TII TLS1.3

1.1

Generated by Doxygen 1.8.17

Chapter 1

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 are the only features of C++ that are used. Documentation can be found in the doxygen generated file doc/refman.pdf

1.1 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.

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 (only required for client-side authentication) are stored in the source code file *tls_client_cert.cpp*. The root certificate store is stored in the file *tls_cacert.cpp*. When using secure hardware, the client private key may not be embedded in the source code, rather it exists in secure on-board memory.

The installation process requires the *cmake* utility to be installed. Copy all files and subdirectories from this directory to a working directory.

1.1.1 Miracl

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

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

1.1.2 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

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

1.2 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= B5B74CF6026CF16FA866BA7E7562C53F67A74949FF040319B0BD2149CF4EF97CAD482463F1746D20
Signature Algorithm is RSA_PSS_RSAE_SHA256
Server Cert Verification OK

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

GET / HTTP/1.1
Host: swifttls.org


Waiting for Server input
Got a ticket
Waiting for Server input
Receiving application data (truncated HTML) = 485454502F312E3120323030204F4B0D0A5365727665723A2053776966674544C
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= 04EA04CDA74C1A1942BB8C56C0BD8AE1A4CB9D9B76B5AC64C24CFE7C367B46FA6F06037D945835019D3F1220803
Ticket age= feff
obfuscated age = 447e2e62
Client Hello sent
BND= 258FA2CE9D69253C83646641266B2A81FCEED47348D60E0C7BBB27D2557D1BD2
Sending Binders
Client Early Traffic Secret= CF7D980E8213205CFD35C2194FB75F6D1E98215860BB1F7FA5CFDC8DAE48E9F5
Sending some early data
Sending Application Message

GET / HTTP/1.1
Host: swifttls.org


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

serverHello= 0200009D0303268C69B38026464DFFE72A496662627EC35798DA3F98437042E39CAF404C888520557742FB051C8ADC0A4
Shared Secret= 8C7784C539C0144B8FADCBF065637418F190C49995E79660919E204F05287C2D
Handshake Secret= 4025A7EE2C1B634C9FC83FDF5CFB2FCB5498EA3F5D019EEDC6D3C1D751C87C47
Client handshake traffic secret= 5FC1307F4E7ED84B4196B83EA19D69724812C25A571061FB53B5B6E9FD7FCABE
```

```
Server handshake traffic secret= 1E84FEBA7F8D75F756408906C608925F9A6445292BA614BB398E634CF5854B2A
Early Data Accepted
Warning - ALPN extension NOT acknowledged by server
Server Name NOT acknowledged
Max frag length request NOT acknowledged
Transcript Hash (CH+SH+EE) = DCB73D7B5416D91546EF7D625FBB6A84105CCCE5F054D753275325A822D394E9
Send End of Early Data
Transcript Hash (CH+SH+EE+SF+ED) = FE1FADC8085B3B41A9146647FC9A40F6F2A303533B237112564A2F51F82B64C4
Server Data is verified
Client Verify Data= 350E968A15D36F16BC20D80789E9DB2792A2975765F9BE537407165F7E7366B8
Client application traffic secret= 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. 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)

A resumption ticket can be deleted by

```
./client -r
```

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

1.2.1 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
```

and inserted into the file *tls_client_cert.cpp*

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

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

then for example

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

acts as a normal Website, while

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

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

```
./client localhost
```

1.2.2 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, P↩
OST_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 2

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 ECC↩ X08SelfSignedCert 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.

2.1 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/, lib/ibe, 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_octads.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, 31, 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 Dilithium and Kyber 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 3

Data Structure Index

3.1 Data Structures

Here are the data structures with brief descriptions:

Chapter 4

File Index

4.1 File List

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

Chapter 5

Data Structure Documentation

5.1 crypto Struct Reference

crypto context structure

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

Data Fields

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

5.1.1 Detailed Description

crypto context structure

5.1.2 Field Documentation

5.1.2.1 active

```
bool crypto::active
```

Indicates if encryption has been activated

5.1.2.2 k

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

AEAD cryptographic Key bytes

5.1.2.3 iv

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

AEAD cryptographic IV bytes

5.1.2.4 K

```
octad crypto::K
```

Key as octad

5.1.2.5 IV

```
octad crypto::IV
```

IV as octad

5.1.2.6 record

```
unsign32 crypto::record
```

current record number - to be incremented

5.1.2.7 suite

```
int crypto::suite
```

Cipher Suite

5.1.2.8 taglen

```
int crypto::taglen
```

Tag Length

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

- tls1_3.h

5.2 ECCX08Class Class Reference

Public Member Functions

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

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

- ECCX08.h
- ECCX08.cpp

5.3 ee_status Struct Reference

server encrypted extensions expectations/responses

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

Data Fields

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

5.3.1 Detailed Description

server encrypted extensions expectations/responses

5.3.2 Field Documentation

5.3.2.1 early_data

```
bool ee_status::early_data
```

true if early data accepted

5.3.2.2 alpn

```
bool ee_status::alpn
```

true if ALPN accepted

5.3.2.3 server_name

```
bool ee_status::server_name
```

true if server name accepted

5.3.2.4 max_frag_length

```
bool ee_status::max_frag_length
```

true if max frag length respected

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

- tls1_3.h

5.4 octad Struct Reference

Safe representation of an octad.

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

Data Fields

- int len
- int max
- char ∗ val

5.4.1 Detailed Description

Safe representation of an octad.

5.4.2 Field Documentation

5.4.2.1 len

```
int octad::len
```

length in bytes

5.4.2.2 max

```
int octad::max
```

max length allowed - enforce truncation

5.4.2.3 val

```
char∗ octad::val
```

byte array

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

- tls_octads.h

5.5 pktype Struct Reference

Public key type.

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

Data Fields

- int type
- int hash
- int curve

5.5.1 Detailed Description

Public key type.

5.5.2 Field Documentation

5.5.2.1 type

```
int pktype::type
```

signature type (ECC or RSA)

5.5.2.2 hash

```
int pktype::hash
```

hash type

5.5.2.3 curve

```
int pktype::curve
```

elliptic curve used or RSA key length in bits

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

- tls_x509.h

5.6 ret Struct Reference

function return structure

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

Data Fields

- unsign32 val
- int err

5.6.1 Detailed Description

function return structure

5.6.2 Field Documentation

5.6.2.1 val

```
unsign32 ret::val
```

return value

5.6.2.2 err

```
int ret::err
```

error return

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

- tls1_3.h

5.7 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** ()

5.7.1 Detailed Description

Socket instance.

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

- tls_sockets.h

5.8 ticket Struct Reference

ticket context structure

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

Data Fields

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

5.8.1 Detailed Description

ticket context structure

5.8.2 Field Documentation

5.8.2.1 valid

```
bool ticket::valid
```

Is ticket valid?

5.8.2.2 tick

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

Ticket bytes

5.8.2.3 nonce

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

nonce

5.8.2.4 psk

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

pre-shared key

5.8.2.5 TICK

