Design and Verification of the TLS 1.3 Handshake State Machine in LibreSSL

Theo Buehler
tb@openbsd.org

BSDCan – May 18, 2019
TLS Basics

TLS stands for Transport Layer Security. Successor of the SSL (Secure Socket Layer) protocol.

Client wants to connect to a server.

- Establishes a connection
- Negotiates connection with server (TLS handshake)
- Application data
- End of connection

We will take a closer look at the handshake later on.
History

SSL protocol developed in the mid nineties by Netscape.

- **Legacy versions:**
  - SSL 1.0 (never released)
  - SSL 2.0 (1995–2011)
  - SSL 3.0 (1996–2015)
  - TLS 1.0 (1999–2020?)
  - TLS 1.1 (2006–2020?)

- **Current versions:**
  - TLS 1.3 RFC 8446 (2018).
TLS 1.2 vs TLS 1.3

TLS 1.2 is still fine.

TLS 1.3 brings some improvements:

- Legacy algorithms removed
- Better elliptic curve support (no point format negotiation)
- Forward secrecy
- Optimized handshake state machine
- Much more...

Summary: improved cryptography and performance.
The TLS 1.2 handshake takes two full round trips:

1. Client initiates handshake: ClientHello.

This usually takes 300 – 500 milliseconds.
The TLS 1.3 handshake takes only one round trip:

1. The ClientHello includes the client key exchange.
3. Client sends ClientFinished

Takes about 200 – 350 milliseconds.
Visualization

CLIENT HELLO

SERVER HELLO

SERVER CERTIFICATE

SERVER CERTIFICATE_VERIFY

SERVER FINISHED

CLIENT FINISHED

APPLICATION DATA
Basic design of the state machine

Fundamental observation: while the RFC’s state machine has a loop, it can be modeled on a directed acyclic graph (DAG).

Therefore it is possible to enumerate all legal paths.

Linearized in a static table.

Design based on s2n’s state machine.
Main property

At every point we know what message comes next and we can call a specialized handler.

Almost... After the encrypted extensions there is no way to know whether the server will send a certificate request or a certificate message. Needs an ugly workaround.

By design, we are safe from out-of-order messages (cf. libssh).
The message types

define the following message types:

```
enum tls13_message_type {
    INVALID,
    CLIENT_HELLO,
    SERVER_HELLO,
    CLIENT_HELLO_RETRY,
    ...
    APPLICATION_DATA,
    TLS13_NUM_MESSAGE_TYPES,
    ...
};
```
The handshake actions

Every message type has an associated action:

```c
struct tls13_handshake_action {
    uint8_t handshake_type;
    uint8_t sender;
    uint8_t handshake_complete;
    uint8_t preserve_transcript_hash;
}
```

```c
int (*send)(struct tls13_ctx *ctx);
int (*sent)(struct tls13_ctx *ctx);
int (*recv)(struct tls13_ctx *ctx);
};
```
Example:

```
[CLIENT_HELLO] = {
  .handshake_type = TLS13_MT_CLIENT_HELLO,
  .sender = TLS13_HS_CLIENT,
  .send = tls13_client_hello_send,
  .recv = tls13_client_hello_recv,
},
```
The handshakes table

```c
enum tls13_message_type_handshakes[]
    [TLS13_NUM_MESSAGE_TYPES] = {
        ...
        [NEGOTIATED | WITHOUT_CR] = {
            CLIENT_HELLO,
            SERVER_HELLO,
            SERVER_ENCRYPTED_EXTENSIONS,
            SERVER_CERTIFICATE,
            SERVER_CERTIFICATE_VERIFY,
            SERVER_FINISHED,
            CLIENT_FINISHED,
            APPLICATION_DATA,
        },
        ...
    }
```
Advancing the state machine

Simply increment a value:

```c
int tls13_handshake_advance_state_machine (struct tls13_ctx *ctx)
{
    if (++ctx->handshake_stage.message_number
        >= TLS13_NUM_MESSAGE_TYPES)
        return 0;

    return 1;
}
```
Regress tests

The handshakes table is generated by code in the regress test.

```
$ cd /usr/src/regress/lib/libssl/handshake
$ make print
```

Tests are run daily by bluhm@ on his regress machines.
Visualization

Regress target to generate graphics in various formats. Uses the math/graphviz package (thanks, edd®).

$ cd /usr/src/regress/lib/libssl/handshake
$ doas pkg_add graphviz
$ make handshake.png
References

- RFC 8446
- A Detailed Look at RFC 8446
- Source code: lib/libssl, regress/lib/libssl/handshake
- Tweet thread by Colm MacCárthaigh
Status of TLS 1.3 in LibreSSL

About 60% there.

The client side is mostly done

Work on the server side has not yet started
Thanks

The entire OpenBSD team. In particular:

- daniel@
- deraadt@
- beck@
- bcook@
- inoguchi@
- jsing@
- bluhm@
- schwarze@
- sthen@

Finally, I would like to thank my employer, ARCATrust SA, for supporting part of my work on TLS 1.3.