certificates

7.1.2.93 TICKET

```
#define TICKET 0x04
```

Ticket message

7.1.2.94 TIMED_OUT

```
#define TIMED_OUT 0x01
```

Time-out

7.1.2.95 TINY_ECC

```
#define TINY_ECC 0
```

ECC keys only

7.1.2.96 TLS13_CONNECTED

```
#define TLS13_CONNECTED 1
```

TLS1.3 Connection is made

7.1.2.97 TLS13_DISCONNECTED

```
#define TLS13_DISCONNECTED 0
```

TLS1.3 Connection is broken

7.1.2.98 TLS1_0

```
#define TLS1_0 0x0301
```

TLS 1.0 version

7.1.2.99 TLS1_2

```
#define TLS1_2 0x0303
```

TLS 1.2 version

7.1.2.100 TLS1_3

```
#define TLS1_3 0x0304
```

TLS 1.3 version

7.1.2.101 TLS_AES_128_CCM_8_SHA256

```
#define TLS_AES_128_CCM_8_SHA256 0x1305
```

AES/SHA256/CCM 8 cipher suite - optional

7.1.2.102 TLS_AES_128_CCM_SHA256

```
#define TLS_AES_128_CCM_SHA256 0x1304
```

AES/SHA256/CCM cipher suite - optional

7.1.2.103 TLS_AES_128_GCM_SHA256

```
#define TLS_AES_128_GCM_SHA256 0x1301
```

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

7.1.2.104 TLS_AES_256_GCM_SHA384

```
#define TLS_AES_256_GCM_SHA384 0x1302
```

AES256/SHA384/GCM cipher suite

7.1.2.105 TLS_APPLICATION_PROTOCOL

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

Support ALPN protocol

7.1.2.106 TLS_CHACHA20_POLY1305_SHA256

```
#define TLS_CHACHA20_POLY1305_SHA256 0x1303
```

CHACHA20/SHA256/POLY1305 cipher suite

7.1.2.107 TLS_EARLY_DATA_ACCEPTED

```
#define TLS_EARLY_DATA_ACCEPTED 3
```

Connection succeeded, and early data was accepted

7.1.2.108 TLS_EXTERNAL_PSK

```
#define TLS_EXTERNAL_PSK 2
```

External Pre-Shared Key

7.1.2.109 TLS_FAILURE

```
#define TLS_FAILURE 0
```

Failed to cmake TLS1.3 connection

7.1.2.110 TLS_FULL_HANDSHAKE

```
#define TLS_FULL_HANDSHAKE 1
```

Came from Full Handshake

7.1.2.111 TLS_MAX_CERT_B64

```
#define TLS_MAX_CERT_B64 2800
```

In base64 - current max for root CAs is 2688

7.1.2.112 TLS_MAX_CERT_SIZE

```
#define TLS_MAX_CERT_SIZE 2048
```

I checked - current max for root CAs is 2016

7.1.2.113 TLS_MAX_CIPHER_FRAG

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

Max Ciphertext Fragment size

7.1.2.114 TLS_MAX_CIPHER_SUITES

```
#define TLS_MAX_CIPHER_SUITES 5
```

Max number of supported cipher suites

7.1.2.115 TLS_MAX_CLIENT_CHAIN_LEN

```
#define TLS_MAX_CLIENT_CHAIN_LEN 1
```

Maximum Client Certificate chain length - one self signed here

7.1.2.116 TLS_MAX_CLIENT_CHAIN_SIZE

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

Maximum Client Certificate chain length in bytes

7.1.2.117 TLS_MAX_COOKIE

```
#define TLS_MAX_COOKIE 128
```

Max Cookie size

7.1.2.118 TLS_MAX_ECC_FIELD

```
#define TLS_MAX_ECC_FIELD 66
```

Max ECC field size in bytes

7.1.2.119 TLS_MAX_EXT_LABEL

```
#define TLS_MAX_EXT_LABEL 256
```

Max external psk label size

7.1.2.120 TLS_MAX_EXTENSIONS

```
#define TLS_MAX_EXTENSIONS 2048
```

Max extensions size

7.1.2.121 TLS_MAX_FRAG

```
#define TLS_MAX_FRAG 4
```

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

7.1.2.122 TLS_MAX_HASH

```
#define TLS_MAX_HASH 64
```

Maximum hash output length in bytes

7.1.2.123 TLS_MAX_HASH_STATE

```
#define TLS_MAX_HASH_STATE 768
```

Maximum memory required to store hash function state

7.1.2.124 TLS_MAX_HELLO

```
#define TLS_MAX_HELLO 1024
```

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

7.1.2.125 TLS_MAX_IO_SIZE

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

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

7.1.2.126 TLS_MAX_IV_SIZE

```
#define TLS_MAX_IV_SIZE 12
```

Max IV size in bytes

7.1.2.127 TLS_MAX_KEX_CIPHERTEXT_SIZE

```
#define TLS_MAX_KEX_CIPHERTEXT_SIZE 97
```

Max key exchange (KEM) ciphertext size ECC

7.1.2.128 TLS_MAX_KEX_PUB_KEY_SIZE

```
#define TLS_MAX_KEX_PUB_KEY_SIZE 97
```

Max key exchange public key size in bytes ECC

7.1.2.129 TLS_MAX_KEX_SECRET_KEY_SIZE

```
#define TLS_MAX_KEX_SECRET_KEY_SIZE 48
```

Max key exchange private key size in bytes ECC

7.1.2.130 TLS_MAX_KEY

```
#define TLS_MAX_KEY 32
```

Maximum key length in bytes

7.1.2.131 TLS_MAX_PLAIN_FRAG

```
#define TLS_MAX_PLAIN_FRAG 16384
```

Max Plaintext Fragment size

7.1.2.132 TLS_MAX_PSK_MODES

```
#define TLS_MAX_PSK_MODES 2
```

Max preshared key modes

7.1.2.133 TLS_MAX_SERVER_CHAIN_LEN

```
#define TLS_MAX_SERVER_CHAIN_LEN 2
```

Maximum Server Certificate chain length - omitting root CA

7.1.2.134 TLS_MAX_SERVER_CHAIN_SIZE

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

Maximum Server Certificate chain length in bytes

7.1.2.135 TLS_MAX_SERVER_NAME

```
#define TLS_MAX_SERVER_NAME 128
```

Max server name size in bytes

7.1.2.136 TLS_MAX_SHARED_SECRET_SIZE

```
#define TLS_MAX_SHARED_SECRET_SIZE 256
```

Max key exchange Shared secret size

7.1.2.137 TLS_MAX_SIG_PUB_KEY_SIZE

```
#define TLS_MAX_SIG_PUB_KEY_SIZE 512
```

Max signature public key size in bytes RSA

7.1.2.138 TLS_MAX_SIG_SECRET_KEY_SIZE

```
#define TLS_MAX_SIG_SECRET_KEY_SIZE 512
```

Max signature private key size in bytes RSA

7.1.2.139 TLS_MAX_SIGNATURE_SIZE

```
#define TLS_MAX_SIGNATURE_SIZE 512
```

Max digital signature size in bytes RSA

7.1.2.140 TLS_MAX_SUPPORTED_GROUPS

```
#define TLS_MAX_SUPPORTED_GROUPS 10
```

Max number of supported crypto groups

7.1.2.141 TLS_MAX_SUPPORTED_SIGS

```
#define TLS_MAX_SUPPORTED_SIGS 16
```

Max number of supported signature schemes

7.1.2.142 TLS_MAX_TAG_SIZE

```
#define TLS_MAX_TAG_SIZE 16
```

Max HMAC tag length in bytes

7.1.2.143 TLS_MAX_TICKET_SIZE

```
#define TLS_MAX_TICKET_SIZE 1024
```

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

7.1.2.144 TLS_RESUMPTION_REQUIRED

```
#define TLS_RESUMPTION_REQUIRED 2
```

Connection succeeded, but handshake retry was needed

7.1.2.145 TLS_SHA256_T

```
#define TLS_SHA256_T 1
```

SHA256 hash

7.1.2.146 TLS_SHA384_T

```
#define TLS_SHA384_T 2
```

SHA384 hash

7.1.2.147 TLS_SHA512_T

```
#define TLS_SHA512_T 3
```

SHA512 hash

7.1.2.148 TLS_SUCCESS

```
#define TLS_SUCCESS 1
```

Succeeded in making TLS1.3 connection

7.1.2.149 TLS_VER

```
#define TLS_VER 0x002b
```

TLS version extension

7.1.2.150 TLS_X509_MAX_FIELD

```
#define TLS_X509_MAX_FIELD 256
```

Maximum X.509 field size

7.1.2.151 TRY_EARLY_DATA

```
#define TRY_EARLY_DATA
```

Try to send early data on resumptions

7.1.2.152 TYPICAL

```
#define TYPICAL 1
```

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

7.1.2.153 UNEXPECTED_MESSAGE

```
#define UNEXPECTED_MESSAGE 0x0A
```

Unexpected message alert

7.1.2.154 UNKNOWN_CA

```
#define UNKNOWN_CA 0x30
```

Unrecognised Certificate Authority

7.1.2.155 UNRECOGNIZED_EXT

```
#define UNRECOGNIZED_EXT -5
```

Unrecognised extension error

7.1.2.156 UNSUPPORTED_EXTENSION

```
#define UNSUPPORTED_EXTENSION 0x6E
```

Unsupported extension alert

7.1.2.157 VERBOSITY

```
#define VERBOSITY IO_PROTOCOL
```

Set to level of output information desired - see above

7.1.2.158 WRONG_MESSAGE

```
#define WRONG_MESSAGE -7
```

Message out-of-order error

7.1.2.159 X25519

```
#define X25519 0x001d
```

X25519 elliptic curve key exchange

7.1.2.160 X448

```
#define X448 0x001e
```

X448 elliptic curve key exchange

7.1.3 Typedef Documentation

7.1.3.1 byte

```
typedef uint8_t byte
```

8-bit unsigned integer

7.1.3.2 sign16

```
typedef int16_t sign16
```

16-bit signed integer

7.1.3.3 sign32

```
typedef int32_t sign32
```

32-bit signed integer

7.1.3.4 sign64

```
typedef int64_t sign64
```

64-bit signed integer

7.1.3.5 sign8

```
typedef int8_t sign8
```

8-bit signed integer

7.1.3.6 unsign32

```
typedef uint32_t unsign32
```

32-bit unsigned integer

7.1.3.7 unsign64

```
typedef uint64_t unsign64
```

64-bit unsigned integer

7.2 tls1_3.h

Go to the documentation of this file.

```
1
8 #ifndef TLS1_3_H
9 #define TLS1_3_H
10
11 #include <stdint.h>
12 #include "tls_octads.h"
13 #include "tls_sockets.h"
14
15 typedef uint8_t byte;
16 typedef int8_t sign8 ;
17 typedef int16_t sign16;
18 typedef int32_t sign32;
19 typedef int64_t sign64;
20 typedef uint32_t unsign32 ;
21 typedef uint64_t unsign64;
23 // Terminal Output
24 #define IO_NONE 0
25 #define IO_APPLICATION 1
26 #define IO_PROTOCOL 2
27 #define IO_DEBUG 3
28 #define IO_WIRE 4
30 // Cryptographic Environment
31 #define TINY_ECC 0
32 #define TYPICAL 1
33 #define POST_QUANTUM 2
34 #define HYBRID 3
36 // Client Certificate Chain + Key
37 #define NOCERT 0
38 #define RSA_SS 1
39 #define ECC_SS 2
40 #define DLT_SS 3
41 #define HYB_SS 6
42 #define HW_1 4
43 #define HW_2 5
45 // THESE ARE IMPORTANT USER DEFINED SETTINGS **********************************
46
47 // Note that favourite group (as used in client hello) is determined by the SAL ordering - see
       tls_sal.cpp
48 // If server does not support it, an expensive Handshake Retry will be required
49 // So best to place a popular group (such as X25519) at top of list in SAL
50
51 #define VERBOSITY IO_PROTOCOL
52 #define THIS_YEAR 2022
53 #define CLIENT_CERT ECC_SS
54 #define CRYPTO_SETTING TYPICAL
55 // Supported protocols
56 #define TLS_APPLICATION_PROTOCOL (char *)("http/1.1")
57 #define ALLOW_SELF_SIGNED
58 #define TRY_EARLY_DATA
60 // Note that the IOBUFF, Certificates and crypto keys can be quite large, and therefore maybe better
       taken from the heap
61 // on systems with a shallow stack. Define this to use the heap.
62
63 //#define SHALLOW_STACK           /**< Get large arrays from heap, else stack */
64