```
octad ticket::TICK
```

Ticket or external PSK label as octad

5.8.2.6 NONCE

```
octad ticket::NONCE
```

Nonce as octad

5.8.2.7 PSK

```
octad ticket::PSK
```

PSK as octad

5.8.2.8 age_obfuscator

```
unsign32 ticket::age_obfuscator
```

ticket age obfuscator - 0 for external PSK

5.8.2.9 max_early_data

```
unsign32 ticket::max_early_data
```

Maximum early data allowed for this ticket

5.8.2.10 birth

```
unsign32 ticket::birth
```

Birth time of this ticket

5.8.2.11 lifetime

```
int ticket::lifetime
```

ticket lifetime

5.8.2.12 cipher_suite

```
int ticket::cipher_suite
```

Cipher suite used

5.8.2.13 favourite_group

```
int ticket::favourite_group
```

the server's favourite group

5.8.2.14 origin

```
int ticket::origin
```

Origin of initial handshake - Full or PSK?

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

- tls1_3.h

5.9 TLS_session Struct Reference

TLS1.3 session state.

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

Data Fields

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

5.9.1 Detailed Description

TLS1.3 session state.

5.9.2 Field Documentation

5.9.2.1 status

```
int TLS_session::status
```

Connection status

5.9.2.2 max_record

```
int TLS_session::max_record
```

max record size I should send

5.9.2.3 sockptr

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

Pointer to socket

5.9.2.4 id

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

Session ID

5.9.2.5 hostname

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

Server name for connection

5.9.2.6 cipher_suite

```
int TLS_session::cipher_suite
```

agreed cipher suite

5.9.2.7 favourite_group

```
int TLS_session::favourite_group
```

favourite key exchange group - may be changed on handshake retry

5.9.2.8 K_send

```
crypto TLS_session::K_send
```

Sending Key

5.9.2.9 K_recv

```
crypto TLS_session::K_recv
```

Receiving Key

5.9.2.10 HS

```
octad TLS_session::HS
```

Handshake secret

5.9.2.11 hs

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

Handshake secret data

5.9.2.12 RMS

```
octad TLS_session::RMS
```

Resumption Master Secret

5.9.2.13 rms

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

Resumption Master Secret data

5.9.2.14 STS

```
octad TLS_session::STS
```

Server Traffic secret

5.9.2.15 sts

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

Server Traffic secret data

5.9.2.16 CTS

```
octad TLS_session::CTS
```

Client Traffic secret

5.9.2.17 cts

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

Client Traffic secret data

5.9.2.18 CTX

```
octad TLS_session::CTX
```

Certificate Request Context

5.9.2.19 ctx

```
char TLS_session::ctx[TLS_MAX_HASH]
```

Certificate Request Context data

5.9.2.20 IO

```
octad TLS_session::IO
```

Main IO buffer for this connection

5.9.2.21 io

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

Byte array for main IO buffer for this connection

5.9.2.22 ptr

```
int TLS_session::ptr
```

pointer into IO buffer

5.9.2.23 tlshash

```
unihash TLS_session::tlshash
```

Transcript hash recorder

5.9.2.24 T

```
ticket TLS_session::T
```

resumption ticket

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

- tls1_3.h

5.10 unihash Struct Reference

Universal Hash Function.

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

Data Fields

- char state [TLS_MAX_HASH_STATE]
- int htype

5.10.1 Detailed Description

Universal Hash Function.

5.10.2 Field Documentation

5.10.2.1 state

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

hash function state

5.10.2.2 htype

```
int unihash::htype
```

The hash type (typically SHA256)

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

- tls1_3.h

Chapter 6

File Documentation

6.1 tls1_3.h File Reference

Main TLS 1.3 Header File for constants and structures.

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

Data Structures

- struct ret

    function return structure
- struct ee_status

    server encrypted extensions expectations/responses
- struct crypto

    crypto context structure
- struct ticket

    ticket context structure
- struct unihash

    Universal Hash Function.
- struct TLS_session

    TLS1.3 session state.

Macros

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

- #define 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 2023
- #define POST_HS_AUTH
- #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_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 4196
- #define TLS_MAX_EXTENSIONS 6144
- #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 TLS13_UPDATE_NOT_REQUESTED 0
- #define TLS13_UPDATE_REQUESTED 1
- #define SERVER_NAME 0x0000
- #define SUPPORTED_GROUPS 0x000a
- #define SIG_ALGS 0x000d
- #define POST_HANDSHAKE_AUTH 0x0031
- #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 ERROR_ALERT_RECEIVED -22
- #define BAD_MESSAGE -23
- #define CERT_VERIFY_FAIL -24
- #define BAD_HANDSHAKE -26
- #define BAD_REQUEST_UPDATE -27
- #define CLOSURE_ALERT_RECEIVED -28
- #define MISSING_EXTENSIONS -30
- #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 BAD_RECORD_MAC 0x14
- #define HANDSHAKE_FAILURE 0x28
- #define CLOSE_NOTIFY 0x00
- #define MISSING_EXTENSION 0x6D;
- #define LOG_OUTPUT_TRUNCATION 256
- #define TLS13_DISCONNECTED 0
- #define TLS13_CONNECTED 1
- #define TLS13_HANDSHAKING 2
- #define TLS_FAILURE 0
- #define TLS_SUCCESS 1
- #define TLS_RESUMPTION_REQUIRED 2
- #define TLS_EARLY_DATA_ACCEPTED 3
- #define PSK_NOT 0
- #define PSK_KEY 1
- #define PSK_IBE 2

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

6.1.1 Detailed Description

Main TLS 1.3 Header File for constants and structures.

Author

Mike Scott

6.1.2 Macro Definition Documentation

6.1.2.1 IO_NONE

```
#define IO_NONE 0
```

Run silently

6.1.2.2 IO_APPLICATION

```
#define IO_APPLICATION 1
```

just print application traffic

6.1.2.3 IO_PROTOCOL

```
#define IO_PROTOCOL 2
```

print protocol progress + application traffic

6.1.2.4 IO_DEBUG

```
#define IO_DEBUG 3
```

print lots of debug information + protocol progress + application progress

6.1.2.5 IO_WIRE

```
#define IO_WIRE 4
```

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

6.1.2.6 TINY_ECC

```
#define TINY_ECC 0
```

ECC keys only

6.1.2.7 TYPICAL

```
#define TYPICAL 1
```

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

6.1.2.8 POST_QUANTUM

```
#define POST_QUANTUM 2
```

Post quantum (Dilithium+Kyber?)

6.1.2.9 HYBRID

```
#define HYBRID 3
```

Hybrid, Kyber/Dilithium + X25519

6.1.2.10 NOCERT

```
#define NOCERT 0
```

Don't have a Client Cert

6.1.2.11 RSA_SS

```
#define RSA_SS 1
```

self signed RSA cert

6.1.2.12 ECC_SS

```
#define ECC_SS 2
```

self signed ECC cert

6.1.2.13 DLT_SS

```
#define DLT_SS 3
```

self signed Dilithium cert

6.1.2.14 HYB_SS

```
#define HYB_SS 6
```

self signed Hybrid cert (Dilithium+ECC)

6.1.2.15 HW_1

```
#define HW_1 4
```

RP2040 1 Hardware cert

6.1.2.16 HW_2

```
#define HW_2 5
```

RP2040 2 Hardware cert

6.1.2.17 VERBOSITY

```
#define VERBOSITY IO_PROTOCOL
```

Set to level of output information desired - see above

6.1.2.18 THIS_YEAR

