TLS Working Group E. Rescorla
Internet-Draft Mozilla
Intended status: Standards Track N. Sullivan
Expires: 8 September 2020 Cloudflare
C.A. Wood
Apple Inc.
7 March 2020

Semi-Static Diffie-Hellman Key Establishment for TLS 1.3
draft-ietf-tls-semistatic-dh-01

Abstract

TLS 1.3 [RFC8446] specifies a signed Diffie-Hellman exchange modelled after SIGMA [SIGMA]. This design is suitable for endpoints whose certified credential is a signing key, which is the common situation for current TLS servers. This document describes a mode of TLS 1.3 in which one or both endpoints have a certified DH key which is used to authenticate the exchange.

Note to Readers

Source for this draft and an issue tracker can be found at https://github.com/ekr/draft-rescorla-tls13-semistatic-dh (https://github.com/ekr/draft-rescorla-tls13-semistatic-dh).

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

DISCLAIMER: This is a work-in-progress draft and has not yet seen significant security analysis. Thus, this draft should not be used as a basis for building production systems.

TLS 1.3 [RFC8446] specifies a signed Diffie-Hellman (DH) exchange modeled after SIGMA [SIGMA]. This design is suitable for endpoints whose certified credential is a signing key, which is the common situation for current TLS servers, which is why it was selected for TLS 1.3.

However, it is also possible - although currently rare - for endpoints to have a credential which is an (EC)DH key. This can happen in one of two ways:

* They may be issued a certificate with an (EC)DH key, as specified for instance in [I-D.ietf-curdle-pkix]

* They may have a signing key which they use to generate a delegated credential [I-D.ietf-tls-subcerts] containing an (EC)DH key.

In these situations, a signed DH exchange is not appropriate, and instead a design in which the endpoint authenticates via its long-

term (EC)DH key is suitable. This document describes such a design modeled on that described in OPTLS [KW16].

This design has a number of potential advantages over the signed exchange in TLS 1.3, specifically:

* If the end-entity certificate contains an (EC)DH key, TLS can operate with a single asymmetric primitive (Diffie-Hellman). The PKI component will still need signatures, but the TLS stack need not have one. Note that this advantage is somewhat limited if the (EC)DH key is in a delegated credential, but that allows for a clean transition to (EC)DH certificates.

* If the endpoint has a comparatively slow signing cert (e.g., P-256) it can amortize that signature over a large number of connections by creating a delegated credential with an (EC)DH key from a faster group (e.g., X25519).

* Because there is no signature, the endpoint has deniability for the existence of the communication. Note that it could always have denied the contents of the communication.

This exchange is not generally faster than a signed exchange if comparable groups are used. In fact, if delegated credentials are used, it may be slower on the client as it has to validate the delegated credential, though the result may be cached.

2. Protocol Overview

The overall protocol flow remains the same as that in ordinary TLS 1.3, as shown below:

```
          Client                                               Server

   Key  ^ ClientHello
   Exch | + key_share*
        | + signature_algorithms*
        | + psk_key_exchange_modes*
        v + pre_shared_key*         -------->
                                                          ServerHello  ^ Key
                                                         + key_share*  | Exch
                                                    + pre_shared_key*  v
                                                {EncryptedExtensions}  ^
Server
                                                {CertificateRequest*}  v
Params
                                                       {Certificate*}  ^
                                                 {CertificateVerify*}  | Auth
                                                           {Finished}  v
                                    <--------     [Application Data*]
        ^ {Certificate*}
   Auth | {CertificateVerify*}
        v {Finished}                -------->
          [Application Data]        <------->      [Application Data]
```

As usual, the client and server each supply an (EC)DH share in their "key_share" extensions. However, in addition, the server supplies a (signed) static (EC)DH share in its Certificate message, either directly in its end-entity certificate or in a delegated credential. The client and server then perform two (EC)DH exchanges:

* Between the client and server "key_share" values to form an ephemeral secret (ES). This is the same value as is computed in TLS 1.3 currently.

* Between the client's "key_share" and the server's static share, to form a static secret (SS).

Note that this means that the server's static secret MUST be in the same group as selected group for the ephemeral (EC)DH exchange.

The handshake then proceeds as usual, except that instead of containing a signature, the CertificateVerify contains a MAC of the handshake transcript, computed based on SS.