65 // comment out if no max record size. In practise TLS1.3 doesn't seem to support this record_size_limit
       extension, so use with caution
66 // #define MAX_RECORD 1024     /**< Maximum record size client is willing to receive - should be less
       than TLS_MAX_IO_SIZE below */
67 // Note that if this is not used, max_fragment_size extension is tried instead, see TLS_MAX_FRAG below
68 // ******************************************************************************
69
70 // Standard Hash Types
71
72 #define TLS_SHA256_T 1
73 #define TLS_SHA384_T 2
74 #define TLS_SHA512_T 3
76 // Some maximum sizes for stack allocated memory. Handshake will fail if these sizes are exceeded!
77
78 #define TLS_MAX_HASH_STATE 768
79 #define TLS_MAX_HASH 64
80 #define TLS_MAX_KEY 32
81 #define TLS_X509_MAX_FIELD 256
82 #define TLS_MAX_EXT_LABEL 256
84 // Max Frag length must be less than TLS_MAX_IO_SIZE
85 #define TLS_MAX_FRAG 4
87 #if CRYPTO_SETTING==TYPICAL
88  #define TLS_MAX_IO_SIZE (16384+256)
89  #define TLS_MAX_PLAIN_FRAG 16384
90  #define TLS_MAX_CIPHER_FRAG (16384+256)
92  #define TLS_MAX_CERT_SIZE 2048
93  #define TLS_MAX_CERT_B64 2800
```

```
94  #define TLS_MAX_HELLO 1024
96  #define TLS_MAX_SIG_PUB_KEY_SIZE 512
97  #define TLS_MAX_SIG_SECRET_KEY_SIZE 512
98  #define TLS_MAX_SIGNATURE_SIZE 512
99  #define TLS_MAX_KEX_PUB_KEY_SIZE 97
100  #define TLS_MAX_KEX_CIPHERTEXT_SIZE 97
101  #define TLS_MAX_KEX_SECRET_KEY_SIZE 48
102 #endif
103
104 #if CRYPTO_SETTING == POST_QUANTUM
105
106  #define TLS_MAX_IO_SIZE (16384+256)
107  #define TLS_MAX_PLAIN_FRAG 16384
108  #define TLS_MAX_CIPHER_FRAG (16384+256)
110  #define TLS_MAX_CERT_SIZE 6144
111  #define TLS_MAX_CERT_B64 8192
112  #define TLS_MAX_HELLO 2048
114 // These all blow up post quantum
115  #define TLS_MAX_SIG_PUB_KEY_SIZE 1952
116  #define TLS_MAX_SIG_SECRET_KEY_SIZE 4000
117  #define TLS_MAX_SIGNATURE_SIZE 3296
118  #define TLS_MAX_KEX_PUB_KEY_SIZE 1184
119  #define TLS_MAX_KEX_CIPHERTEXT_SIZE 1088
120  #define TLS_MAX_KEX_SECRET_KEY_SIZE 2400
121 #endif
122
123 #if CRYPTO_SETTING == HYBRID
124
125  #define TLS_MAX_IO_SIZE (16384+256)
126  #define TLS_MAX_PLAIN_FRAG 16384
127  #define TLS_MAX_CIPHER_FRAG (16384+256)
129  #define TLS_MAX_CERT_SIZE 6144
130  #define TLS_MAX_CERT_B64 8192
131  #define TLS_MAX_HELLO 2048
133 // These all blow up post quantum
134  #define TLS_MAX_SIG_PUB_KEY_SIZE 1312+65
135  #define TLS_MAX_SIG_SECRET_KEY_SIZE 2528+200
136  #define TLS_MAX_SIGNATURE_SIZE 2420+100
137  #define TLS_MAX_KEX_PUB_KEY_SIZE 1184+32
138  #define TLS_MAX_KEX_CIPHERTEXT_SIZE 1088+32
139  #define TLS_MAX_KEX_SECRET_KEY_SIZE 2400+32
140 #endif
141
142
143 #if CRYPTO_SETTING==TINY_ECC
144  #define TLS_MAX_IO_SIZE (4096+256)
145  #define TLS_MAX_PLAIN_FRAG 4096
146  #define TLS_MAX_CIPHER_FRAG (4096+256)
148  #define TLS_MAX_CERT_SIZE 2048
149  #define TLS_MAX_CERT_B64 2800
150  #define TLS_MAX_HELLO 1024
152  #define TLS_MAX_SIG_PUB_KEY_SIZE 133
153  #define TLS_MAX_SIG_SECRET_KEY_SIZE 66
154  #define TLS_MAX_SIGNATURE_SIZE 132
155  #define TLS_MAX_KEX_PUB_KEY_SIZE 97
156  #define TLS_MAX_KEX_CIPHERTEXT_SIZE 97
157  #define TLS_MAX_KEX_SECRET_KEY_SIZE 48
158 #endif
159
160 // Certificate size limits
161 #define TLS_MAX_SERVER_CHAIN_LEN 2
162 #define TLS_MAX_SERVER_CHAIN_SIZE (TLS_MAX_SERVER_CHAIN_LEN*TLS_MAX_CERT_SIZE)
163 #define TLS_MAX_CLIENT_CHAIN_LEN 1
164 #define TLS_MAX_CLIENT_CHAIN_SIZE (TLS_MAX_CLIENT_CHAIN_LEN*TLS_MAX_CERT_SIZE)
167 #define TLS_MAX_SHARED_SECRET_SIZE 256
168 #define TLS_MAX_TICKET_SIZE 1024
169 #define TLS_MAX_EXTENSIONS 2048
171 #define TLS_MAX_ECC_FIELD 66
172 #define TLS_MAX_IV_SIZE 12
173 #define TLS_MAX_TAG_SIZE 16
174 #define TLS_MAX_COOKIE 128
176 #define TLS_MAX_SERVER_NAME 128
177 #define TLS_MAX_SUPPORTED_GROUPS 10
178 #define TLS_MAX_SUPPORTED_SIGS 16
179 #define TLS_MAX_PSK_MODES 2
180 #define TLS_MAX_CIPHER_SUITES 5
182 // Cipher Suites
183 #define TLS_AES_128_GCM_SHA256 0x1301
184 #define TLS_AES_256_GCM_SHA384 0x1302
185 #define TLS_CHACHA20_POLY1305_SHA256 0x1303
186 #define TLS_AES_128_CCM_SHA256 0x1304
187 #define TLS_AES_128_CCM_8_SHA256 0x1305
189 // Key exchange groups
190 #define X25519 0x001d
191 #define SECP256R1 0x0017
192 #define SECP384R1 0x0018
193 #define SECP521R1 0x0019
```

```
194 #define X448 0x001e
195 #define KYBER768 0x4242
196 #define HYBRID_KX 0x421d
198 // Signature algorithms for TLS1.3 and Certs that we can handle
199 #define ECDSA_SECP256R1_SHA256 0x0403
200 #define ECDSA_SECP256R1_SHA384 0x0413
201 #define ECDSA_SECP384R1_SHA384 0x0503
202 #define RSA_PSS_RSAE_SHA256 0x0804
203 #define RSA_PSS_RSAE_SHA384 0x0805
204 #define RSA_PSS_RSAE_SHA512 0x0806
205 #define RSA_PKCS1_SHA256 0x0401
206 #define RSA_PKCS1_SHA384 0x0501
207 #define RSA_PKCS1_SHA512 0x0601
208 #define ED25519 0x0807
209 #define DILITHIUM2 0x0902
210 #define DILITHIUM3 0x0903
211 #define DILITHIUM2_P256 0x09F2
213 // pre-shared Key (PSK) modes
214 #define PSKOK 0x00
215 #define PSKWECDHE 0x01
217 // ticket origin
218 #define TLS_FULL_HANDSHAKE  1
219 #define TLS_EXTERNAL_PSK  2
221 // TLS versions
222 #define TLS1_0 0x0301
223 #define TLS1_2 0x0303
224 #define TLS1_3 0x0304
226 // Extensions
227 #define SERVER_NAME 0x0000
228 #define SUPPORTED_GROUPS 0x000a
229 #define SIG_ALGS 0x000d
230 #define SIG_ALGS_CERT 0x0032
231 #define KEY_SHARE 0x0033
232 #define PSK_MODE 0x002d
233 #define PRESHARED_KEY 0x0029
234 #define TLS_VER 0x002b
235 #define COOKIE 0x002c
236 #define EARLY_DATA 0x002a
237 #define MAX_FRAG_LENGTH 0x0001
238 #define PADDING 0x0015
239 #define APP_PROTOCOL 0x0010
240 #define RECORD_SIZE_LIMIT 0x001c
242 // record types
243 #define HSHAKE 0x16
244 #define APPLICATION 0x17
245 #define ALERT 0x15
246 #define CHANGE_CIPHER 0x14
247 // pseudo record types
248 #define TIMED_OUT 0x01
250 // message types
251 #define CLIENT_HELLO 0x01
252 #define SERVER_HELLO 0x02
253 #define CERTIFICATE 0x0b
254 #define CERT_REQUEST 0x0d
255 #define CERT_VERIFY 0x0f
256 #define FINISHED 0x14
257 #define ENCRYPTED_EXTENSIONS 0x08
258 #define TICKET 0x04
259 #define KEY_UPDATE 0x18
260 #define MESSAGE_HASH 0xFE
261 #define END_OF_EARLY_DATA 0x05
262 // pseudo message types
263 #define HANDSHAKE_RETRY 0x102
265 // Causes of server error - which should generate a client alert
266 #define NOT_TLS1_3 -2
267 #define BAD_CERT_CHAIN -3
268 #define ID_MISMATCH -4
269 #define UNRECOGNIZED_EXT -5
270 #define BAD_HELLO -6
271 #define WRONG_MESSAGE -7
272 #define MISSING_REQUEST_CONTEXT -8
273 #define AUTHENTICATION_FAILURE -9
274 #define BAD_RECORD -10
275 #define BAD_TICKET -11
276 #define NOT_EXPECTED -12
277 #define CA_NOT_FOUND -13
278 #define CERT_OUTOFDATE -14
279 #define MEM_OVERFLOW -15
280 #define FORBIDDEN_EXTENSION -16
281 #define MAX_EXCEEDED -17
282 #define EMPTY_CERT_CHAIN -18
283 #define SELF_SIGNED_CERT -20
284 #define BAD_MESSAGE -23
286 // client alerts
287 #define ILLEGAL_PARAMETER 0x2F
288 #define UNEXPECTED_MESSAGE 0x0A
289 #define DECRYPT_ERROR 0x33
```

```
290 #define BAD_CERTIFICATE 0x2A
291 #define UNSUPPORTED_EXTENSION 0x6E
292 #define UNKNOWN_CA 0x30
293 #define CERTIFICATE_EXPIRED 0x2D
294 #define PROTOCOL_VERSION 0x46
295 #define DECODE_ERROR 0x32
296 #define RECORD_OVERFLOW 0x16
297 #define CLOSE_NOTIFY 0x00
299 #define LOG_OUTPUT_TRUNCATION 256
301 #define TLS13_DISCONNECTED 0
302 #define TLS13_CONNECTED 1
304 // protocol returns..
305 #define TLS_FAILURE 0
306 #define TLS_SUCCESS 1
307 #define TLS_RESUMPTION_REQUIRED 2
308 #define TLS_EARLY_DATA_ACCEPTED 3
312 typedef struct
313 {
314     unsign32 val;
315     int err;
316 } ret;
317
320 typedef struct
321 {
322     bool early_data;
323     bool alpn;
324     bool server_name;
325     bool max_frag_length;
326 } ee_status;
327
330 typedef struct
331 {
332     bool active;
333     char k[TLS_MAX_KEY];
334     char iv[12];
335     octad K;
336     octad IV;
337     unsign32 record;
338     int suite;
339     int taglen;
340 } crypto;
341
344 typedef struct
345 {
346     bool valid;
347     char tick[TLS_MAX_TICKET_SIZE];
348     char nonce[256];
349     char psk[TLS_MAX_HASH];
350     octad TICK;
351     octad NONCE;
352     octad PSK;
353     unsign32 age_obfuscator;
354     unsign32 max_early_data;
355     unsign32 birth;
356     int lifetime;
357     int cipher_suite;
358     int favourite_group;
359     int origin;
360 } ticket;
361
364 typedef struct
365 {
366     char state[TLS_MAX_HASH_STATE];
367     int htype;
368 } unihash;
369
372 typedef struct
373 {
374     int status;
375     int max_record;
376     Socket *sockptr;
377     char id[32];
378     char hostname[TLS_MAX_SERVER_NAME];
379     int cipher_suite;
380     int favourite_group;
381     crypto K_send;
382     crypto K_recv;
383     octad HS;
384     char hs[TLS_MAX_HASH];
385     octad RMS;
386     char rms[TLS_MAX_HASH];
387     octad STS;
388     char sts[TLS_MAX_HASH];
389     octad CTS;
390     char cts[TLS_MAX_HASH];
391     octad IO;
392 #ifndef SHALLOW_STACK
```

```
393     char io[TLS_MAX_IO_SIZE];
394 #endif
395     int ptr;
396     unihash tlshash;
397     ticket T;
398 } TLS_session;
399
400 // IO buffer
401 // xxxxxxxxxxxxxxxxxxxxxxxxxxxxyyyyyyyyyyyyyyyyyyyyyyyyyyyy
402 // -------------ptr---------->----------IO.len----------->
403 //
404 // when ptr becomes equal to IO.len, pull in another record (and maybe decrypt it)
405
406 #endif
```