```
#define THIS_YEAR 2023
```

Set to this year - crudely used to deprecate old certificates

6.1.2.19 POST_HS_AUTH

```
#define POST_HS_AUTH
```

Willing to do post handshake authentication

6.1.2.20 CLIENT_CERT

```
#define CLIENT_CERT ECC_SS
```

Indicate capability of authenticating with a cert plus signing key

6.1.2.21 CRYPTO_SETTING

```
#define CRYPTO_SETTING TYPICAL
```

Determine Cryptography settings

6.1.2.22 TLS_APPLICATION_PROTOCOL

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

Support ALPN protocol

6.1.2.23 ALLOW_SELF_SIGNED

```
#define ALLOW_SELF_SIGNED
```

allow self-signed server cert

6.1.2.24 TRY_EARLY_DATA

```
#define TRY_EARLY_DATA
```

Try to send early data on resumptions

6.1.2.25 TLS_SHA256_T

```
#define TLS_SHA256_T 1
```

SHA256 hash

6.1.2.26 TLS_SHA384_T

```
#define TLS_SHA384_T 2
```

SHA384 hash

6.1.2.27 TLS_SHA512_T

```
#define TLS_SHA512_T 3
```

SHA512 hash

6.1.2.28 TLS_MAX_HASH_STATE

```
#define TLS_MAX_HASH_STATE 768
```

Maximum memory required to store hash function state

6.1.2.29 TLS_MAX_HASH

```
#define TLS_MAX_HASH 64
```

Maximum hash output length in bytes

6.1.2.30 TLS_MAX_KEY

```
#define TLS_MAX_KEY 32
```

Maximum key length in bytes

6.1.2.31 TLS_X509_MAX_FIELD

```
#define TLS_X509_MAX_FIELD 256
```

Maximum X.509 field size

6.1.2.32 TLS_MAX_EXT_LABEL

```
#define TLS_MAX_EXT_LABEL 256
```

Max external psk label size

6.1.2.33 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

6.1.2.34 TLS_MAX_SERVER_CHAIN_LEN

```
#define TLS_MAX_SERVER_CHAIN_LEN 2
```

Maximum Server Certificate chain length - omitting root CA

6.1.2.35 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

6.1.2.36 TLS_MAX_CLIENT_CHAIN_LEN

```
#define TLS_MAX_CLIENT_CHAIN_LEN 1
```

Maximum Client Certificate chain length - one self signed here

6.1.2.37 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

6.1.2.38 TLS_MAX_SHARED_SECRET_SIZE

```
#define TLS_MAX_SHARED_SECRET_SIZE 256
```

Max key exchange Shared secret size

6.1.2.39 TLS_MAX_TICKET_SIZE

```
#define TLS_MAX_TICKET_SIZE 4196
```

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

6.1.2.40 TLS_MAX_EXTENSIONS

```
#define TLS_MAX_EXTENSIONS 6144
```

Max extensions size

6.1.2.41 TLS_MAX_ECC_FIELD

```
#define TLS_MAX_ECC_FIELD 66
```

Max ECC field size in bytes

6.1.2.42 TLS_MAX_IV_SIZE

```
#define TLS_MAX_IV_SIZE 12
```

Max IV size in bytes

6.1.2.43 TLS_MAX_TAG_SIZE

```
#define TLS_MAX_TAG_SIZE 16
```

Max HMAC tag length in bytes

6.1.2.44 TLS_MAX_COOKIE

```
#define TLS_MAX_COOKIE 128
```

Max Cookie size

6.1.2.45 TLS_MAX_SERVER_NAME

```
#define TLS_MAX_SERVER_NAME 128
```

Max server name size in bytes

6.1.2.46 TLS_MAX_SUPPORTED_GROUPS

```
#define TLS_MAX_SUPPORTED_GROUPS 10
```

Max number of supported crypto groups

6.1.2.47 TLS_MAX_SUPPORTED_SIGS

```
#define TLS_MAX_SUPPORTED_SIGS 16
```

Max number of supported signature schemes

6.1.2.48 TLS_MAX_PSK_MODES

```
#define TLS_MAX_PSK_MODES 2
```

Max preshared key modes

6.1.2.49 TLS_MAX_CIPHER_SUITES

```
#define TLS_MAX_CIPHER_SUITES 5
```

Max number of supported cipher suites

6.1.2.50 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

6.1.2.51 TLS_AES_256_GCM_SHA384

```
#define TLS_AES_256_GCM_SHA384 0x1302
```

AES256/SHA384/GCM cipher suite

6.1.2.52 TLS_CHACHA20_POLY1305_SHA256

```
#define TLS_CHACHA20_POLY1305_SHA256 0x1303
```

CHACHA20/SHA256/POLY1305 cipher suite

6.1.2.53 TLS_AES_128_CCM_SHA256

```
#define TLS_AES_128_CCM_SHA256 0x1304
```

AES/SHA256/CCM cipher suite - optional

6.1.2.54 TLS_AES_128_CCM_8_SHA256

```
#define TLS_AES_128_CCM_8_SHA256 0x1305
```

AES/SHA256/CCM 8 cipher suite - optional

6.1.2.55 X25519

```
#define X25519 0x001d
```

X25519 elliptic curve key exchange

6.1.2.56 SECP256R1

```
#define SECP256R1 0x0017
```

NIST SECP256R1 elliptic curve key exchange

6.1.2.57 SECP384R1

```
#define SECP384R1 0x0018
```

NIST SECP384R1 elliptic curve key exchange

6.1.2.58 SECP521R1

```
#define SECP521R1 0x0019
```

NIST SECP521R1 elliptic curve key exchange

6.1.2.59 X448

```
#define X448 0x001e
```

X448 elliptic curve key exchange

6.1.2.60 KYBER768

```
#define KYBER768 0x4242
```

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

6.1.2.61 HYBRID_KX

```
#define HYBRID_KX 0x421d
```

Hybrid key exchange, Kyber+X25519

6.1.2.62 ECDSA_SECP256R1_SHA256

```
#define ECDSA_SECP256R1_SHA256 0x0403
```

Supported ECDSA Signature algorithm

6.1.2.63 ECDSA_SECP256R1_SHA384

```
#define ECDSA_SECP256R1_SHA384 0x0413
```

Non-standard ECDSA Signature algorithm

6.1.2.64 ECDSA_SECP384R1_SHA384

```
#define ECDSA_SECP384R1_SHA384 0x0503
```

Supported ECDSA Signature algorithm

6.1.2.65 RSA_PSS_RSAE_SHA256

```
#define RSA_PSS_RSAE_SHA256 0x0804
```

Supported RSA Signature algorithm

6.1.2.66 RSA_PSS_RSAE_SHA384

