TLS 1.3 Status

Eric Rescorla

ekr@rtfm.com

Overview

- Review of changes
- Overview of 1RTT handshake
- 1RTT open issues

Changes since -01

- Increment version number.
- Removed support for compression.
- Removed support for static RSA and DH key exchange.
- Removed support for non-AEAD ciphers
- Remove custom DHE groups.*
- Reworked handshake to provide 1-RTT mode.*

- * More on following slides

1RTT Assumptions

- Client can make a good guess at server groups
  - (Hence forbidding custom groups)
- Defer SNI encryption
  - But don't deal with it just yet (see this afternoon)
- Encrypt as much of handshake as possible

Overall 1RTT Flow

```
ClientHello
ClientKeyExchange            -------->
                                                ServerHello
                                          ServerKeyExchange
                                         [ChangeCipherSpec]
                                    {EncryptedExtensions*}
                                             {Certificate*}
                                      {CertificateRequest*}
                                       {CertificateVerify*}
                             <--------           {Finished}
[ChangeCipherSpec]
{Certificate*}
{CertificateVerify*}
{Finished}                   -------->
Application Data             <------->     Application Data
```

New `ClientKeyExchange`

- Client can provide an arbitrary number of (EC)DHE shares
- Each corresponds to a single potential group
  - Only one (EC)DHE share per group
- MUST be independently generated

New ClientKeyExchange Syntax

```
enum { dhe(1), (255) } KeyExchangeAlgorithm;

struct {
    KeyExchangeAlgorithm algorithm;
    select (KeyExchangeAlgorithm) {
       dhe:
           ClientDiffieHellmanParams;
    } exchange_keys;
} ClientKeyExchangeOffer;

struct {
    ClientKeyExchangeOffer offers<0..2^16-1>;
} ClientKeyExchange;

struct {
    DiscreteLogDHEGroup group;  // from draft-gillmor
    opaque dh_Yc<1..2^16-1>;
} ClientDiffieHellmanParams;
```

Should we be renaming this message (WTC)

- Very different syntax from current CKE
  - You'll need different code in any case
  - We've got plenty of code points
- Though serves the same purpose
  - What will we call it, `ClientKeyExchange2`?
- Proposal: ???

`https://github.com/tlswg/tls13-spec/issues/58`

Extension handling

- All client extensions are in the clear as before
- Server extensions are split
  - Extensions needed to establish cryptographic parameters go in `ServerHello`
  - All other extensions go in `EncryptedExtensions`
- Currently `EncryptedExtensions` override other extensions
- Proposal
  - Each extension MUST identify where it goes (default is encrypted)
  - Misplaced extensions generate an error

`https://github.com/tlswg/tls13-spec/issues/66`

Revised `ServerKeyExchange`

- The original `ServerKeyExchange` carried the server parameters and a signature
- Parameters are now in the ECC or `draft-gillmor` extensions
- Signature moved to cover whole server flight

New `ServerKeyExchange` Syntax

```
struct {
    opaque dh_Ys<1..2^16-1>;
} ServerDiffieHellmanParams;     /* Ephemeral DH parameters */

struct {
    select (KeyExchangeAlgorithm) {
        case dhe:
            ServerDiffieHellmanParams;
        /* may be extended, e.g., for ECDH -- see [RFC4492] */
    } params;
} ServerKeyExchange;
```

- No need to identify parameters, since they are negotiated before

What about the server's signature?

- It's now in `CertificateVerify`
- This needs to be the last message so it covers the entire handshake
- Improves commonality between client and server

Backward Compatibility

- You can't put extra handshake messages in the first message flight
  - Breaks old TLS implementations
- Instead stuff them in an extension

`EarlyData` **Syntax**

```
struct {
  TLSCipherText messages<5 .. 2^24-1>;
} EarlyDataExtension;
```

- Note that these are *TLS Records*
- Overkill for now but will be useful for 0-RTT
  - Since we can carry `application_data`

What if the client guesses wrong?

```
ClientHello
ClientKeyExchange            -------->
                             <--------          ServerHello

ClientHello
ClientKeyExchange            -------->
                                                ServerHello
                                          ServerKeyExchange
                                         [ChangeCipherSpec]
                                    {EncryptedExtensions*}
                                             {Certificate*}
                                      {CertificateRequest*}
                                       {CertificateVerify*}
                             <--------           {Finished}
[ChangeCipherSpec]
{Certificate*}
{CertificateVerify*}
{Finished}                   -------->
Application Data             <------->     Application Data
```

- The last half of this is the same as the normal handshake (consensus from Denver)

How does client distinguish these two handshakes?

- Current model
  - Compare the ciphersuite/group to your CKE
  - If no match, then you need to try again
- Other options
  - Have some explicit rejection indicator
  - Add a new message type, though it is pretty much going to have the same contents.

`https://github.com/tlswg/tls13-spec/issues/57`

Interaction with Triple Handshake Fix

- `draft-bhargavan-tls-session-hash-00` specifies computing the master keys from the handshake transcript
- But at time of key computation server and client certificate have not yet been sent
  - However, transcript would cover both DHE shares
- This is inherent in encrypting the certificates*
  - Since you need to have keys before they are sent
- Needs analysis

- Proposal: Postpone till we know about removing renegotiation

*Though we could compute two sets of keys

Other issues?