7.3 C:/Users/mscot/TLS1.3/cpp/include/tls_cert_chain.h File Reference

Process Certificate Chain.

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

Functions

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

7.3.1 Detailed Description

Process Certificate Chain.

Author

Mike Scott

7.3.2 Function Documentation

7.3.2.1 checkServerCertChain()

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

Check Certificate Chain for hostname, and extract public key.

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
SIG	signature (supplied as workspace)

Returns

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

7.3.2.2 getClientPrivateKeyandCertChain()

```
int getClientPrivateKeyandCertChain (
            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

7.4 tls_cert_chain.h

Go to the documentation of this file.

```
1
8 // TLS1.3 Server Certificate Chain Code
9
10 #ifndef TLS_CERT_CHAIN_H
11 #define TLS_CERT_CHAIN_H
12 #include "tls1_3.h"
13 #include "tls_x509.h"
14 #include "tls_sal.h"
15 #include "tls_client_recv.h"
16 #include "tls_logger.h"
17 #include "tls_certs.h"
18
19 using namespace std;
20
29 extern int checkServerCertChain(octad *CERTCHAIN,char *hostname,octad *PUBKEY,octad *SIG);
30
39 extern int getClientPrivateKeyandCertChain(int nccsalgs,int *csigAlgs,octad *PRIVKEY,octad *CERTCHAIN);
40
41 #endif
```

7.5 C:/Users/mscot/TLS1.3/cpp/include/tls_certs.h File Reference

Certificate Authority root certificate store.

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

Variables

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

7.5.1 Detailed Description

Certificate Authority root certificate store.

Author

Mike Scott

7.5.2 Variable Documentation

7.5.2.1 cacerts

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

The Root Certificate store

7.5.2.2 mycert

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

Initial value:

```
=(char *)
"-----BEGIN CERTIFICATE-----\n"
"MIIBKzCB0aADAgECAgEBMAoGCCqGSM49BAMCMB0xGzAZBgNVBAMTEjAxMjM0NjI0QjIwMjYwRDdF\n"
"RTAeFw0yMTExMTgxMTAwMDBaFw0yNjExMTgxMTAwMDBaMB0xGzAZBgNVBAMTEjAxMjM0NjI0QjIw\n"
"MjYwRDdFRTBZMBMGByqGSM49AgEGCCqGSM49AwEHA0IABDOFj/SnArwqM15cZs/bXppfTuAxgMzB\n"
"N3LS48xHSqpLhHlVnvOvWqyhE8v+ZX4Jzlo7Z9LGOG537EeldBeGjYijAjAAMAoGCCqGSM49BAMC\n"
"A0kAMEYCIQC9Oll85YX1+9vZ0t/SHQ3zFH5e7Vc8XtrZ+mTtMc5riwIhAL/SektrG3C0JwII0VV5\n"
"pSR9RRnuwo810km81P4S56/m\n"
"-----END CERTIFICATE-----\n"
```

Client certificate

7.5.2.3 myprivate

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

Client private key

7.6 tls_certs.h

Go to the documentation of this file.

```
1
8 #ifndef TLS_CA_CERTS_H
9 #define TLS_CA_CERTS_H
10
11 #include "tls1_3.h"
12
13 extern const char *myprivate;
14 extern const char *mycert;
15 extern const char *cacerts;
17 #endif
```

7.7 C:/Users/mscot/TLS1.3/cpp/include/tls_client_recv.h File Reference

Process Input received from the Server.

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

Functions

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

  Parse out an octad from a pointer into an octad.
- ret parsebytes (char *e, int len, octad *M, int &ptr)

  Parse out byte array from a pointer into an octad.
- ret parseInt (octad *M, int len, int &ptr)

  Parse out an unsigned integer from a pointer into an octad.
- ret parseoctadptr (octad *E, int len, octad *M, int &ptr)

  Return a pointer to an octad from a pointer into an octad.
- int getServerFragment (TLS_session *session)

  Read a record from the Server, a fragment of a full protocol message.
- ret parseIntorPull (TLS_session *session, int len)

  Parse out an unsigned integer from a pointer into an octad, if necessary pulling in a new fragment.
- ret parseoctadorPull (TLS_session *session, octad *O, int len)

  Parse out an octad from a pointer into an octad, if necessary pulling in a new fragment.
- ret parsebytesorPull (TLS_session *session, char *o, int len)

  Parse out a byte array from a pointer into an octad, if necessary pulling in a new fragment.
- ret parseoctadorPullptrX (TLS_session *session, octad *O, int len)

  Return a pointer to an octad from a pointer into an octad, if necessary pulling in a new fragment.
- bool badResponse (TLS_session *session, ret r)

Process response from server input.

- ret seeWhatsNext (TLS_session *session)

  Identify type of incoming message.
- ret getServerEncryptedExtensions (TLS_session *session, ee_status *enc_ext_expt, ee_status *enc_ext↩ _resp)

  Receive and parse Server Encrypted Extensions.
- ret getServerCertVerify (TLS_session *session, octad *SCVSIG, int &sigalg)

  Get Server proof that he owns the Certificate, by receiving and verifying its signature on transcript hash.
- ret getServerFinished (TLS_session *session, octad *HFIN)

  Get final handshake message from Server, a HMAC on the transcript hash.
- ret getServerHello (TLS_session *session, int &kex, octad *CK, octad *PK, int &pskid)

  Receive and parse initial Server Hello.
- ret getCheckServerCertificateChain (TLS_session *session, octad *PUBKEY, octad *SIG)

  Receive and check certificate chain.
- ret getCertificateRequest (TLS_session *session, int &nalgs, int *sigalgs)

  process a Certificate Request

7.7.1 Detailed Description

Process Input received from the Server.

Author

Mike Scott

7.7.2 Function Documentation

7.7.2.1 badResponse()

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

Process response from server input.

Parameters

session	the TLS1.3 session structure
r	return value to be processed

Returns

true, if its a bad response requiring an abort

7.7.2.2 getCertificateRequest()

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

process a Certificate Request

Parameters

session	the TLS session structure
nalgs	the number of acceptable signature algorithms
sigalgs	an array of nalgs signature algorithms

Returns

response structure

7.7.2.3 getCheckServerCertificateChain()

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

Receive and check certificate chain.

Parameters

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

Returns

response structure

7.7.2.4 getServerCertVerify()