```
#define RSA_PSS_RSAE_SHA384 0x0805
```

Supported RSA Signature algorithm

6.1.2.67 RSA_PSS_RSAE_SHA512

```
#define RSA_PSS_RSAE_SHA512 0x0806
```

Supported RSA Signature algorithm

6.1.2.68 RSA_PKCS1_SHA256

```
#define RSA_PKCS1_SHA256 0x0401
```

Supported RSA Signature algorithm

6.1.2.69 RSA_PKCS1_SHA384

```
#define RSA_PKCS1_SHA384 0x0501
```

Supported RSA Signature algorithm

6.1.2.70 RSA_PKCS1_SHA512

```
#define RSA_PKCS1_SHA512 0x0601
```

Supported RSA Signature algorithm

6.1.2.71 ED25519

```
#define ED25519 0x0807
```

Ed25519 EdDSA Signature algorithm

6.1.2.72 DILITHIUM2

```
#define DILITHIUM2 0x0902
```

Dilithium2 Signature algorithm

6.1.2.73 DILITHIUM3

```
#define DILITHIUM3 0x0903
```

Dilithium3 Signature algorithm

6.1.2.74 DILITHIUM2_P256

```
#define DILITHIUM2_P256 0x09F2
```

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

6.1.2.75 PSKOK

```
#define PSKOK 0x00
```

Preshared Key only mode

6.1.2.76 PSKWECDHE

```
#define PSKWECDHE 0x01
```

Preshared Key with Diffie-Hellman key exchange mode

6.1.2.77 TLS_FULL_HANDSHAKE

```
#define TLS_FULL_HANDSHAKE 1
```

Came from Full Handshake

6.1.2.78 TLS_EXTERNAL_PSK

```
#define TLS_EXTERNAL_PSK 2
```

External Pre-Shared Key

6.1.2.79 TLS1_0

```
#define TLS1_0 0x0301
```

TLS 1.0 version

6.1.2.80 TLS1_2

```
#define TLS1_2 0x0303
```

TLS 1.2 version

6.1.2.81 TLS1_3

```
#define TLS1_3 0x0304
```

TLS 1.3 version

6.1.2.82 TLS13_UPDATE_NOT_REQUESTED

```
#define TLS13_UPDATE_NOT_REQUESTED 0
```

Updating my keys

6.1.2.83 TLS13_UPDATE_REQUESTED

```
#define TLS13_UPDATE_REQUESTED 1
```

Updating my keys and telling you to update yours

6.1.2.84 SERVER_NAME

```
#define SERVER_NAME 0x0000
```

Server Name extension

6.1.2.85 SUPPORTED_GROUPS

```
#define SUPPORTED_GROUPS 0x000a
```

Supported Group extension

6.1.2.86 SIG_ALGS

```
#define SIG_ALGS 0x000d
```

Signature algorithms extension

6.1.2.87 POST_HANDSHAKE_AUTH

```
#define POST_HANDSHAKE_AUTH 0x0031
```

Post Handshake Authentication

6.1.2.88 SIG_ALGS_CERT

```
#define SIG_ALGS_CERT 0x0032
```

Signature algorithms Certificate extension

6.1.2.89 KEY_SHARE

```
#define KEY_SHARE 0x0033
```

Key Share extension

6.1.2.90 PSK_MODE

```
#define PSK_MODE 0x002d
```

Preshared key mode extension

6.1.2.91 PRESHARED_KEY

```
#define PRESHARED_KEY 0x0029
```

Preshared key extension

6.1.2.92 TLS_VER

```
#define TLS_VER 0x002b
```

TLS version extension

6.1.2.93 COOKIE

```
#define COOKIE 0x002c
```

Cookie extension

6.1.2.94 EARLY_DATA

```
#define EARLY_DATA 0x002a
```

Early Data extension

6.1.2.95 MAX_FRAG_LENGTH

```
#define MAX_FRAG_LENGTH 0x0001
```

max fragmentation length extension

6.1.2.96 PADDING

```
#define PADDING 0x0015
```

Padding extension

6.1.2.97 APP_PROTOCOL

```
#define APP_PROTOCOL 0x0010
```

Application Layer Protocol Negotiation (ALPN)

6.1.2.98 RECORD_SIZE_LIMIT

```
#define RECORD_SIZE_LIMIT 0x001c
```

Record Size Limit

6.1.2.99 HSHAKE

```
#define HSHAKE 0x16
```

Handshake record

6.1.2.100 APPLICATION

```
#define APPLICATION 0x17
```

Application record

6.1.2.101 ALERT

```
#define ALERT 0x15
```

Alert record

6.1.2.102 CHANGE_CIPHER

```
#define CHANGE_CIPHER 0x14
```

Change Cipher record

6.1.2.103 TIMED_OUT

```
#define TIMED_OUT 0x01
```

Time-out

6.1.2.104 CLIENT_HELLO

```
#define CLIENT_HELLO 0x01
```

Client Hello message

6.1.2.105 SERVER_HELLO

```
#define SERVER_HELLO 0x02
```

Server Hello message

6.1.2.106 CERTIFICATE

```
#define CERTIFICATE 0x0b
```

Certificate message

6.1.2.107 CERT_REQUEST

```
#define CERT_REQUEST 0x0d
```

Certificate Request

6.1.2.108 CERT_VERIFY

```
#define CERT_VERIFY 0x0f
```

Certificate Verify message

6.1.2.109 FINISHED

```
#define FINISHED 0x14
```

Handshake Finished message

6.1.2.110 ENCRYPTED_EXTENSIONS

```
#define ENCRYPTED_EXTENSIONS 0x08
```

Encrypted Extensions message

6.1.2.111 TICKET

```
#define TICKET 0x04
```

Ticket message

6.1.2.112 KEY_UPDATE

```
#define KEY_UPDATE 0x18
```

Key Update message

6.1.2.113 MESSAGE_HASH

```
#define MESSAGE_HASH 0xFE
```

Special synthetic message hash message

6.1.2.114 END_OF_EARLY_DATA

```
#define END_OF_EARLY_DATA 0x05
```

End of Early Data message

6.1.2.115 HANDSHAKE_RETRY

```
#define HANDSHAKE_RETRY 0x102
```

Handshake retry

6.1.2.116 NOT_TLS1_3

```
#define NOT_TLS1_3 -2
```

Wrong version error, not TLS1.3

6.1.2.117 BAD_CERT_CHAIN

```
#define BAD_CERT_CHAIN -3
```

Bad Certificate Chain error

6.1.2.118 ID_MISMATCH

```
#define ID_MISMATCH -4
```

Session ID mismatch error

6.1.2.119 UNRECOGNIZED_EXT

```
#define UNRECOGNIZED_EXT -5
```

Unrecognised extension error

6.1.2.120 BAD_HELLO

```
#define BAD_HELLO -6
```

badly formed Hello message error

6.1.2.121 WRONG_MESSAGE

```
#define WRONG_MESSAGE -7
```

Message out-of-order error

6.1.2.122 MISSING_REQUEST_CONTEXT

```
#define MISSING_REQUEST_CONTEXT -8
```

Request context missing error

6.1.2.123 AUTHENTICATION_FAILURE

