Using Static Diffie-Hellman in TLS 1.3
(Working Draft)

Matthew Green
November 11, 2016

1 Introduction

Unlike earlier versions of TLS, current drafts of TLS 1.3 [4] do not provide support for the RSA handshake, and have instead adopted ephemeral-mode Diffie-Hellman and elliptic-curve (EC) Diffie-Hellman as the primary cryptographic key exchange mechanism used in TLS.

While ephemeral Diffie-Hellman is in nearly all ways an improvement over the TLS RSA handshake, it has a limitation in certain enterprise settings. Specifically, the use of ephemeral (PFS) ciphersuites is not compatible with enterprise network monitoring tools such as Intrusion Detection Systems (IDS) that must passively monitor intranet TLS connections made to endpoints under the enterprise’s control. This includes TLS connections made from enterprise load balancers at the edge of the enterprise network to internal enterprise TLS servers. It does not include TLS connections traveling over the external Internet. Such monitoring is ubiquitous and indispensable in some industries, and loss of this capability may slow adoption of TLS 1.3 in these settings.

This document describes an optional configuration for TLS servers that allows the server to use a static Diffie-Hellman secret for all TLS connections terminated by the server. Passive monitoring of TLS connections can be enabled by installing a corresponding copy of this key in each monitoring device. This document expands upon a proposed Internet Draft currently in submission to the TLS WG [3].

An advantage of this proposal is that it can be implemented using software modifications to the TLS server only, without the need for any change in TLS client implementations.

2 Background: TLS 1.3

The current TLS specification [4] provides three different mechanisms for exchanging keys: pre-shared key (PSK), Diffie-Hellman (either Finite-Field DHE or Elliptic Curve DHE), and a combination of PSK and DHE/ECDHE. In this document we focus exclusively on the use of DHE and ECDHE without a pre-shared key.

An overview of the full TLS handshake (from [4]) appears in Figure 2. A brief summary of the handshake procedure is expressed below:

1. The client generates one or more ephemeral public and private key pairs, one for each DH group supported by the client.\footnote{A list of supported FF-DH and EC-DH groups can be found in [4, §4.2.4].}
2. The client transmits each public key within a KeyShare data structure in the ClientHello message, along with a random nonce (ClientHello.random).

3. The server selects one group specified by the client and generates an ephemeral public and private key, then transmits the public key within a KeyShare message, along with a random nonce (ServerHello.random). If the server cannot identify a valid group in the ClientHello message, it may transmit a HelloRetryRequest specifying a different group. In response to this message, the client sends a new ClientHello with an appropriate key share.

4. The two parties validate the other party’s key share, and calculate a shared (EC/FF) Diffie-Hellman secret by combining the other party’s ephemeral public key with their own ephemeral secret.

5. A series of traffic and handshake session keys is derived by combining this shared secret with various inputs from the handshake, including the ClientHello.random and ServerHello.random. The ephemeral private keys are deleted.

6. Data encryption is performed using these session keys.

**Structure of keypairs.** A list of Diffie-Hellman groups can be found in [4, 2]. When using Finite-Field Diffie-Hellman, each group consists of a tuple of integers \((g, q, p)\), where \(p\) is prime and \(q = p - 1\) or \(q = (p - 1)/2\). Each ephemeral keypair has the form \(sk = x \in [0, q), pk = g^x \mod p\).

When using EC Diffie-Hellman, each group is an elliptic curve with a generator \(P\) and order \(q\), and each keypair has the form \(sk = x \in [0, q), pk = xP\).

### 3 Introducing static Diffie-Hellman Keys on the Server

The proposal discussed in this document and in [3] does not change the interoperability of TLS clients and servers. Instead, it alters the operation of the server in a manner that retains compati-
bility with all TLS clients. The proposal modifies the standard TLS 1.3 handshake as follows:

1. The server will be configured with one or more static Diffie-Hellman secret keys (one for each supported group). These keys may be generated at server installation, or they may be delivered to the server using a secure key distribution mechanism.

2. The server will at this time generate an appropriate Diffie-Hellman public key.

3. During all subsequent TLS 1.3 handshakes, the server will deliver the appropriate public Diffie-Hellman key inside of the server’s KeyShare data structure.

In summary, this modification changes the operation of the Diffie-Hellman handshake to derive a series of keys based on a single server secret. A consequence of this modification is that a middlebox with knowledge of the secret key can now passively examine the transcript of a TLS handshake between a client and a device within the enterprise and can derive the various encryption keys used by TLS to encrypt the handshake and application data. This allows for passive decryption of TLS 1.3 connections, but only in situations where the server and passive monitoring device are controlled by the same organization.

4 Security implications

We now discuss the security implications of our proposed change. First, we note that TLS (versions 1.2 and earlier) have long supported the use of static Diffie-Hellman keys on the server. The use of a static Diffie-Hellman secret as a configuration option effectively revives a long-supported configuration for the TLS protocol.