```
ret getServerCertVerify (
            TLS_session * session,
            octad * SCVSIG,
            int & sigalg )
```

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

Parameters

session	the TLS session structure
SCVSIG	the received signature on the transcript hash
sigalg	the type of the received signature

Returns

response structure

7.7.2.5 getServerEncryptedExtensions()

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

Receive and parse Server Encrypted Extensions.

Parameters

session	the TLS session structure
enc_ext_expt	ext structure containing server expectations
enc_ext_resp	ext structure containing server responses

Returns

response structure

7.7.2.6 getServerFinished()

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

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

Parameters

session	the TLS session structure
HFIN	an octad containing HMAC on transcript as calculated by Server

Returns

response structure

7.7.2.7 getServerFragment()

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

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

Parameters

session	the TLS session structure

Returns

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

7.7.2.8 getServerHello()

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

Receive and parse initial Server Hello.

Parameters

session	the TLS session structure
kex	key exchange data
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

response structure

7.7.2.9 parsebytes()

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

Parse out byte array from a pointer into an octad.

Parameters

e	the output byte array copied out from the octad M
len	the expected length of 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

7.7.2.10 parsebytesorPull()

```
ret parsebytesorPull (
            TLS_session * session,
            char * o,
            int len )
```

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

Parameters

session	the TLS session structure
o	the output bytes
len	the expected length of the output

Returns

the actual length of o extracted, and an error flag

7.7.2.11 parseInt()

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

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

Parameters

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

Returns

the integer value, and an error flag

7.7.2.12 parseIntorPull()

```
ret parseIntorPull (
            TLS_session * session,
            int len )
```

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

Parameters

session	the TLS session structure
len	the number of bytes in integer

Returns

the unsigned integer, and an error flag

7.7.2.13 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

7.7.2.14 parseoctadorPull()

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

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

Parameters

session	the TLS session structure
O	the output octad
len	the expected length of the output octad O

Returns

the actual length of O extracted, and an error flag

7.7.2.15 parseoctadorPullptrX()

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

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

Parameters

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

Returns

the actual length of O extracted, and an error flag

7.7.2.16 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

7.7.2.17 seeWhatsNext()

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

Identify type of incoming message.

Parameters

session	the TLS session structure

Returns

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

7.8 tls_client_recv.h

Go to the documentation of this file.

```
1
7 // Process input received from Server
8
9 #ifndef TLS_CLIENT_RECV_H
10 #define TLS_CLIENT_RECV_H
11
12 #include "tls_sal.h"
13 #include "tls1_3.h"
14 #include "tls_sockets.h"
15 #include "tls_keys_calc.h"
16 #include "tls_client_send.h"
17
```

```
26 extern ret parseoctad(octad *E,int len,octad *M,int &ptr);
27
36 extern ret parsebytes(char *e,int len,octad *M,int &ptr);
37
45 extern ret parseInt(octad *M,int len,int &ptr);
46
55 extern ret parseoctadptr(octad *E,int len,octad *M,int &ptr);
56
62 extern int getServerFragment(TLS_session *session);
63
70 extern ret parseIntorPull(TLS_session *session,int len);
71
79 extern ret parseoctadorPull(TLS_session *session,octad *O,int len);
80
81
89 extern ret parsebytesorPull(TLS_session *session,char *o,int len);
90
98 extern ret parseoctadorPullptrX(TLS_session *session,octad *O,int len);
99
106 extern bool badResponse(TLS_session *session,ret r);
107
113 extern ret seeWhatsNext(TLS_session *session);
114
122 extern ret getServerEncryptedExtensions(TLS_session *session,ee_status *enc_ext_expt,ee_status
       *enc_ext_resp);
123
131 extern ret getServerCertVerify(TLS_session *session,octad *SCVSIG,int &sigalg);
132
139 extern ret getServerFinished(TLS_session *session,octad *HFIN);
140
150 extern ret getServerHello(TLS_session *session,/*int &cipher,*/int &kex,octad *CK,octad *PK,int &pskid);
151
159 extern ret getCheckServerCertificateChain(TLS_session *session,octad *PUBKEY,octad *SIG);
160
168 extern ret getCertificateRequest(TLS_session *session,int &nalgs,int *sigalgs);
169
170
171
172 #endif
```

7.9 C:/Users/mscot/TLS1.3/cpp/include/tls_client_send.h File Reference

Process Output to be sent to the Server.

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

Functions

- void sendCCCS (TLS_session *session)

  Send Change Cipher Suite message.
- int addPreSharedKeyExt (octad *EXT, unsign32 age, octad *IDS, int sha)

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

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

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

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

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

Add Key Share extension to under-construction Extensions Octet.

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

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

  Add Maximum Fragment Length extension to under-construction Extensions Octet.
- void addRSLExt (octad *EXT, int size)

  Add Record Size Limit extension to under-construction Extensions Octet.
- void addPSKModesExt (octad *EXT, int mode)

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

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

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

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

  Indicate desire to send Early Data in under-construction Extensions Octet.
- int clientRandom (octad *RN)

  Generate 32-byte random octad.
- int cipherSuites (octad *CS, int ncs, int *ciphers)

  Build a cipher-suites octad from supported ciphers.
- void sendFlushIO (TLS_session *session)

  Flush IO buffer.
- void sendClientMessage (TLS_session *session, int rectype, int version, octad *CM, octad *EXT, bool flush)

  Send a generic client message (as a single record) to the Server.
- void sendBinder (TLS_session *session, octad *BND, bool flush)

  Send a preshared key binder message to the Server.
- void sendClientHello (TLS_session *session, int version, octad *CH, bool already_agreed, octad *EXTENSIONS, int extra, bool resume, bool flush)

  Prepare and send Client Hello message to the Server, appending prepared extensions.
- void sendAlert (TLS_session *session, int type)

  Prepare and send an Alert message to the Server.
- void sendClientFinish (TLS_session *session, octad *CHF)

  Prepare and send a final handshake Verification message to the Server.
- void sendClientCertificateChain (TLS_session *session, octad *CERTCHAIN)

  Prepare and send client certificate message to the Server.
- void sendClientCertVerify (TLS_session *session, int sigAlg, octad *CCVSIG)

  Send client Certificate Verify message to the Server.
- void sendEndOfEarlyData (TLS_session *session)

  Indicate End of Early Data in message to the Server.
- int alert_from_cause (int rtn)

  Maps problem cause to Alert.

7.9.1 Detailed Description

Process Output to be sent to the Server.

Author

Mike Scott

7.9.2 Function Documentation

7.9.2.1 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

7.9.2.2 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

7.9.2.3 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

7.9.2.4 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

7.9.2.5 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

7.9.2.6 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

7.9.2.7 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

7.9.2.8 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

7.9.2.9 addRSLExt()

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

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

Parameters

EXT	the extensions octad which is being built
size	the demanded maximum fragment size

7.9.2.10 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

7.9.2.11 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

7.9.2.12 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

7.9.2.13 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

7.9.2.14 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

7.9.2.15 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

7.9.2.16 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

7.9.2.17 clientRandom()

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

Generate 32-byte random octad.

Parameters

RN	the output 32-byte octad

Returns

length of output octad

7.9.2.18 sendAlert()

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

Prepare and send an Alert message to the Server.

Parameters

session	the TLS session structure
type	the type of the Alert

7.9.2.19 sendBinder()

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

Send a preshared key binder message to the Server.

Parameters

session	the TLS session structure
BND	binding HMAC of truncated transcript hash
flush	transmit immediately if true

7.9.2.20 sendCCCS()

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

Send Change Cipher Suite message.

Parameters

session	the TLS session structure

7.9.2.21 sendClientCertificateChain()

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

Prepare and send client certificate message to the Server.

Parameters

session	the TLS session structure
CERTCHAIN	the client certificate chain

7.9.2.22 sendClientCertVerify()

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

Send client Certificate Verify message to the Server.

Parameters

session	the TLS session structure
sigAlg	the client's digital signature algorithm
CCVSIG	the client's signature

7.9.2.23 sendClientFinish()

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

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

Parameters

session	the TLS session structure
CHF	the client verify data HMAC

7.9.2.24 sendClientHello()

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

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

Parameters

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

7.9.2.25 sendClientMessage()

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

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

Parameters

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

7.9.2.26 sendEndOfEarlyData()

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

Indicate End of Early Data in message to the Server.

Parameters

session	the TLS session structure

7.9.2.27 sendFlushIO()

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

Flush IO buffer.

Parameters

session	the TLS session structure

7.10 tls_client_send.h

Go to the documentation of this file.

