

TLS Working Group
Internet-Draft
Intended status: Standards Track
Expires: September 6, 2018

E. Rescorla
Mozilla
N. Sullivan
Cloudflare
March 05, 2018

Semi-Static Diffie-Hellman Key Establishment for TLS 1.3
draft-rescorla-tls13-semistatic-dh-00

Abstract

TLS 1.3 [[I-D.ietf-tls-tls13](#)] 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.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of [BCP 78](#) and [BCP 79](#).

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <http://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on September 6, 2018.

Copyright Notice

Copyright (c) 2018 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to [BCP 78](#) and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in [Section 4.e](#) of

Internet-Draft

TLS 1.3 Semi-Static KX

March 2018

the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1.	Introduction	2
2.	Protocol Overview	3
3.	Negotiation	5
4.	Certificate Format	5
5.	Cryptographic Details	5
5.1.	Certificate Verify Computation	5
5.2.	Key Schedule	6
6.	Client Authentication	6
7.	0-RTT	6
8.	Security Considerations	7
9.	IANA Considerations	7
10.	References	7
10.1.	Normative References	7
10.2.	Informative References	8
	Authors' Addresses	8

[1.](#) Introduction

DISCLAIMER: This is a work-in-progress draft and is currently totally handwavy, so it has not yet seen significant security analysis. It should not be used as a basis for building production systems.

TLS 1.3 [[I-D.ietf-tls-tls13](#)] 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, 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:

- o They may be issued a certificate with an (EC)DH key, as specified for instance in [[I-D.ietf-curdle-pkix](#)]
- o 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 server authenticates via its long-term (EC)DH key is suitable. This document describes such a design modelled on that described in OPTLS [[KW16](#)].

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

- o 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.
- o It is more resistant to random number generation failures on the server because the attacker needs to have both the server's long-term (EC)DH key and the ephemeral (EC)DH key in order to compute the traffic secrets. [Note: [\[I-D.cremers-cfrg-randomness-improvements\]](#) describes a technique for accomplishing this with a signed exchange.]
- o If the server 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).
- o Because there is no signature, the server 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 this operation is cacheable.

[2.](#) Protocol Overview

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

Internet-Draft

TLS 1.3 Semi-Static KX

March 2018

	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 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:

- o 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.

- o 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:

- o Instead of containing a signature, the CertificateVerify contains a MAC of the handshake transcript, computed based on SS.
- o SS is mixed into the key schedule at the last HKDF-Extract stage (where currently a 0 is used as the IKM input).

[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.

[4.](#) Certificate Format

Like 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 [\[I-D.ietf-curdle-pkix\]](#) and [\[\[TBD: P-256, etc.\]\]](#) for specific details about these formats.

[5.](#) Cryptographic Details

[5.1.](#) 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 - SS-Base-Key - is derived from SS as follows:

SS-Base-Key = HKDF-Extract(0, SS)

The MAC is then computed using the Finished computation described in [\[I-D.ietf-tls-tls13\]](#) [Section 4.4](#), with SS-Base-Key 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. [\[\[OPEN ISSUE: Verify that this is OK because neither MAC is computed with the mixed key. At least one version of OPTLS was somewhat like that, however.\]\]](#)

[5.2.](#) Key Schedule

The final HKDF-Extract stage of the TLS 1.3 key schedule has an HKDF-Extract with the IKM of 0. When static key exchange is negotiated, that 0 is replaced with SS, as shown below.

```
...
    Derive-Secret(., "derived", "")
        |
        v
    SS -> HKDF-Extract = Master Secret
        |
        +-----> Derive-Secret(., "c ap traffic",
        |                                     ClientHello...server Finished)
        |                                     = client_application_traffic_secret_0
        |
...

```

[6.](#) Client Authentication

[[OPEN ISSUE]] In principle, we can do client authentication the same way, with the client's DH key in Certificate and a MAC in CertificateVerify. However, it's less good because the client's static key doesn't get mixed in at all. Also, client DH keys seem even further off.

[7.](#) 0-RTT

[[OPEN ISSUE]] It seems like one ought to be able to publish the server's static key and use it for 0-RTT, but actually we don't know how to do the publication piece, so I think we should leave this out for now.

[8.](#) Security Considerations

[[OPEN ISSUE: This is a -00, so the security considerations are kind of sketchy.]]

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

- o There are open questions about how much key mixing we want to do, especially with respect to client authentication.
- o I'm not sure I like the double extract of SS. I've looked it over and the SS-Base-Key and the HKDF-Extract to make the MS should be independent, but I'd like to give it another look-over to see if there is a cleaner way to do it.

9. 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.

10. References

10.1. 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", [draft-ietf-curdle-pkix-07](#) (work in progress), November 2017.

[I-D.ietf-tls-subcerts]

Barnes, R., Iyengar, S., Sullivan, N., and E. Rescorla, "Delegated Credentials for TLS", [draft-ietf-tls-subcerts-00](#) (work in progress), October 2017.

[I-D.ietf-tls-tls13]

Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.3", [draft-ietf-tls-tls13-26](#) (work in progress), March 2018.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", [BCP 14](#), [RFC 2119](#), DOI 10.17487/RFC2119, March 1997, <<https://www.rfc-editor.org/info/rfc2119>>.

10.2. Informative References

[I-D.cremers-cfrg-randomness-improvements]

Cremers, C., Garratt, L., Smyshlyaev, S., Sullivan, N.,
and C. Wood, "Randomness Improvements for Security
Protocols", [draft-cremers-cfrg-randomness-improvements-00](#)
(work in progress), March 2018.

[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