```
#define AUTHENTICATION_FAILURE -9
```

Authentication error - AEAD Tag incorrect

6.1.2.124 BAD_RECORD

```
#define BAD_RECORD -10
```

Badly formed Record received

6.1.2.125 BAD_TICKET

```
#define BAD_TICKET -11
```

Badly formed Ticket received

6.1.2.126 NOT_EXPECTED

```
#define NOT_EXPECTED -12
```

Received ack for something not requested

6.1.2.127 CA_NOT_FOUND

```
#define CA_NOT_FOUND -13
```

Certificate Authority not found

6.1.2.128 CERT_OUTOFDATE

```
#define CERT_OUTOFDATE -14
```

Certificate Expired

6.1.2.129 MEM_OVERFLOW

```
#define MEM_OVERFLOW -15
```

Memory Overflow

6.1.2.130 FORBIDDEN_EXTENSION

```
#define FORBIDDEN_EXTENSION -16
```

Forbidden Encrypted Extension

6.1.2.131 MAX_EXCEEDED

```
#define MAX_EXCEEDED -17
```

Maximum record size exceeded

6.1.2.132 EMPTY_CERT_CHAIN

```
#define EMPTY_CERT_CHAIN -18
```

Empty Certificate Message

6.1.2.133 SELF_SIGNED_CERT

```
#define SELF_SIGNED_CERT -20
```

Self signed certificate

6.1.2.134 ERROR_ALERT_RECEIVED

```
#define ERROR_ALERT_RECEIVED -22
```

Alert has been received

6.1.2.135 BAD_MESSAGE

```
#define BAD_MESSAGE -23
```

Badly formed message

6.1.2.136 CERT_VERIFY_FAIL

```
#define CERT_VERIFY_FAIL -24
```

Certificate Verification failure

6.1.2.137 BAD_HANDSHAKE

```
#define BAD_HANDSHAKE -26
```

Could not agree

6.1.2.138 BAD_REQUEST_UPDATE

```
#define BAD_REQUEST_UPDATE -27
```

Bad Request Update value

6.1.2.139 CLOSURE_ALERT_RECEIVED

```
#define CLOSURE_ALERT_RECEIVED -28
```

Alert has been received

6.1.2.140 MISSING_EXTENSIONS

```
#define MISSING_EXTENSIONS -30
```

Some mandatory extensions are missing

6.1.2.141 ILLEGAL_PARAMETER

```
#define ILLEGAL_PARAMETER 0x2F
```

Illegal parameter alert

6.1.2.142 UNEXPECTED_MESSAGE

```
#define UNEXPECTED_MESSAGE 0x0A
```

Unexpected message alert

6.1.2.143 DECRYPT_ERROR

```
#define DECRYPT_ERROR 0x33
```

Decryption error alert

6.1.2.144 BAD_CERTIFICATE

```
#define BAD_CERTIFICATE 0x2A
```

Bad certificate alert

6.1.2.145 UNSUPPORTED_EXTENSION

```
#define UNSUPPORTED_EXTENSION 0x6E
```

Unsupported extension alert

6.1.2.146 UNKNOWN_CA

```
#define UNKNOWN_CA 0x30
```

Unrecognised Certificate Authority

6.1.2.147 CERTIFICATE_EXPIRED

```
#define CERTIFICATE_EXPIRED 0x2D
```

Certificate Expired

6.1.2.148 PROTOCOL_VERSION

```
#define PROTOCOL_VERSION 0x46
```

Wrong TLS version

6.1.2.149 DECODE_ERROR

```
#define DECODE_ERROR 0x32
```

Decode error alert

6.1.2.150 RECORD_OVERFLOW

```
#define RECORD_OVERFLOW 0x16
```

Record Overflow

6.1.2.151 BAD_RECORD_MAC

```
#define BAD_RECORD_MAC 0x14
```

Bad Record Mac

6.1.2.152 HANDSHAKE_FAILURE

```
#define HANDSHAKE_FAILURE 0x28
```

Could not agree

6.1.2.153 CLOSE_NOTIFY

```
#define CLOSE_NOTIFY 0x00
```

Orderly shut down of connection

6.1.2.154 MISSING_EXTENSION

```
#define MISSING_EXTENSION 0x6D;
```

Missing extension

6.1.2.155 LOG_OUTPUT_TRUNCATION

```
#define LOG_OUTPUT_TRUNCATION 256
```

Output Hex digits before truncation

6.1.2.156 TLS13_DISCONNECTED

```
#define TLS13_DISCONNECTED 0
```

TLS1.3 Connection is broken

6.1.2.157 TLS13_CONNECTED

```
#define TLS13_CONNECTED 1
```

TLS1.3 Connection is made

6.1.2.158 TLS13_HANDSHAKING

```
#define TLS13_HANDSHAKING 2
```

TLS1.3 is handshaking

6.1.2.159 TLS_FAILURE

```
#define TLS_FAILURE 0
```

Failed to cmake TLS1.3 connection

6.1.2.160 TLS_SUCCESS

```
#define TLS_SUCCESS 1
```

Succeeded in making TLS1.3 connection

6.1.2.161 TLS_RESUMPTION_REQUIRED

```
#define TLS_RESUMPTION_REQUIRED 2
```

Connection succeeded, but handshake retry was needed

6.1.2.162 TLS_EARLY_DATA_ACCEPTED

```
#define TLS_EARLY_DATA_ACCEPTED 3
```

Connection succeeded, and early data was accepted

6.1.2.163 PSK_NOT

```
#define PSK_NOT 0
```

No PSK

6.1.2.164 PSK_KEY

```
#define PSK_KEY 1
```

Using PSK from database

6.1.2.165 PSK_IBE

```
#define PSK_IBE 2
```

Using IBE based PSK

6.1.3 Typedef Documentation

6.1.3.1 byte

```
typedef uint8_t byte
```

8-bit unsigned integer

6.1.3.2 sign8

```
typedef int8_t sign8
```

8-bit signed integer

6.1.3.3 sign16

```
typedef int16_t sign16
```

16-bit signed integer

6.1.3.4 sign32

```
typedef int32_t sign32
```

32-bit signed integer

6.1.3.5 sign64

```
typedef int64_t sign64
```

64-bit signed integer

6.1.3.6 unsign32

```
typedef uint32_t unsign32
```

32-bit unsigned integer

6.1.3.7 unsign64

```
typedef uint64_t unsign64
```

64-bit unsigned integer

6.2 tls_cert_chain.h File Reference

Process Certificate Chain.

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

Functions

- int checkServerCertChain (octad ∗CERTCHAIN, char ∗hostname, octad ∗PUBKEY, octad ∗SIG)

    Check Certificate Chain for hostname, and extract public key.
- int getClientPrivateKeyandCertChain (octad ∗PRIVKEY, octad ∗CERTCHAIN)

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

6.2.1 Detailed Description

Process Certificate Chain.

Author

Mike Scott

6.2.2 Function Documentation

6.2.2.1 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

6.2.2.2 getClientPrivateKeyandCertChain()

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

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

Parameters

PRIVKEY	the Client's private key
CERTCHAIN	the Client's certificate chain

Returns

type of private key, ECC or RSA

6.3 tls_certs.h File Reference

Certificate Authority root certificate store.

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

Functions

- int getSigRequirements (int ∗sigReqs)

  Get Client Certificate chain requirements.

Variables

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

6.3.1 Detailed Description

Certificate Authority root certificate store.

Author

Mike Scott

6.3.2 Function Documentation

6.3.2.1 getSigRequirements()