```
1
8 // Process output sent to Server
9 #ifndef TLS_CLIENT_SEND_H
10 #define TLS_CLIENT_SEND_H
11
12 #include "tls_sal.h"
13 #include "tls1_3.h"
14 #include "tls_sockets.h"
15 #include "tls_keys_calc.h"
16
21 extern void sendCCCS(TLS_session *session);
22
31 extern int addPreSharedKeyExt(octad *EXT,unsign32 age,octad *IDS,int sha);
32
38 extern void addServerNameExt(octad *EXT,char *servername);
39
46 extern void addSupportedGroupsExt(octad *EXT,int nsg,int *supportedGroups);
47
48
55 extern void addSigAlgsExt(octad *EXT,int nsa,int *sigAlgs);
56
63 extern void addSigAlgsCertExt(octad *EXT,int nsac,int *sigAlgsCert);
64
65
72 extern void addKeyShareExt(octad *EXT,int alg,octad *PK);
73
74
80 extern void addALPNExt(octad *EXT,octad *AP);
81
82
88 extern void addMFLExt(octad *EXT,int mode);
89
95 extern void addRSLExt(octad *EXT,int size);
96
102 extern void addPSKModesExt(octad *EXT,int mode);
103
109 extern void addVersionExt(octad *EXT,int version);
110
116 extern void addPadding(octad *EXT,int n);
117
123 extern void addCookieExt(octad *EXT,octad *CK);
124
129 extern void addEarlyDataExt(octad *EXT);
130
136 extern int clientRandom(octad *RN);
137
145 extern int cipherSuites(octad *CS,int ncs,int *ciphers);
146
151 extern void sendFlushIO(TLS_session *session);
152
162 extern void sendClientMessage(TLS_session *session,int rectype,int version,octad *CM,octad *EXT,bool
    flush);
163
170 extern void sendBinder(TLS_session *session,octad *BND,bool flush);
171
183 extern void sendClientHello(TLS_session *session,int version,octad *CH,bool already_agreed,octad
    *EXTENSIONS,int extra,bool resume,bool flush);
184
190 extern void sendAlert(TLS_session *session,int type);
191
```

```
197 extern void sendClientFinish(TLS_session *session,octad *CHF);
198
204 extern void sendClientCertificateChain(TLS_session *session,octad *CERTCHAIN);
205
212 extern void sendClientCertVerify(TLS_session *session, int sigAlg, octad *CCVSIG);
213
214
219 extern void sendEndOfEarlyData(TLS_session *session);
220
226 extern int alert_from_cause(int rtn);
227 #endif
```

7.11 C:/Users/mscot/TLS1.3/cpp/include/tls_keys_calc.h File Reference

TLS 1.3 crypto support functions.

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

Functions

- void initTranscriptHash (TLS_session *session)

  Initialise Transcript hash.
- void runningHash (TLS_session *session, octad *O)

  Accumulate octad into ongoing hashing.
- void runningHashIO (TLS_session *session)

  Accumulate transcript hash from IO buffer.
- void rewindIO (TLS_session *session)

  rewind the IO buffer
- void runningHashIOrewind (TLS_session *session)

  Accumulate transcript hash and from IO buffer, and rewind IO buffer.
- void transcriptHash (TLS_session *session, octad *O)

  Output current hash value.
- void runningSyntheticHash (TLS_session *session, octad *O, octad *E)

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

  Initiate a Crypto Context.
- void updateCryptoContext (crypto *C, octad *K, octad *IV)

  Build a Crypto Context.
- void incrementCryptoContext (crypto *C)

  Increment a Crypto Context for the next record, updating IV.
- void createCryptoContext (int cipher, octad *TS, crypto *context)

  Create a crypto context from an input raw Secret and an agreed cipher_suite.
- void createSendCryptoContext (TLS_session *session, octad *TS)

  Build a crypto context for transmission from an input raw Secret and an agreed cipher_suite.
- void createRecvCryptoContext (TLS_session *session, octad *TS)

  Build a crypto context for reception from an input raw Secret and an agreed cipher_suite.
- void recoverPSK (TLS_session *)

  Recover pre-shared key from the Resumption Master Secret and store with ticket.
- void deriveEarlySecrets (int htype, octad *PSK, octad *ES, octad *BKE, octad *BKR)

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

- void deriveLaterSecrets (int htype, octad *H, octad *ES, octad *CETS, octad *EEMS)

  Extract more secrets from Early Secret.
- void deriveHandshakeSecrets (TLS_session *session, octad *SS, octad *ES, octad *H)

  Extract Handshake Secret from Shared Secret and Early Secret. Use Handshake Secret to extract Client and Server Handshake Traffic secrets.
- void deriveApplicationSecrets (TLS_session *session, octad *SFH, octad *CFH, octad *EMS)

  Extract Application Secret from Handshake Secret and Early Secret. Use Handshake Secret to extract Client and Server Application Traffic secrets.
- void deriveUpdatedKeys (crypto *context, octad *TS)

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

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

  Create handshake verification data for Client to send to Server from client traffic secret and a transcript hash.
- bool checkServerCertVerifier (int sigalg, octad *SCVSIG, octad *H, octad *CERTPK)

  verify Server's signature on protocol transcript
- void createClientCertVerifier (int sigAlg, octad *H, octad *KEY, octad *CCVSIG)

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

7.11.1 Detailed Description

TLS 1.3 crypto support functions.

Author

Mike Scott

7.11.2 Function Documentation

7.11.2.1 checkServerCertVerifier()

```
bool checkServerCertVerifier (
            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

7.11.2.2 checkVeriferData()

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

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

Parameters

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

7.11.2.3 createClientCertVerifier()

```
void createClientCertVerifier (
            int sigAlg,
            octad * H,
            octad * KEY,
            octad * CCVSIG )
```

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

Parameters

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

7.11.2.4 createCryptoContext()

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

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

Parameters

cipher	the chosen cipher site
TS	the input raw secret
context	the output crypto conetext

7.11.2.5 createRecvCryptoContext()

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

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

Parameters

session	TLS session structure
TS	the input raw secret

7.11.2.6 createSendCryptoContext()

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

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

Parameters

session	TLS session structure
TS	the input raw secret

7.11.2.7 deriveApplicationSecrets()

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

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

Parameters

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

7.11.2.8 deriveEarlySecrets()

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

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

Parameters

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)

7.11.2.9 deriveHandshakeSecrets()

```
void deriveHandshakeSecrets (
            TLS_session * session,
            octad * SS,
            octad * ES,
            octad * H )
```

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

Parameters

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

7.11.2.10 deriveLaterSecrets()

```
void deriveLaterSecrets (
            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)

7.11.2.11 deriveUpdatedKeys()

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

Perform a Key Update on a crypto context.

Parameters

context	an AEAD encryption context
TS	the updated Traffic secret

7.11.2.12 deriveVeriferData()

```
void deriveVeriferData (
            int htype,
```

```
            octad * SF,
            octad * CTS,
            octad * H )
```

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

Parameters

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

7.11.2.13 incrementCryptoContext()

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

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

Parameters

C	an AEAD encryption context

7.11.2.14 initCryptoContext()

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

Initiate a Crypto Context.

Parameters

C	an AEAD encryption context

7.11.2.15 initTranscriptHash()

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

Initialise Transcript hash.

Parameters

session	the TLS session structure

7.11.2.16 recoverPSK()

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

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

Parameters

session	the TLS session structure

7.11.2.17 rewindIO()

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

rewind the IO buffer

Parameters

session	the TLS session structure

7.11.2.18 runningHash()

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

Accumulate octad into ongoing hashing.

Parameters

session	the TLS session structure
O	an octad to be included in hash

7.11.2.19 runningHashIO()

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

Accumulate transcript hash from IO buffer.

Parameters

session	the TLS session structure

7.11.2.20 runningHashIOrewind()

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

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

Parameters

session	the TLS session structure

7.11.2.21 runningSyntheticHash()

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

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

Parameters

session	the TLS session structure
O	an octad containing clientHello
E	an octad containing clientHello extensions

7.11.2.22 transcriptHash()

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

Output current hash value.

Parameters

session	the TLS session structure
O	an output octad containing current hash

7.11.2.23 updateCryptoContext()

```
void updateCryptoContext (
             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

7.12 tls_keys_calc.h

Go to the documentation of this file.

```
1
8 // TLS1.3 crypto support functions
9 #ifndef TLS_KEYS_CALC_H
10 #define TLS_KEYS_CALC_H
11
12 #include "tls1_3.h"
13 #include "tls_sal.h"
14 #include "tls_client_recv.h"
15
16 // transcript hash support
21 extern void initTranscriptHash(TLS_session *session);
22
29 extern void runningHash(TLS_session *session,octad *O);
30
36 extern void runningHashIO(TLS_session *session);
37
43 extern void rewindIO(TLS_session *session);
44
45
51 extern void runningHashIOrewind(TLS_session *session);
52
53
59 extern void transcriptHash(TLS_session *session,octad *O);
60
68 extern void runningSyntheticHash(TLS_session *session,octad *O,octad *E);
69
74 extern void initCryptoContext(crypto *C);
75
82 extern void updateCryptoContext(crypto *C,octad *K,octad *IV);
83
88 extern void incrementCryptoContext(crypto *C);
89
90
97 extern void createCryptoContext(int cipher,octad *TS,crypto *context);
98
104 extern void createSendCryptoContext(TLS_session *session,octad *TS);
105
111 extern void createRecvCryptoContext(TLS_session *session,octad *TS);
```

```
112
117 extern void recoverPSK(TLS_session *);
118
127 extern void deriveEarlySecrets(int htype,octad *PSK,octad *ES,octad *BKE,octad *BKR);
128
137 extern void deriveLaterSecrets(int htype,octad *H,octad *ES,octad *CETS,octad *EEMS);
138
146 extern void deriveHandshakeSecrets(TLS_session *session,octad *SS,octad *ES, octad *H);
147
155 extern void deriveApplicationSecrets(TLS_session *session,octad *SFH,octad *CFH,octad *EMS);
156
162 extern void deriveUpdatedKeys(crypto *context,octad *TS);
163
172 extern bool checkVeriferData(int htype,octad *SF,octad *STS,octad *H);
173
181 extern void deriveVeriferData(int htype,octad *SF,octad *CTS,octad *H);
182
191 extern bool checkServerCertVerifier(int sigalg,octad *SCVSIG,octad *H,octad *CERTPK);
192
200 extern void createClientCertVerifier(int sigAlg,octad *H,octad *KEY,octad *CCVSIG);
201
202 #endif
```

7.13 C:/Users/mscot/TLS1.3/cpp/include/tls_logger.h File Reference

TLS 1.3 logging.

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

Functions

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

7.13.1 Detailed Description

TLS 1.3 logging.