However, the use of this configuration has several implications, which we describe below.

4.1 Loss of Perfect Forward Secrecy

A key implication of using static Diffie-Hellman on the server is a loss of the Forward Secrecy (PFS) property provided by standard ephemeral Diffie-Hellman handshakes. The PFS property ensures that even after a compromise of a server that leads to theft of the server’s long-term private key material, an attacker should not be able to decrypt past sessions recorded on the wire. In ephemeral Diffie-Hellman this property is ensured by deleting the server’s ephemeral private keys following each handshake, ensuring that the server retains no long-term key material that could enable passive decryption.

If the server instead uses a static Diffie-Hellman key, this forward secrecy property is not preserved. An attacker who compromises a server (and thus the private key) will be able to decrypt recorded past connections. This tradeoff is inherent in the use of any non-PFS static ciphersuite. Indeed, the same limitation exists in the TLS 1.2 RSA handshake. Thus, the use of static Diffie-Hellman keys on the server does not produce any weakening of security when compared to existing TLS connections used in the data center.

4.2 Security on re-use of client keys

In this section we briefly provide an intuitive argument for the security of the configuration. This can be considered in terms of two separate cases: security with a correct client, and security with an incorrect client.
Security with an correct client. A correct TLS 1.3 client implementation will generate a unique key share for each TLS connection, and deliver this key share along with a random ClientHello.random nonce. As a consequence, if $s$ is the server static secret key and $c$ the client ephemeral private key, then the Diffie-Hellman shared secret $g^{sc}$ will be unpredictable to an attacker.

Security with an incorrect client. Some TLS 1.3 client implementations may choose to re-use Diffie-Hellman secret keys across multiple TLS handshakes. This will result in the same Diffie-Hellman shared secret $g^{sc}$ being used across multiple connections. However, due to the nature of TLS key derivation, as long as the ClientHello.random and ServerHello.random are not both repeated exactly during these connections, the Master Secret and session keys will be distinct between the connections. Thus, it is critical that servers employing static Diffie-Hellman configurations ensure the use of strong randomness in generating each ServerHello.random nonce.

4.3 Replay attacks

A serious concern in the TLS protocol is the possibility that an attacker will replay portions of a legitimate client or server transcript. One hypothetical danger of a replay attack against a static Diffie-Hellman server is that a replayed client connection may result in the derivation of identical encryption keys between two sessions. When these keys are used with a symmetric encryption mode such as GCM, this could lead to keystream re-use and other serious vulnerabilities.

Fortunately, TLS 1.3 provides prevents replay attacks by using a random ServerHello.random nonce in each TLS handshake. This enforces (with overwhelming probability) the use of distinct encryption keys and Finished messages even when client data is replayed. This ensures that handshakes will fail to complete in this case, and security failures will not occur.

4.4 Private key leakage

One potential pitfall of re-using a single static private key for long-term use is the possibility that the key, or portions of the key, might become compromised through a cryptographic vulnerability or a side-channel vulnerability. For example, a small subgroup attack [5] on Diffie-Hellman could reveal bits of the static key, leading to system compromise. Similarly, side channel vulnerabilities could result in the loss of key information over time.

Cryptographic vulnerabilities. To mitigate the threat of cryptographic vulnerabilities, we recommend using a distinct TLS static private key with each Diffie-Hellman group. Furthermore, we stress the importance of performing standard tests against the Diffie-Hellman share provided by a potential adversary, e.g. ensuring that the share is an element in the appropriate cyclic group. These tests are described in FIPS documents [1].

Side channel attacks. Many modern TLS implementations incorporate mechanisms designed to resist side channel attacks such as remote timing attacks and cache timing attacks. These mechanisms aim to eliminate leakage by avoiding detectable timing differences that are caused by secret data. When using a static Diffie-Hellman configuration, implementers should aim to use these implementations.
5 Key Distribution

For large-scale deployments using the static Diffie-Hellman configuration, it is desirable to use an automated system to distribute static secret keys to devices. A number of commercial solutions currently exist for this purpose in the TLS-RSA setting. These devices can be altered to support delivery of static Diffie-Hellman secrets.

The main challenge in the distribution of secrets is the need to distribute one secret for each FF and EC Diffie-Hellman group supported by the server. To minimize the cost of this solution, the server may be configured to support a relatively small set of groups. Alternatively, the key distribution system may be used to deliver a single e.g., 256-bit seed $S$ that can be pluralized into many distinct Diffie-Hellman keys using a key derivation mechanism. For a given group identifier $GID$, this key derivation mechanism may be constructed by calculating a function $F(S, GID, q) \rightarrow x$ with the property that $x$ is indistinguishable from a random element in $[0, q)$.

6 Acknowledgements

This modification to TLS was initially suggested by Hugo Krawczyk.

References