```
int getSigRequirements (
            int * sigReqs )
```

Get Client Certificate chain requirements.

Parameters

sigReqs	list of signature requirements

Returns

number of such requirements

6.3.3 Variable Documentation

6.3.3.1 myprivate

```
const char * myprivate =NULL
```

Client private key

6.3.3.2 mycert

```
const char * mycert
```

Initial value:

```
=(char *)
"-----BEGIN CERTIFICATE-----\n"
"MIIBKzCB0aADAgECAgEBMAoGCCqGSM49BAMCMB0xGzAZBgNVBAMTEjAxMjM0NjI0QjIwMjYwRDdF\n"
"RTAeFw0yMTExMTgxMTAwMDBaFw0yNjExMTgxMTAwMDBaMB0xGzAZBgNVBAMTEjAxMjM0NjI0QjIw\n"
"MjYwRDdFRTBZMBMGByqGSM49AgEGCCqGSM49AwEHA0IABDOFj/SnArwqM15cZs/bXppfTuAxgMzB\n"
"N3LS48xHSqpLhHlVnvOvWqyhE8v+ZX4Jzlo7Z9LGOG537EeldBeGjYijAjAAMAoGCCqGSM49BAMC\n"
"A0kAMEYCIQC9O1l85YX1+9vZ0t/SHQ3zFH5e7Vc8XtrZ+mTtMc5riwIhAL/SektrG3C0JwII0VV5\n"
"pSR9RRnuwo810km81P4S56/m\n"
"-----END CERTIFICATE-----\n"
```

Client certificate

6.3.3.3 cacerts

```
const char* cacerts
```

The Root Certificate store

6.4 tls_client_recv.h File Reference

Process Input received from the Server.

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

Functions

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

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

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

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

  Return a pointer to an octad from a pointer into an octad.
- int getServerRecord (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, bool context)

  process a Certificate Request

6.4.1 Detailed Description

Process Input received from the Server.

Author

Mike Scott

6.4.2 Function Documentation

6.4.2.1 parseoctad()

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

Parse out an octad from a pointer into an octad.

Parameters

E	the output octad copied out from the octad M
len	the expected length of the output octad E
M	the input octad
ptr	a pointer into M, which advances after use

Returns

the actual length of E extracted, and an error flag

6.4.2.2 parsebytes()

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

Parse out byte array from a pointer into an octad.

Parameters

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

6.4.2.3 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

6.4.2.4 parseoctadptr()

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

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

Parameters

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

6.4.2.5 getServerRecord()

```
int getServerRecord (
            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

6.4.2.6 parseIntorPull()

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

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

Parameters

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

Returns

the unsigned integer, and an error flag

6.4.2.7 parseoctadorPull()

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

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

Parameters

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

6.4.2.8 parsebytesorPull()

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

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

Parameters

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

6.4.2.9 parseoctadorPullptrX()

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

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

Parameters

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

6.4.2.10 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

6.4.2.11 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

6.4.2.12 getServerEncryptedExtensions()

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

Receive and parse Server Encrypted Extensions.

Parameters

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

Returns

response structure

6.4.2.13 getServerCertVerify()

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

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

Parameters

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

Returns

response structure

6.4.2.14 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

6.4.2.15 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

6.4.2.16 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

6.4.2.17 getCertificateRequest()

```
ret getCertificateRequest (
             TLS_session * session,
             bool context )
```

process a Certificate Request

Parameters

session	the TLS session structure
context	true if expecting a context

Returns

response structure

6.5 tls_client_send.h File Reference

Process Output to be sent to the Server.

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

Functions

- void sendCCCS (TLS_session ∗session)

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

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

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

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

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

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

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

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

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

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

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

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

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

  Add Cookie extension to under-construction Extensions Octet.

- void addEarlyDataExt (octad ∗EXT)

  Indicate desire to send Early Data in under-construction Extensions Octet.
- void addPostHSAuth (octad ∗EXT)

  indicate willingness to do post handshake authentication
- 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, octad ∗CRN, 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 sendKeyUpdate (TLS_session ∗session, int type)

  Prepare and send a key update 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.

6.5.1 Detailed Description

Process Output to be sent to the Server.

Author

Mike Scott

6.5.2 Function Documentation

6.5.2.1 sendCCCS()

```
void sendCCCS (
            TLS_session ∗ session )
```

Send Change Cipher Suite message.

Parameters

session	the TLS session structure

6.5.2.2 addPreSharedKeyExt()

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

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

Parameters

EXT	the extensions octad which is being built
age	the obfuscated age of the preshared key
IDS	the proposed preshared key identity
sha	the hash algorithm used to calculate the HMAC binder

Returns

length of binder to be sent later

6.5.2.3 addServerNameExt()

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

Add Server name extension to under-construction Extensions Octet.

Parameters

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

6.5.2.4 addSupportedGroupsExt()

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

Add Supported Groups extension to under-construction Extensions Octet.

Parameters

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

6.5.2.5 addSigAlgsExt()

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

Add Supported TLS1.3 Signature algorithms to under-construction Extensions Octet.

Parameters

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

6.5.2.6 addSigAlgsCertExt()

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

Add Supported X.509 Certificate Signature algorithms to under-construction Extensions Octet.

Parameters

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

6.5.2.7 addKeyShareExt()

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

Add Key Share extension to under-construction Extensions Octet.

Parameters

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

6.5.2.8 addALPNExt()

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

Add Application Layer Protocol Negotiation (ALPN) extension to under-construction Extensions Octet.

Parameters

EXT	the extensions octad which is being built
AP	the IANA sequence associated with the expected protocol

6.5.2.9 addMFLExt()

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

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

Parameters

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

6.5.2.10 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

6.5.2.11 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

6.5.2.12 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

6.5.2.13 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

6.5.2.14 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

6.5.2.15 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

6.5.2.16 addPostHSAuth()

```
void addPostHSAuth (
            octad * EXT )
```

indicate willingness to do post handshake authentication

Parameters

EXT	the extensions octad which is being built

6.5.2.17 clientRandom()

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

Generate 32-byte random octad.

Parameters

RN	the output 32-byte octad

Returns

length of output octad

6.5.2.18 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

6.5.2.19 sendFlushIO()

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

Flush IO buffer.