Author

Mike Scott

7.13.2 Function Documentation

7.13.2.1 log()

```
void log (
            int logit,
            char * preamble,
            char * string,
            unsign32 info,
            octad * O )
```

basic logging function

Parameters

logit	logging level
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)

7.13.2.2 logAlert()

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

log Server Alert

Parameters

detail	the server's alert code

7.13.2.3 logCert()

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

logging a Certificate in standard base 64 format

Parameters

CERT	the certificate to be logged

7.13.2.4 logCertDetails()

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

logging Certificate details

Parameters

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

7.13.2.5 logCipherSuite()

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

log Cipher Suite

Parameters

cipher_suite	the Cipher Suite to be logged

7.13.2.6 logEncExt()

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

logging server extended extensions responses vs expectations

Parameters

e	structure containing server expectations
r	structure containing server responses

7.13.2.7 logKeyExchange()

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

log Key Exchange Group

Parameters

kex	the Key Exchange Group to be logged

7.13.2.8 logServerHello()

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

logging the Server hello

Parameters

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

7.13.2.9 logServerResponse()

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

log client processing of a Server response

Parameters

r	the Server response

7.13.2.10 logSigAlg()

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

log Signature Algorithm

Parameters

sigAlg	the Signature Algorithm to be logged

7.13.2.11 logTicket()

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

logging a resumption ticket

Parameters

T	a resumption ticket

7.13.2.12 myprintf()

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

internal printf function - all output funnels through this function

Parameters

s	a string to be output

7.14 tls_logger.h

Go to the documentation of this file.

```
1
7 // Log protocol progress
8 #ifndef TLS_LOGGER_H
9 #define TLS_LOGGER_H
10
11 #include <string.h>
12 #include "tls1_3.h"
13 #include "tls_x509.h"
14
19 extern void myprintf(char *s);
20
29 extern void log(int logit,char *preamble,char *string,unsign32 info,octad *O);
30
38 extern void logServerHello(int cipher_suite,int pskid,octad *PK,octad *CK);
39
44 extern void logTicket(ticket *T);
45
51 extern void logEncExt(ee_status *e,ee_status *r);
52
57 extern void logCert(octad *CERT);
58
68 extern void logCertDetails(octad *PUBKEY,pktype pk,octad *SIG,pktype sg,octad *ISSUER,octad *SUBJECT);
69
74 extern void logServerResponse(ret r);
75
80 extern void logAlert(int detail);
81
86 extern void logCipherSuite(int cipher_suite);
87
92 extern void logKeyExchange(int kex);
93
98 extern void logSigAlg(int sigAlg);
99
100 #endif
```

7.15 C:/Users/mscot/TLS1.3/cpp/include/tls_octads.h File Reference

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

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

Classes

- struct octad

  Safe representation of an octad.

Functions

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

7.15.1 Detailed Description

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

Author

Mike Scott

7.15.2 Function Documentation

7.15.2.1 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

7.15.2.2 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

7.15.2.3 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

7.15.2.4 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

7.15.2.5 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

7.15.2.6 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

7.15.2.7 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

7.15.2.8 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

7.15.2.9 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

7.15.2.10 OCT_kill()

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

Wipe clean an octad.

Parameters

O	octad to be cleared

7.15.2.11 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

7.15.2.12 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

7.15.2.13 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

7.15.2.14 OCT_reverse()

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

Reverse bytes in an octad.

Parameters

O	octad to be reversed

7.15.2.15 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

7.15.2.16 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

7.16 tls_octads.h

Go to the documentation of this file.

```
1
8 #ifndef TLS_OCTADS_H
9 #define TLS_OCTADS_H
10
11 // An octad - "a group or set of eight" - Oxford dictionary
12
13 #include <stddef.h>
14
18 typedef struct
19 {
20     int len;
21     int max;
22     char *val;
23 } octad;
24
32 extern void OCT_append_int(octad *O, unsigned int x, int len);
33
39 extern void OCT_append_octad(octad *O, octad *P);
40
47 extern bool OCT_compare(octad *O, octad *P);
48
56 extern void OCT_shift_left(octad *O, int n);
57
62 extern void OCT_kill(octad *O);
63
69 extern void OCT_from_hex(octad *O, char *src);
70
76 extern void OCT_append_string(octad *O, char *s);
77
84 extern void OCT_append_byte(octad *O, int b, int n);
85
92 extern void OCT_append_bytes(octad *O, char *s, int n);
93
100 extern void OCT_from_base64(octad *O, char *b);
101
106 extern void OCT_reverse(octad *O);
107
113 extern void OCT_truncate(octad *O,int n);
114
121 extern void OCT_copy(octad *O, octad *P);
122
130 extern bool OCT_output_hex(octad *O,int max,char *s);
131
139 extern bool OCT_output_string(octad *O,int max,char *s);
140
148 extern void OCT_output_base64(octad *O,int max,char *s);
149 #endif
```

7.17 C:/Users/mscot/TLS1.3/cpp/include/tls_protocol.h File Reference

TLS 1.3 main client-side protocol functions.

```
#include "tls_keys_calc.h"
#include "tls_cert_chain.h"
#include "tls_client_recv.h"
#include "tls_client_send.h"
#include "tls_tickets.h"
#include "tls_logger.h"
```

Functions

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

  initialise a TLS 1.3 session structure
- void TLS13_end (TLS_session *session)

  terminate a session structure

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

  TLS 1.3 forge connection.
- void TLS13_send (TLS_session *session, octad *DATA)

  TLS 1.3 send data.
- int TLS13_recv (TLS_session *session, octad *DATA)

  TLS 1.3 receive data.
- void TLS13_clean (TLS_session *session)

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

7.17.1 Detailed Description

TLS 1.3 main client-side protocol functions.

Author

Mike Scott

7.17.2 Function Documentation

7.17.2.1 TLS13_clean()

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

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

Parameters

session	an initialised TLS session structure

7.17.2.2 TLS13_connect()

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

TLS 1.3 forge connection.

Parameters

session	an initialised TLS session structure
EARLY	some early data to be transmitted

Returns

false for failure, true for success

7.17.2.3 TLS13_end()

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

terminate a session structure

Parameters

session	the session structure

7.17.2.4 TLS13_recv()

```
int TLS13_recv (
            TLS_session * session,
            octad * DATA )
```

TLS 1.3 receive data.

Parameters

session	an initialised TLS session structure
DATA	that has been received

Returns

0 for failure, otherwise success

7.17.2.5 TLS13_send()

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

TLS 1.3 send data.

Parameters

session	an initialised TLS session structure
DATA	some data to be transmitted

7.17.2.6 TLS13_start()

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

initialise a TLS 1.3 session structure

Parameters

client	the socket connection to the Server
hostname	the host name (URL) of the server

Returns

an initialised TLS1.3 session structure

7.18 tls_protocol.h

Go to the documentation of this file.

```
1
7 // Main TLS 1.3 Protocol
8
9 #ifndef TLS_PROTOCOL_H
10 #define TLS_PROTOCOL_H
11
12 #include "tls_keys_calc.h"
13 #include "tls_cert_chain.h"
14 #include "tls_client_recv.h"
15 #include "tls_client_send.h"
16 #include "tls_tickets.h"
17 #include "tls_logger.h"
18
25 extern TLS_session TLS13_start(Socket *client,char *hostname);
26
31 extern void TLS13_end(TLS_session *session);
32
39 extern bool TLS13_connect(TLS_session *session,octad *EARLY);
40
46 extern void TLS13_send(TLS_session *session,octad *DATA);
47
54 extern int TLS13_recv(TLS_session *session,octad *DATA);
55
60 extern void TLS13_clean(TLS_session *session);
61 #endif
```

7.19 C:/Users/mscot/TLS1.3/cpp/include/tls_sal.h File Reference

Security Abstraction Layer for TLS.

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

Functions

- char * SAL_name ()

  Return name of SAL provider.
- int SAL_ciphers (int *ciphers)

  Return supported ciphers.
- int SAL_groups (int *groups)

  Return supported groups in preferred order.
- int SAL_sigs (int *sigAlgs)

  Return supported TLS signature algorithms in preferred order.
- int SAL_sigCerts (int *sigAlgsCert)

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

  Initialise library for use.
- void SAL_endLib ()

  finish use of library
- int SAL_hashType (int cipher_suite)

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

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

  return key length associated with a cipher suite
- int SAL_aeadTaglen (int cipher_suite)

  return authentication tag length associated with a cipher suite
- int SAL_randomByte ()

  get a random byte
- void SAL_randomOctad (int len, octad *R)

  get a random octad
- void SAL_hkdfExtract (int sha, octad *PRK, octad *SALT, octad *IKM)

  HKDF Extract function.
- void SAL_hkdfExpand (int htype, int olen, octad *OKM, octad *PRK, octad *INFO)

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

  simple HMAC function
- void SAL_hashNull (int sha, octad *H)

  simple HASH of nothing function
- void SAL_hashInit (int hlen, unihash *h)

  Initiate Hashing context.
- void SAL_hashProcessArray (unihash *h, char *b, int len)

  Hash process an array of bytes.
- int SAL_hashOutput (unihash *h, char *d)

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

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

  AEAD decryption.
- void SAL_generateKeyPair (int group, octad *SK, octad *PK)

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

  generate a Diffie-Hellman shared secret
- bool SAL_tlsSignatureVerify (int sigAlg, octad *TRANS, octad *SIG, octad *PUBKEY)

  Verify a generic TLS signature.
- void SAL_tlsSignature (int sigAlg, octad *KEY, octad *TRANS, octad *SIG)

  Apply a generic TLS transcript signature.

7.19.1 Detailed Description

Security Abstraction Layer for TLS.

Author

Mike Scott

7.19.2 Function Documentation

7.19.2.1 SAL_aeadDecrypt()

```
bool SAL_aeadDecrypt (
            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

false if tag is wrong, else true

7.19.2.2 SAL_aeadEncrypt()

```
void SAL_aeadEncrypt (
            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
hdr	the header bytes
ptlen	the plaintext length
pt	the input plaintext and output ciphertext
TAG	the output authentication tag

7.19.2.3 SAL_aeadKeylen()

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

return key length associated with a cipher suite

Parameters

cipher_suite	a TLS cipher suite

Returns

key length

7.19.2.4 SAL_aeadTaglen()

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

return authentication tag length associated with a cipher suite

Parameters

cipher_suite	a TLS cipher suite

Returns

tag length

7.19.2.5 SAL_ciphers()

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

Return supported ciphers.

Parameters

ciphers	array of supported ciphers in preferred order

Returns

number of supported ciphers

7.19.2.6 SAL_endLib()

```
void SAL_endLib ( )
```

finish use of library

7.19.2.7 SAL_generateKeyPair()

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

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

Parameters

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

7.19.2.8 SAL_generateSharedSecret()

```
void SAL_generateSharedSecret (
            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

7.19.2.9 SAL_groups()

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

Return supported groups in preferred order.

Parameters

groups	array of supported groups

Returns

number of supported groups

7.19.2.10 SAL_hashInit()

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

Initiate Hashing context.

Parameters

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

7.19.2.11 SAL_hashLen()

```
int 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

7.19.2.12 SAL_hashNull()

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

simple HASH of nothing function

Parameters

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

7.19.2.13 SAL_hashOutput()

```
int SAL_hashOutput (
            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

7.19.2.14 SAL_hashProcessArray()

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

Hash process an array of bytes.

Parameters

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

7.19.2.15 SAL_hashType()

```
int 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

7.19.2.16 SAL_hkdfExpand()

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

Special HKDF Expand function (for TLS)

Parameters

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

7.19.2.17 SAL_hkdfExtract()

```
void SAL_hkdfExtract (
            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

7.19.2.18 SAL_hmac()

```
void SAL_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

7.19.2.19 SAL_initLib()

```
bool SAL_initLib ( )
```

Initialise library for use.