3. Negotiation

In order to negotiate this mode, we treat the (EC)DH MAC as if it were a signature and negotiate it with a set of new signature scheme values:

```
enum {
  sig_p256(0x0901),
  sig_p384(0x0902),
  sig_p521(0x0903),
  sig_x52219(0x0904),
  sig_x448(0x0905),
} SignatureScheme;
```

When present in the "signature_algorithms" extension or CertificateVerify.signature_scheme, these values indicate DH MAC with the specified key exchange mode. These values MUST NOT appear in "signature_algorithms_cert".

Before sending and upon receipt, endpoints MUST ensure that the signature scheme is consistent with the ephemeral (EC)DH group in use. Clients MUST NOT advertise signature scheme values that are inconsistent with the "named_groups" extension they offer.

4. Certificate Format

Similar to signing keys, static DH keys are carried in the Certificate message, either directly in the EE certificate, or in a delegated credential. In either case, the OID for the SubjectPublicKeyInfo MUST be appropriate for use with (EC)DH key establishment. If in a certificate, the key usage and EKU MUST also be set appropriately. See [I-D.ietf-curdle-pkix] for specific details about these formats.

5. Certificate Verify Computation

Instead of a signature, the server proves knowledge of the private key associated with its static share by computing a MAC over the handshake transcript using SS. The transcript thus far includes all messages up to and including Certificate, i.e.:

```
Transcript-Hash(Handshake Context, Certificate)
```

The MAC key, xSS, is derived from SS as follows:

```
xSS = HKDF-Extract(0, SS)
```

The MAC is then computed using the Finished computation described in [RFC8446] Section 4.4, with xSS as the Base Key value. Receivers MUST validate the MAC and terminate the handshake with a "decrypt_error" alert upon failure.

Note that this means that the server sends two MAC computations in the handshake, one in CertificateVerify using SS and the other in

Finished using the Master Secret. These MACs serve different purposes: the first authenticates the handshake and the second proves possession of the ephemeral secret.

6. Client Authentication

Client authentication works similar to that of server authentication described in Section 2. In particular, servers indicate support of semi-static keys by sending one of the relevant SignatureScheme values defined in Section 3 inside the CertificateRequest "signature_algorithms" extension. If applicable, clients reply with a non-empty Certificate message carrying a corresponding certificate with static DH key matching the chosen signature algorithm. Clients then also compute the CertificateVerify message using the procedure of Section 5, over the transcript hash Handshake Context described in [RFC8446], Section 4.4.

If no matching certificate is available, clients send an empty Certificate message as per [RFC8446]; Section 4.4.2.

7. Security Considerations

[[OPEN ISSUE: This design requires formal analysis.]]

This is intended to have roughly equivalent security properties to current TLS 1.3, except for the points raised in the introduction.

Open questions:

* Should semi-static key shares be mixed into the key schedule?

8. IANA Considerations

IANA [SHOULD add/has added] the new code points specified in Section 3 to the TLS 1.3 signature scheme registry, with a "recommended" value of TBD.

9. References

9.1. Normative References

[I-D.ietf-curdle-pkix]
Josefsson, S. and J. Schaad, "Algorithm Identifiers for Ed25519, Ed448, X25519 and X448 for use in the Internet X.509 Public Key Infrastructure", Work in Progress, Internet-Draft, draft-ietf-curdle-pkix-10, 8 May 2018, <http://www.ietf.org/internet-drafts/draft-ietf-curdle-pkix-10.txt>.

[I-D.ietf-tls-subcerts]
Barnes, R., Iyengar, S., Sullivan, N., and E. Rescorla, "Delegated Credentials for TLS", Work in Progress, Internet-Draft, draft-ietf-tls-subcerts-06, 5 February 2020, <http://www.ietf.org/internet-drafts/draft-ietf-tls-subcerts-06.txt>.

[RFC8446] Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.3", RFC 8446, DOI 10.17487/RFC8446, August 2018, <https://www.rfc-editor.org/info/rfc8446>.

9.2. Informative References

[KW16] Krawczyk, H. and H. Wee, "The OPTLS Protocol and TLS 1.3", Proceedings of Euro S"P 2016 , 2016, <https://eprint.iacr.org/2015/978>.

[SIGMA] Krawczyk, H., "SIGMA: the 'SIGn-and-MAc' approach to authenticated Diffie-Hellman and its use in the IKE protocols", Proceedings of CRYPTO 2003 , 2003.

Authors' Addresses

Eric Rescorla
Mozilla

Email: ekr@rtfm.com

Nick Sullivan
Cloudflare

Email: nick@cloudflare.com

Christopher A. Wood
Apple Inc.

Email: cawood@apple.com