Abstract

This specification defines a Transport Layer Security (TLS) extension which allows endpoints to negotiate the use of encryption with forward secrecy at the beginning of the handshake. Two levels of functionality are defined. Implementations are free to support one or both levels, with the first level incurring no additional computational or round-trip overhead. The TLS cryptographic calculations are unchanged.

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 November 03, 2012.

Copyright Notice

Copyright (c) 2012 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 the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents
1. Introduction

Although TLS [RFC 5246] provides a variety of cryptographic options for securing application data it exposes an unnecessarily large amount of plaintext during its initial handshake. The server’s identity, session resumption data, and session ticket data are all sent in the clear. When client cert authentication is used, the client’s identity and the server’s list of trusted root CAs is also exposed. Typically the leaked information is sufficient to allow a passive observer to fingerprint the TLS client and server applications in use, and even to learn something about the connection history of the user. This is arguably not a characteristic of an ideal data security protocol.

Currently, the only defined way to fully encrypt a TLS handshake is to first perform an unencrypted TLS handshake as an anonymous client and server and then immediately renegotiate the connection all over again in the new encrypted channel. The overhead this imposes is significant, effectively doubling the number of (relatively expensive) public key operations and requiring several additional round trips. Furthermore, implementations often completely forbid both the initial anon-anon TLS connection as well as client-initiated renegotiation. Consequently, renegotiation is of limited use to endpoints desiring greater protection for their handshake data.

The amount of interesting information sent in the clear will continue to increase as new protocol features and applications are enabled by new TLS extensions. But some interesting new applications may be limited by the lack of encryption during the initial handshake.
In the unmodified TLS handshake, encryption of the record layer begins seemingly as late in the process as possible. The reasons for this design decision are unclear but it seems likely to be an artifact of the historical under-use of ephemeral Diffie-Hellman (DHE) cipher suites at the time (SSLv2 did not support them at all). At the time, the need for the additional computation required by traditional Diffie-Hellman key exchange was considered expensive and for the RC4 cipher (very popular for its efficiency) DHE cipher suites have never even been defined. Before the Server Name Indication (SNI) extension [RFC 6066], it was not possible to serve multiple HTTPS sites from a single IP address, so the specific site to which the user was connecting was effectively already known. There were no TLS extensions to carry interesting plaintext, so there was little interesting plaintext to protect (client certs being a notable exception).

Today we have a greater amount of interesting information present in the handshake, faster hardware, and the far more efficient Elliptic Curve DH algorithms. Some very high-traffic sites are configured to select (EC)DHE cipher suites whenever the option is presented by the client.[XXX cite] There are new extensions being proposed that would benefit from encryption in the handshake.[XXX cite NP(N)] So this appears to be an appropriate time to introduce early encryption via ephemeral key exchange into the handshake.

1.1. Requirements Language

XXX reference presentation language in 5246 here?

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119].

2. Functional Overview

The Encrypted Handshake (EH) extension allows endpoints to perform a key exchange and begin record layer encryption as early in the handshake as possible. Two levels of protection provide flexibility for the client and server (and the implementation) to manage practical considerations and level of effort.

While in general it is not possible to protect pre-authentication data from an active man-in-the-middle type attacker, this extension hides the bulk of the handshake data from a passive observer and can detect active attacks as a failed handshake.

Level one provides encryption with forward secrecy for all data in the Server Hello message (including Server Hello extensions) and all data following. This level incurs no additional computational or round-trip overhead over the traditional handshake and is intended to require minimal changes to the implementation of existing libraries, but it has the limitation that data sent by the client in the Client Hello continues to be sent in the clear.
The second level of implementation encrypts all the same handshake data as the first and additionally encrypts the most interesting parts of the Client Hello (e.g. the extensions), but it requires an additional round trip to the server.

Note that in the following diagrams, an asterisk (*) is used to indicate a optional or situation-dependent messages, while square brackets are placed around the [ChangeCipherSpec] message to reflect its distinct record type.