Returns

return true if successful, else false

7.19.2.20 SAL_name()

```
char * SAL_name ( )
```

Return name of SAL provider.

Returns

name of SAL provider

7.19.2.21 SAL_randomByte()

```
int SAL_randomByte ( )
```

get a random byte

Returns

a random byte

7.19.2.22 SAL_randomOctad()

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

get a random octad

Parameters

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

7.19.2.23 SAL_sigCerts()

```
int 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

7.19.2.24 SAL_sigs()

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

Return supported TLS signature algorithms in preferred order.

Parameters

sigAlgs	array of supported signature algorithms

Returns

number of supported groups

7.19.2.25 SAL_tlsSignature()

```
void SAL_tlsSignature (
            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

7.19.2.26 SAL_tlsSignatureVerify()

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

Verify a generic TLS signature.

Parameters

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

Returns

true if signature is valid, else false

7.20 tls_sal.h

Go to the documentation of this file.

```
1
7 // Process input received from Server
8
9 #ifndef TLS_SAL_H
10 #define TLS_SAL_H
11
12 // Use MIRACL core library
13
14 #include "tls1_3.h"
15
20 extern char *SAL_name();
21
27 extern int SAL_ciphers(int *ciphers);
28
34 extern int SAL_groups(int *groups);
35
41 extern int SAL_sigs(int *sigAlgs);
42
48 extern int SAL_sigCerts(int *sigAlgsCert);
49
54 extern bool SAL_initLib();
55
59 extern void SAL_endLib();
60
66 extern int SAL_hashType(int cipher_suite);
```

```
67
73 extern int SAL_hashLen(int hash_type);
74
80 int SAL_aeadKeylen(int cipher_suite);
81
87 int SAL_aeadTaglen(int cipher_suite);
88
93 extern int SAL_randomByte();
94
100 extern void SAL_randomOctad(int len, octad *R);
101
109 extern void SAL_hkdfExtract(int sha,octad *PRK,octad *SALT,octad *IKM);
110
119 extern void SAL_hkdfExpand(int htype, int olen, octad *OKM,octad *PRK, octad *INFO);
120
128 extern void SAL_hmac(int htype,octad *T,octad *K,octad *M);
129
135 extern void SAL_hashNull(int sha,octad *H);
136
137 // hash functions
138
144 extern void SAL_hashInit(int hlen,unihash *h);
145
152 extern void SAL_hashProcessArray(unihash *h,char *b,int len);
153
154
161 extern int SAL_hashOutput(unihash *h,char *d);
162
172 extern void SAL_aeadEncrypt(crypto *send,int hdrlen,char *hdr,int ptlen,char *pt,octad *TAG);
173
184 extern bool SAL_aeadDecrypt(crypto *recv,int hdrlen,char *hdr,int ctlen,char *ct,octad *TAG);
185
192 extern void SAL_generateKeyPair(int group,octad *SK,octad *PK);
193
201 extern void SAL_generateSharedSecret(int group,octad *SK,octad *PK,octad *SS);
202
203
212 extern bool SAL_tlsSignatureVerify(int sigAlg,octad *TRANS,octad *SIG,octad *PUBKEY);
213
221 extern void SAL_tlsSignature(int sigAlg,octad *KEY,octad *TRANS,octad *SIG);
222
223
224 #endif
```

7.21 C:/Users/mscot/TLS1.3/cpp/include/tls_sockets.h File Reference

set up sockets for reading and writing

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

Classes

- class Socket

  Socket instance.

Functions

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

7.21.1 Detailed Description

set up sockets for reading and writing

Author

Mike Scott

7.21.2 Function Documentation

7.21.2.1 getByte()

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

receive a single byte from a socket

Parameters

client	the socket connection to the Server

Returns

a byte

7.21.2.2 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

7.21.2.3 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

7.21.2.4 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

7.21.2.5 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

7.21.2.6 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

7.21.2.7 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

7.21.2.8 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

7.21.2.9 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

7.22 tls_sockets.h

Go to the documentation of this file.

```
1
8 // Set up and read/write sockets
```

```
9
10 #ifndef TLS_SOCKETS_H
11 #define TLS_SOCKETS_H
12
13 //#define TLS_ARDUINO            /**< Define for Arduino-based implementation */
14
15 #include <string.h>
16 #include "tls_octads.h"
17
18 #ifdef TLS_ARDUINO
19 #include "tls_wifi.h"
20 #else
21 #include <time.h>
22 #include <unistd.h>
23 #include <stdio.h>
24 #include <sys/socket.h>
25 #include <arpa/inet.h>
26 #include <stdlib.h>
27 #include <netinet/in.h>
28 #include <netdb.h>
29 #include <netinet/in.h>
30 #include <sys/un.h>
31 #endif
32
33 #ifndef TLS_ARDUINO
34
42 extern int setclientsock(int port, char *ip, int toms);
43
50 extern int getIPaddress(char *ip,char *hostname);
51
52 // Simple socket class, mimics Arduino
55 class Socket
56 {
57     int sock;
58     int toms;
59     bool is_af_unix;
61 private:
62     Socket(bool is_af_unix) {
63         this->sock = 0;
64         this->toms = 5000;
65         this->is_af_unix = is_af_unix;
66     }
67
68     static int afunix_setclientsock(const char *const socket_path)
69     {
70         int sock;
71         struct sockaddr_un serv_addr;
72         if ((sock = socket(AF_UNIX, SOCK_STREAM, 0)) < 0)
73             return -2;
74
75         serv_addr.sun_family = AF_UNIX;
76         strcpy(serv_addr.sun_path, socket_path);
77
78         if (::connect(sock, (struct sockaddr *)&serv_addr, sizeof(serv_addr)) < 0)
79             return -1;
80
81         return sock;
82     }
83
84
85 public:
86     bool connect(char *host,int port) {
87         if(!this->is_af_unix) {
88             char ip[40];
89             sock = 0;
90             if (!getIPaddress(ip, host))
91                 return false;
92             sock = setclientsock(port, ip, toms);
93             if (sock <= 0)
94                 return false;
95             return true;
96         } else {
97             bool connected = true;
98             sock = afunix_setclientsock(host);
99             if (sock <= 0) {
100                 connected = false;
101              }
102              return connected;
103         }
104     }
105
106     static Socket InetSocket() {
107         return Socket(false);
108     }
109
110     static Socket UnixSocket() {
111         return Socket(true);
```

```
112     }
113
114     void setTimeout(int to) {toms=to;}
115     int write(char *buf,int len) {return ::send(sock,buf,len,0);}
116     int read(char *buf,int len) {return ::recv(sock,buf,len,0);}
117     void stop() {::close(sock);}
118
119     ~Socket() {::close(sock);}
120 };
121 #else
122
128 extern void clearsoc(Socket &client,octad *IO);
129
130 #endif
131
137 extern void sendOctad(Socket *client,octad *B);
138
144 extern void sendLen(Socket *client,int len);
145
153 extern int getBytes(Socket *client,char *b,int expected);
154
160 extern int getInt16(Socket *client);
161
167 extern int getInt24(Socket *client);
168
174 extern int getByte(Socket *client);
175
183 extern int getOctad(Socket *client,octad *B,int expected);
184
185 #endif
```

7.23 C:/Users/mscot/TLS1.3/cpp/include/tls_tickets.h File Reference

TLS 1.3 process resumption tickets.

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

Functions

- unsigned long millis ()
  read milliseconds from a stop-watch
- int parseTicket (octad *TICK, unsign32 birth, ticket *T)
  parse a received ticket octad into a ticket structure
- void initTicketContext (ticket *T)
  initialize a ticket structure
- void endTicketContext (ticket *T)
  terminate a ticket structure
- bool ticket_still_good (ticket *T)
  Check that a ticket is still good, and not out-of-date.

7.23.1 Detailed Description

TLS 1.3 process resumption tickets.

Author

Mike Scott

7.23.2 Function Documentation

7.23.2.1 endTicketContext()

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

terminate a ticket structure

Parameters

T	the ticket structure

7.23.2.2 initTicketContext()

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

initialize a ticket structure

Parameters

T	the ticket structure

7.23.2.3 millis()

```
unsigned long millis ( )
```

read milliseconds from a stop-watch

Returns

milliseconds read from stop-watch

7.23.2.4 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

7.23.2.5 ticket_still_good()

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

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

Parameters

T	the ticket structure

Returns

true if ticket is still good

7.24 tls_tickets.h

Go to the documentation of this file.

```
1
7 // Process Resumption Tickets
8
9 #ifndef TLS_TICKETS_H
10 #define TLS_TICKETS_H
11
12 #include "tls1_3.h"
13 #include "tls_client_recv.h"
14
19 extern unsigned long millis();
20
28 extern int parseTicket(octad *TICK,unsign32 birth,ticket *T);
29
34 extern void initTicketContext(ticket *T);
35
40 extern void endTicketContext(ticket *T);
41
47 extern bool ticket_still_good(ticket *T);
48
49 #endif
```

7.25 C:/Users/mscot/TLS1.3/cpp/include/tls_wifi.h File Reference

define Socket structure depending on processor context

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

7.25.1 Detailed Description

define Socket structure depending on processor context

Author

Mike Scott

7.26 tls_wifi.h

Go to the documentation of this file.

```
1
7 // Set up WiFi environment for Arduino boards
8
9 #ifndef TLS_WIFI_H
10 #define TLS_WIFI_H
11
12 #include "tls1_3.h"
13
14 #ifdef TLS_ARDUINO
15
16 #ifdef FISHINO_PIRANHA
17 // Fishino Piranha board
18 #define PARTICULAR_BOARD
19 #include <Fishino.h>
20 #include <SPI.h>
21 typedef FishinoClient Socket;
22 #define WiFi Fishino
23 #define FISHINO
24
25 #endif
26
27 #ifdef ESP32
28 // ESP32 board
29 #define PARTICULAR_BOARD
30 #include <WiFi.h>
31 typedef WiFiClient Socket;
32
33 #endif
34
35 #ifndef PARTICULAR_BOARD
36 // any other board
37 #include <WiFiNINA.h>
38 typedef WiFiClient Socket;
39
40 #endif
41 #endif
42
43 #endif
```

7.27 C:/Users/mscot/TLS1.3/cpp/include/tls_x509.h File Reference

X509 function Header File.

Classes

- struct pktype

  Public key type.