Parameters

session	the TLS session structure

6.5.2.20 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

6.5.2.21 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

6.5.2.22 sendClientHello()

```
void sendClientHello (
            TLS_session * session,
            int version,
            octad * CH,
            octad * CRN,
            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
CRN	Random bytes
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

6.5.2.23 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

6.5.2.24 sendKeyUpdate()

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

Prepare and send a key update message to the Server.

Parameters

session	the TLS session structure
type	the type of the update

6.5.2.25 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

6.5.2.26 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

6.5.2.27 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

6.5.2.28 sendEndOfEarlyData()

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

Indicate End of Early Data in message to the Server.

Parameters

session	the TLS session structure

6.5.2.29 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

6.6 tls_keys_calc.h File Reference

TLS 1.3 crypto support functions.

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

Functions

- void initTranscriptHash (TLS_session ∗session)

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

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

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

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

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

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

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

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

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

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

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

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

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

6.6.1 Detailed Description

TLS 1.3 crypto support functions.

Author

Mike Scott

6.6.2 Function Documentation

6.6.2.1 initTranscriptHash()

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

Initialise Transcript hash.

Parameters

session	the TLS session structure

6.6.2.2 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

6.6.2.3 runningHashIO()

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

Accumulate transcript hash from IO buffer.

Parameters

session	the TLS session structure

6.6.2.4 rewindIO()

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

rewind the IO buffer

Parameters

session	the TLS session structure

6.6.2.5 runningHashIOrewind()

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

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

Parameters

session	the TLS session structure

6.6.2.6 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

6.6.2.7 runningSyntheticHash()

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

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

Parameters

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

6.6.2.8 initCryptoContext()

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

Initiate a Crypto Context.

Parameters

C	an AEAD encryption context

6.6.2.9 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

6.6.2.10 incrementCryptoContext()

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

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

Parameters

C	an AEAD encryption context

6.6.2.11 createCryptoContext()

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

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

Parameters

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

6.6.2.12 createSendCryptoContext()

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

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

Parameters

session	TLS session structure
TS	the input raw secret

6.6.2.13 createRecvCryptoContext()

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

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

Parameters

session	TLS session structure
TS	the input raw secret

6.6.2.14 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

6.6.2.15 deriveEarlySecrets()

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

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

Parameters

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)

6.6.2.16 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)

6.6.2.17 deriveHandshakeSecrets()

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

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

Parameters

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

6.6.2.18 deriveApplicationSecrets()

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

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

Parameters

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)

6.6.2.19 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

6.6.2.20 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

6.6.2.21 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

6.6.2.22 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

6.6.2.23 createClientCertVerifier()

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

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

Parameters

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

6.7 tls_logger.h File Reference

TLS 1.3 logging.

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

Functions

- void myprintf (char ∗s)

  internal printf function - all output funnels through this function
- void log (int logit, char ∗preamble, char ∗string, unsign32 info, octad ∗O)

  basic logging function
- void logServerHello (int cipher_suite, int pskid, octad ∗PK, octad ∗CK)

  logging the Server hello
- void logTicket (ticket ∗T)

  logging a resumption ticket
- void logEncExt (ee_status ∗e, ee_status ∗r)

  logging server extended extensions responses vs expectations
- void logCert (octad ∗CERT)

  logging a Certificate in standard base 64 format
- void logCertDetails (octad ∗PUBKEY, pktype pk, octad ∗SIG, pktype sg, octad ∗ISSUER, octad ∗SUBJECT)

  logging Certificate details
- void logServerResponse (ret r)

  log client processing of a Server response
- void logAlert (int detail)

  log Server Alert
- void logCipherSuite (int cipher_suite)

  log Cipher Suite
- void logKeyExchange (int kex)

  log Key Exchange Group
- void logSigAlg (int sigAlg)

  log Signature Algorithm

6.7.1 Detailed Description

TLS 1.3 logging.

Author

Mike Scott

6.7.2 Function Documentation

6.7.2.1 myprintf()

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

internal printf function - all output funnels through this function

Parameters

s	a string to be output

6.7.2.2 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)

6.7.2.3 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)

6.7.2.4 logTicket()

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

logging a resumption ticket

Parameters

T	a resumption ticket

6.7.2.5 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

6.7.2.6 logCert()

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

logging a Certificate in standard base 64 format

Parameters

CERT	the certificate to be logged

6.7.2.7 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

6.7.2.8 logServerResponse()

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

log client processing of a Server response

Parameters

r	the Server response

6.7.2.9 logAlert()

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

log Server Alert

Parameters

detail	the server's alert code

6.7.2.10 logCipherSuite()

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

log Cipher Suite

Parameters

cipher_suite	the Cipher Suite to be logged

6.7.2.11 logKeyExchange()

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

log Key Exchange Group

Parameters

kex	the Key Exchange Group to be logged

6.7.2.12 logSigAlg()

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

log Signature Algorithm

Parameters

sigAlg	the Signature Algorithm to be logged

6.8 tls_octads.h File Reference

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

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

Data Structures

- struct octad

  Safe representation of an octad.

Functions

- unsigned long millis ()

  read milliseconds from a stop-watch
- 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.

6.8.1 Detailed Description

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

Author

Mike Scott

6.8.2 Function Documentation

6.8.2.1 millis()

```
unsigned long millis ( )
```

read milliseconds from a stop-watch

Returns

milliseconds read from stop-watch

6.8.2.2 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

6.8.2.3 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

6.8.2.4 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

6.8.2.5 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

6.8.2.6 OCT_kill()

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

Wipe clean an octad.

Parameters

O	octad to be cleared

6.8.2.7 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

6.8.2.8 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

6.8.2.9 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

6.8.2.10 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

6.8.2.11 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

6.8.2.12 OCT_reverse()

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

Reverse bytes in an octad.

Parameters

O	octad to be reversed

6.8.2.13 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

6.8.2.14 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

6.8.2.15 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

6.8.2.16 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

6.8.2.17 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

6.9 tls_protocol.h File Reference

TLS 1.3 main client-side protocol functions.

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

Functions

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

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

  terminate a session structure
- void TLS13_stop (TLS_session ∗session)

  stop sending - send CLOSE_NOTIFY and DISCONNECT

- 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.

6.9.1 Detailed Description

TLS 1.3 main client-side protocol functions.

Author

Mike Scott

6.9.2 Function Documentation

6.9.2.1 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

6.9.2.2 TLS13_end()

```
void TLS13_end (
              TLS_session ∗ session )
```

terminate a session structure

Parameters

session	the session structure

6.9.2.3 TLS13_stop()

```
void TLS13_stop (
            TLS_session * session )
```

stop sending - send CLOSE_NOTIFY and DISCONNECT

Parameters

session	the session structure

6.9.2.4 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

6.9.2.5 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

6.9.2.6 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

6.9.2.7 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

6.10 tls_sal.h File Reference

Security Abstraction Layer for TLS.

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

Functions

- char ∗ SAL_name ()

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

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

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

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

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

  Initialise library for use.
- void SAL_endLib ()

  finish use of library
- int SAL_hashType (int cipher_suite)

  return hash type 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
- bool 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.

6.10.1 Detailed Description

Security Abstraction Layer for TLS.

Author

Mike Scott

6.10.2 Function Documentation

6.10.2.1 SAL_name()