For reference, the traditional TLS full handshake without Encrypted Handshake looks like this:

```
Client                                                Server
ClientHello                  -------->        ServerHello
                                      Certificate*
                                      ServerKeyExchange*
                                      CertificateRequest*
                                        <-------- ServerHelloDone
Certificate*
ClientKeyExchange
CertificateVerify*
[ChangeCipherSpec]
Finished                     -------->        [ChangeCipherSpec]
                                        <-------- Finished
Application Data             <------->     Application Data

Figure 1. Unencrypted TLS full handshake.
```

When the use of EH level one has been negotiated, the ServerHello message is split to allow the ChangeCipherSpec record to be sent as soon as possible:

```
Client                                                Server
ClientHello (with EH ext)    -------->        ServerHello2a
                                      [ChangeCipherSpec]
                                      ServerHello2b
                                      Certificate*
                                      ServerKeyExchange*
                                      CertificateRequest*
                                        <-------- ServerHelloDone
[ChangeCipherSpec]
Certificate*
ClientKeyExchange
CertificateVerify*
Finished                     -------->        Finished
Application Data             <------->     Application Data
```

Ray Expires November 03, 2012
When the use of EH level two has been negotiated, the ServerHello message is split and the ChangeCipherSpec record is sent early (as in level one). The protocol then performs an additional round-trip in order to allow the complete ClientHello2 to be transmitted within the encrypted channel:

Client                                               Server
ClientHello (with EH ext) --------> ServerHello2a
[ChangeCipherSpec]
ClientHello2 --------> ServerHello2b
Certificate*
ServerKeyExchange* CertificateRequest*
<-------- ServerHelloDone
Certificate*
ClientKeyExchange
CertificateVerify*
Finished --------> Finished
Application Data --------> Application Data

Figure 3. TLS full handshake using EH level two.

3. Extension Definition

This document defines a new TLS extension, "encrypted_handshake" (having extension type [TBD]), which both negotiates the level of EH implementation used for this handshake as well as transmits the (EC)DH parameters necessary for early key exchange. This extension is sent in both the client-to-server and server-to-client directions, however server acceptance of the use of EH is also indicated by the presence of a ServerHello2a message in response. The "extension data" field of this extension contains one of two different structures, or is empty, depending on the type of the message carrying it.

Levels of support are indicated by the following enumeration. However, levels higher than are defined here are reserved for future use and receivers should treat them accordingly.

```c
enum {
    zero(0), one(1), two(2), inquire(255)
} EncryptedHandshakeLevel;
```

When the extension is carried on the Client Hello message it contains
an EncryptedHandshakeInfoCH structure:

struct {
    EncryptedHandshakeLevel client_requested;
    EncryptedHandshakeLevel client_required;
    ClientDHParamSet client_dh_params_sets<0..2^16-1>;
    Extension conditional_extensions<0..2^16-1>;
} EncryptedHandshakeInfoCH;

client_requested
    The highest EH level being requested by the client.

client_required
    The lowest EH level acceptable by the client.

client_dh_params_sets
    Contains a set of (EC)DH(E) parameters for all
    KeyExchangeAlgorithms used by any of the cipher_suites specified
    in the ClientHello. The element type is defined below.

conditional_extensions
    ClientHello-type extensions which are sent in the clear, but to be
    processed by the server only if the server interprets the EH
    extension and agrees to EH level one or higher. The definition of
    'Extension' is from [TLS].

When the extension is carried on the Server Hello message it contains
an EncryptedHandshakeInfoSH structure:

struct {
    EncryptedHandshakeLevel server_max_supported;
} EncryptedHandshakeInfoSH;

server_max_supported
    The max EH level provided by the server, for informational
    purposes.

The EH extension MAY be included on the ServerHello2a or
ServerHello2b message, or it may be omitted. If present, it MUST be
empty.

The ClientDHParamSet structure represents a set of (EC)DH parameters
and specifies with which cipher suites they may be used:
struct {
    CipherSuite cipher_suites<2..2^16-2>;
    uint16 params_size;
    select (KeyExchangeAlgorithm) {
        case dh_anon:          // [TLS]
        case dhe_dss:
        case dhe_rsa:
            DHParams params;
        case ec_diffie_hellman:  // [RFC 4492]
            ClientECDiffieHellmanPublic params;
    }
} ClientDHParamSet;

cipher_suites
The set of cipher suite(s) to which this ClientDHParamSet applies.

params_size
Number of bytes to follow in the params field.

params
The set of Diffie-Hellman parameters to be used for the specified cipher suites.