Macros

- #define X509_ECC 1
- #define X509_RSA 2
- #define X509_ECD 3
- #define X509_PQ 4
- #define X509_HY 5
- #define X509_H256 2
- #define X509_H384 3
- #define X509_H512 4
- #define USE_NIST256 0
- #define USE_C25519 1
- #define USE_NIST384 10
- #define USE_NIST521 12

Functions

- void ecdsa_sig_encode (octad *c)

  in-place ECDSA signature encoding
- int ecdsa_sig_decode (octad *c)

  in-place ECDSA signature decoding
- pktype X509_extract_private_key (octad *c, octad *pk)

  Extract private key.
- pktype X509_extract_cert_sig (octad *c, octad *s)

  Extract certificate signature.
- int X509_extract_cert (octad *sc, octad *c)
- pktype X509_extract_public_key (octad *c, octad *k)
- int X509_find_issuer (octad *c)
- int X509_find_validity (octad *c)
- int X509_find_subject (octad *c)
- int X509_self_signed (octad *c)
- int X509_find_entity_property (octad *c, octad *S, int s, int *f)
- int X509_find_start_date (octad *c, int s)
- int X509_find_expiry_date (octad *c, int s)
- int X509_find_extensions (octad *c)
- int X509_find_extension (octad *c, octad *S, int s, int *f)
- int X509_find_alt_name (octad *c, int s, char *name)

Variables

- octad X509_CN
- octad X509_ON
- octad X509_EN
- octad X509_LN
- octad X509_UN
- octad X509_MN
- octad X509_SN
- octad X509_AN
- octad X509_KU
- octad X509_BC

7.27.1 Detailed Description

X509 function Header File.

Author

Mike Scott

defines structures declares functions

7.27.2 Macro Definition Documentation

7.27.2.1 USE_C25519

```
#define USE_C25519 1
```

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

7.27.2.2 USE_NIST256

```
#define USE_NIST256 0
```

For the NIST 256-bit standard curve - WEIERSTRASS only

7.27.2.3 USE_NIST384

```
#define USE_NIST384 10
```

For the NIST 384-bit standard curve - WEIERSTRASS only

7.27.2.4 USE_NIST521

```
#define USE_NIST521 12
```

For the NIST 521-bit standard curve - WEIERSTRASS only

7.27.2.5 X509_ECC

```
#define X509_ECC 1
```

Elliptic Curve data type detected

7.27.2.6 X509_ECD

```
#define X509_ECD 3
```

Elliptic Curve (Ed25519) detected

7.27.2.7 X509_H256

```
#define X509_H256 2
```

SHA256 hash algorithm used

7.27.2.8 X509_H384

```
#define X509_H384 3
```

SHA384 hash algorithm used

7.27.2.9 X509_H512

```
#define X509_H512 4
```

SHA512 hash algorithm used

7.27.2.10 X509_HY

```
#define X509_HY 5
```

Hybrid Post_Quantum

7.27.2.11 X509_PQ

```
#define X509_PQ 4
```

Post Quantum method

7.27.2.12 X509_RSA

```
#define X509_RSA 2
```

RSA data type detected

7.27.3 Function Documentation

7.27.3.1 ecdsa_sig_decode()

```
int ecdsa_sig_decode (
            octad * c )
```

in-place ECDSA signature decoding

Parameters

c	an ecdsa signature to be converted from ASN.1 to simple r|s form

Returns

index into c where conversion ended

7.27.3.2 ecdsa_sig_encode()

```
void ecdsa_sig_encode (
            octad * c )
```

in-place ECDSA signature encoding

Parameters

c	an ecdsa signature to be converted from r|s form to ASN.1

7.27.3.3 X509_extract_cert()

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

Parameters

sc	a signed certificate
c	the extracted certificate

Returns

0 on failure

7.27.3.4 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)

7.27.3.5 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)

7.27.3.6 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)

7.27.3.7 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

7.27.3.8 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

7.27.3.9 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

7.27.3.10 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

7.27.3.11 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

7.27.3.12 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

7.27.3.13 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

7.27.3.14 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

7.27.3.15 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

7.27.3.16 X509_self_signed()

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

Parameters

c	an X.509 certificate

Returns

true if self-signed, else false

7.27.4 Variable Documentation

7.27.4.1 X509_AN

```
octad X509_AN  [extern]
```

Alternate Name

7.27.4.2 X509_BC

```
octad X509_BC  [extern]
```

Basic Constraints

7.27.4.3 X509_CN

```
octad X509_CN  [extern]
```

Country Name

7.27.4.4 X509_EN

```
octad X509_EN  [extern]
```

email

7.27.4.5 X509_KU

```
octad X509_KU  [extern]
```

Key Usage

7.27.4.6 X509_LN

```
octad X509_LN  [extern]
```

local name

7.27.4.7 X509_MN

```
octad X509_MN  [extern]
```

My Name (aka Common Name)

7.27.4.8 X509_ON

```
octad X509_ON  [extern]
```

organisation Name

7.27.4.9 X509_SN

```
octad X509_SN  [extern]
```

State Name

7.27.4.10 X509_UN

```
octad X509_UN  [extern]
```

Unit name (aka Organisation Unit OU)

7.28 tls_x509.h

Go to the documentation of this file.

```
1
12 #ifndef TLS_X509_H
13 #define TLS_X509_H
14
15 // Supported Encryption Methods
16
17 #define X509_ECC 1
18 #define X509_RSA 2
19 #define X509_ECD 3
20 #define X509_PQ 4
21 #define X509_HY 5
23 // Supported Hash functions
24
25 #define X509_H256 2
26 #define X509_H384 3
27 #define X509_H512 4
29 // Supported Curves
30
31 #define USE_NIST256 0
32 #define USE_C25519 1
33 //#define USE_BRAINPOOL 2  /**< For Brainpool 256-bit curve - WEIERSTRASS only */
34 //#define USE_ANSSI 3      /**< For French 256-bit standard curve - WEIERSTRASS only */
35 #define USE_NIST384 10
36 #define USE_NIST521 12
38 extern octad X509_CN;
39 extern octad X509_ON;
40 extern octad X509_EN;
41 extern octad X509_LN;
42 extern octad X509_UN;
43 extern octad X509_MN;
44 extern octad X509_SN;
46 extern octad X509_AN;
47 extern octad X509_KU;
48 extern octad X509_BC;
53 typedef struct
54 {
55     int type;
56     int hash;
57     int curve;
58 } pktype;
59
60
65 extern void ecdsa_sig_encode(octad *c);
66
72 extern int ecdsa_sig_decode(octad *c);
73
74 /* X.509 functions */
75
82 extern pktype X509_extract_private_key(octad *c,octad *pk);
83
91 extern pktype X509_extract_cert_sig(octad *c, octad *s);
98 extern int X509_extract_cert(octad *sc, octad *c);
105 extern pktype X509_extract_public_key(octad *c, octad *k);
111 extern int X509_find_issuer(octad *c);
117 extern int X509_find_validity(octad *c);
123 extern int X509_find_subject(octad *c);
124
130 extern int X509_self_signed(octad *c);
131
140 extern int X509_find_entity_property(octad *c, octad *S, int s, int *f);
147 extern int X509_find_start_date(octad *c, int s);
154 extern int X509_find_expiry_date(octad *c, int s);
155
161 extern int X509_find_extensions(octad *c);
170 extern int X509_find_extension(octad *c, octad *S, int s, int *f);
171
179 extern int X509_find_alt_name(octad *c,int s,char *name);
180
181 #endif
```

7.29 ECCX08.h

```
1 /*
2   This file is part of the ArduinoECCX08 library.
3   Copyright (c) 2018 Arduino SA. All rights reserved.
4
5   This library is free software; you can redistribute it and/or
6   modify it under the terms of the GNU Lesser General Public
```

```
7   License as published by the Free Software Foundation; either
8   version 2.1 of the License, or (at your option) any later version.
9
10   This library is distributed in the hope that it will be useful,
11   but WITHOUT ANY WARRANTY; without even the implied warranty of
12   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
13   Lesser General Public License for more details.
14
15   You should have received a copy of the GNU Lesser General Public
16   License along with this library; if not, write to the Free Software
17   Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA  02110-1301  USA
18 */
19
20 #ifndef _ECCX08_H_
21 #define _ECCX08_H_
22
23 #include <Arduino.h>
24 #include <Wire.h>
25
26 class ECCX08Class
27 {
28 public:
29   ECCX08Class(TwoWire& wire, uint8_t address);
30   virtual ~ECCX08Class();
31
32   int begin();
33   void end();
34
35   int serialNumber(byte sn[]);
36   String serialNumber();
37
38   long random(long max);
39   long random(long min, long max);
40   int random(byte data[], size_t length);
41
42   int generatePrivateKey(int slot, byte publicKey[]);
43   int generatePublicKey(int slot, byte publicKey[]);
44   int generateSharedKey(int slot, byte publicKey[], byte sharedKey[]);  // M.Scott 12/7/2021
45
46   int ecdsaVerify(const byte message[], const byte signature[], const byte pubkey[]);
47   int ecSign(int slot, const byte message[], byte signature[]);
48
49   int challenge(const byte message[]);
50   int aesEncrypt(byte block[]);
51   int aesGFM(byte state[],byte H[]);
52
53   int beginSHA256();
54   int beginHMAC(int slot);
55   int updateSHA256(const byte data[],int len); // variable
56   int endSHA256(byte result[]);
57   int endSHA256(const byte data[], int length, byte result[]);
58   int readSHA256(byte context[]);
59   int writeSHA256(byte context[],int length);
60
61   int readSlot(int slot, byte data[], int length);
62   int writeSlot(int slot, const byte data[], int length);
63
64   int locked();
65   int writeConfiguration(const byte data[]);
66   int readConfiguration(byte data[]);
67   int lock();
68
69 private:
70   int wakeup();
71   int sleep();
72   int idle();
73
74   long version();
75   int verify(const byte signature[], const byte pubkey[]);
76   int sign(int slot, byte signature[]);
77
78   int read(int zone, int address, byte buffer[], int length);
79   int write(int zone, int address, const byte buffer[], int length);
80   int lock(int zone);
81
82   int addressForSlotOffset(int slot, int offset);
83
84   int sendCommand(uint8_t opcode, uint8_t param1, uint16_t param2, const byte data[] = NULL, size_t
       dataLength = 0);
85   int receiveResponse(void* response, size_t length);
86   int receiveResponse(void* response);
87   uint16_t crc16(const byte data[], size_t length);
88
89 private:
90   TwoWire* _wire;
91   uint8_t _address;
92
```

```
93   static const uint32_t _wakeupFrequency;
94   static const uint32_t _normalFrequency;
95 };
96
97 extern ECCX08Class ECCX08;
98
99 #endif
```

Index