```
char* SAL_name ( )
```

Return name of SAL provider.

Returns

name of SAL provider

6.10.2.2 SAL_ciphers()

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

Return supported ciphers.

Parameters

ciphers	array of supported ciphers in preferred order

Returns

number of supported ciphers

6.10.2.3 SAL_groups()

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

Return supported groups in preferred order.

Parameters

groups	array of supported groups

Returns

number of supported groups

6.10.2.4 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

6.10.2.5 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

6.10.2.6 SAL_initLib()

```
bool SAL_initLib ( )
```

Initialise library for use.

Returns

return true if successful, else false

6.10.2.7 SAL_endLib()

```
void SAL_endLib ( )
```

finish use of library

6.10.2.8 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

6.10.2.9 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

6.10.2.10 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

6.10.2.11 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

6.10.2.12 SAL_randomByte()

```
int SAL_randomByte ( )
```

get a random byte

Returns

a random byte

6.10.2.13 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

6.10.2.14 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

6.10.2.15 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

6.10.2.16 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

6.10.2.17 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

6.10.2.18 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

6.10.2.19 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

6.10.2.20 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

6.10.2.21 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

6.10.2.22 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

6.10.2.23 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

6.10.2.24 SAL_generateSharedSecret()

```
bool 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

Returns

false for all zeros, else true

6.10.2.25 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

6.10.2.26 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

6.11 tls_sockets.h File Reference

set up sockets for reading and writing

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

Data Structures

- class Socket

  Socket instance.

Functions

- int setclientsock (int port, char ∗ip, int toms)

  create a client socket
- int getIPaddress (char ∗ip, char ∗hostname)

  get the IP address from a URL
- void sendOctad (Socket ∗client, octad ∗B)

  send an octad over a socket
- void sendLen (Socket ∗client, int len)

  send a 16-bit integer as an octad 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 octad from a socket

6.11.1 Detailed Description

set up sockets for reading and writing

Author

Mike Scott

6.11.2 Function Documentation

6.11.2.1 setclientsock()

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

create a client socket

Parameters

port	the TCP/IP port on which to connect
ip	the IP address with which to connect
toms	the time-out period in milliseconds

Returns

the socket handle

6.11.2.2 getIPaddress()

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

get the IP address from a URL

Parameters

ip	the IP address
hostname	the input Server name (URL)

Returns

1 for success, 0 for failure

6.11.2.3 sendOctad()

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

send an octad over a socket

Parameters

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

6.11.2.4 sendLen()

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

send a 16-bit integer as an octad to Server

Parameters

client	the socket connection to the Server
len	the 16-bit integer to be encoded as octad and transmitted

6.11.2.5 getBytes()

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

receive bytes over a socket sonnection

Parameters

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

Returns

-1 on failure, 0 on success

6.11.2.6 getInt16()

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

receive 16-bit integer from a socket

Parameters

client	the socket connection to the Server

Returns

a 16-bit integer

6.11.2.7 getInt24()

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

receive 24-bit integer from a socket

Parameters

client	the socket connection to the Server

Returns

a 24-bit integer

6.11.2.8 getByte()

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

receive a single byte from a socket

Parameters

client	the socket connection to the Server

Returns

a byte

6.11.2.9 getOctad()

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

receive an octad from a socket

Parameters

client	the socket connection to the Server
B	the output octad
expected	the number of bytes expected

Returns

-1 on failure, 0 on success

6.12 tls_tickets.h File Reference

TLS 1.3 process resumption tickets.

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

Functions

- 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.

6.12.1 Detailed Description

TLS 1.3 process resumption tickets.

Author

Mike Scott

6.12.2 Function Documentation

6.12.2.1 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

6.12.2.2 initTicketContext()

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

initialize a ticket structure

Parameters

T	the ticket structure

6.12.2.3 endTicketContext()

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

terminate a ticket structure

Parameters

T	the ticket structure

6.12.2.4 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

6.13 tls_wifi.h File Reference

define Socket structure depending on processor context

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

6.13.1 Detailed Description

define Socket structure depending on processor context

Author

Mike Scott

6.14 tls_x509.h File Reference

X509 function Header File.

Data Structures

- struct pktype

  Public key type.

Macros

- #define X509_ECC 1
- #define X509_RSA 2
- #define X509_ECD 3
- #define X509_PQ 4
- #define X509_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

6.14.1 Detailed Description

X509 function Header File.

Author

Mike Scott defines structures declares functions

6.14.2 Macro Definition Documentation

6.14.2.1 X509_ECC

```
#define X509_ECC 1
```

Elliptic Curve data type detected

6.14.2.2 X509_RSA

```
#define X509_RSA 2
```

RSA data type detected

6.14.2.3 X509_ECD

```
#define X509_ECD 3
```

Elliptic Curve (Ed25519) detected

6.14.2.4 X509_PQ

```
#define X509_PQ 4
```

Post Quantum method

6.14.2.5 X509_HY

```
#define X509_HY 5
```

Hybrid Post_Quantum

6.14.2.6 X509_H256

```
#define X509_H256 2
```

SHA256 hash algorithm used

6.14.2.7 X509_H384

```
#define X509_H384 3
```

SHA384 hash algorithm used

6.14.2.8 X509_H512

```
#define X509_H512 4
```

SHA512 hash algorithm used

6.14.2.9 USE_NIST256

```
#define USE_NIST256 0
```

For the NIST 256-bit standard curve - WEIERSTRASS only

6.14.2.10 USE_C25519

```
#define USE_C25519 1
```

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

6.14.2.11 USE_NIST384

```
#define USE_NIST384 10
```

For the NIST 384-bit standard curve - WEIERSTRASS only

6.14.2.12 USE_NIST521

```
#define USE_NIST521 12
```

For the NIST 521-bit standard curve - WEIERSTRASS only

6.14.3 Function Documentation

6.14.3.1 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

6.14.3.2 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

6.14.3.3 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)

6.14.3.4 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)

6.14.3.5 X509_extract_cert()

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

Parameters

sc	a signed certificate
c	the extracted certificate

Returns

0 on failure

6.14.3.6 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)

6.14.3.7 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

6.14.3.8 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

6.14.3.9 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

6.14.3.10 X509_self_signed()

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

Parameters

c	an X.509 certificate

Returns

true if self-signed, else false

6.14.3.11 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

6.14.3.12 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

6.14.3.13 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

6.14.3.14 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

6.14.3.15 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

6.14.3.16 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

6.14.4 Variable Documentation

6.14.4.1 X509_CN

```
octad X509_CN
```

Country Name

6.14.4.2 X509_ON

```
octad X509_ON
```

organisation Name

6.14.4.3 X509_EN

```
octad X509_EN
```

email

6.14.4.4 X509_LN

```
octad X509_LN
```

local name

6.14.4.5 X509_UN

```
octad X509_UN
```

Unit name (aka Organisation Unit OU)

6.14.4.6 X509_MN

```
octad X509_MN
```

My Name (aka Common Name)

6.14.4.7 X509_SN

```
octad X509_SN
```

State Name

6.14.4.8 X509_AN

```
octad X509_AN
```

Alternate Name

6.14.4.9 X509_KU

```
octad X509_KU
```

Key Usage

6.14.4.10 X509_BC

```
octad X509_BC
```

Basic Constraints