4. ServerHello2a handshake message

This message forms the (unencrypted) first part of the modified Server Hello handshake message when the use of EH level one or level two has been accepted. Its purpose is to transmit to the client the minimum information necessary to complete the key exchange subsequent to a Client Hello which proposed the use of EH.

struct {
    ProtocolVersion server_version;
    Random random;
    CipherSuite cipher_suite;
    EncryptedHandshakeLevel server_accepted;
    CompressionMethod compression_method;
    uint16 params_size;
    select (KeyExchangeAlgorithm) {
        case dh_anon:
        case dhe_dss:
        case dhe_rsa:
            ServerDHPrams params;          // [TLS]
        case ec_diffie_hellman:
            ServerECDHPrams params;        // [RFC 4492]
    }
    Extension extensions<0..2^16-1>;
} ServerHello2a;

server_version, random, compression_method
Defined identically to the corresponding field in the Server Hello message of the TLS version in use.
cipher_suite
Defined identically to the corresponding field in the Server Hello message of the TLS version in use. On a ServerHello2a message, the selected cipher suite MUST be some type of DH cipher suite.

server_accepted
The Encrypted Handshake level accepted by the server for this handshake. On a ServerHello2a message, this field must indicate level one or level two.

extensions
ServerHello-type extensions included on the ServerHello2a handshake message. Most extensions SHOULD sent under encryption on a subsequent message, but if any are needed to establish the cipher state they MUST be included here.

params_size
Number of bytes in the immediately following params field, not including the params_size field itself.

params
Defined identically to the corresponding field in the Server Key Exchange message. However, note that in this structure the parameters are not signed.

5. ServerHello2b handshake message

This message forms the (encrypted) second part of the modified ServerHello handshake message. Its purpose is to transmit to the client the information present in an ordinary ServerHello that was not transmitted in the ServerHello1a.

It is sent by the server subsequent to receiving from the client either:

1. a ServerHello1a accepting EH level one behavior and ChangeCipherSpec, or
2. a ServerHello1a accepting EH level two behavior ChangeCipherSpec, and after the client’s ClientHello2.

struct {
    EncryptedHandshakeLevel server_max_supported;
    SessionID session_id;
    select (extensions_present) {
        case false: 
            struct {};
        case true: 
            Extension extensions<0..2^16-1>;
    };
} ServerHello2b;

server_max_supported
The max EH level provided by the server, for informational purposes.

```
session_id, extensions
Defined identically to the corresponding field in the Server Hello message of the TLS version in use. Ordinarily the EH extension is omitted, must be empty if present.
```

6. ClientHello2 handshake message

This message is used to transmit information under encryption that would have otherwise been sent in the clear in the ClientHello. Certain fields such as protocol_version, client_random, and cipher_suite are always necessary in order to proceed with any type of handshake. However, in anticipation of successful negotiation of EH level two, a client MAY elect to leave the session_id field empty and extensions absent in the ClientHello in order to supply them later under encryption in ClientHello2.

```
struct {
    SessionID session_id;
    select (extensions_present) {
        case false:
        struct {};
        case true:
        Extension extensions<0..2^16-1>;
    }
} ClientHello2;
```

```
session_id, extensions
Defined identically to the corresponding field in the Client Hello message of the TLS version in use. Ordinarily the EH extension is omitted, must be empty if present.
```

7. Session resumption

TLS provides a means to efficiently resume a previously established session without having to perform any asymmetric crypto or cert validation operations (often referred to as an "abbreviated handshake"). Session resumption generally involves the presence of the same Client/ServerHello extensions as a full handshake, so it has equal need of encryption. EH levels one and two are both compatible.
with session resumption.

Traditional TLS session resumption without EH looks like this:

Client                                                Server
ClientHello                   -------->                  ServerHello
                                  [ChangeCipherSpec]
                                  <--------                  Finished
[ChangeCipherSpec]
Finished                    -------->                  Application Data
Application Data              <------->     Application Data

Figure 4. Unencrypted TLS abbreviated handshake.

When the use of EH level one has been negotiated a handshake with
session resumption looks like this:

Client                                                Server
ClientHello (with EH ext)     -------->                  ServerHello2a
                                  [ChangeCipherSpec]
                                  ServerHello2b
                                  <--------                  Finished
[ChangeCipherSpec]
Finished                    -------->                  Application Data
Application Data              <------->     Application Data

Figure 5. TLS abbreviated handshake with EH level one.

With EH level two, the client MAY leave the session_id field empty
when it is transmitted in the clear on the ClientHello, and still
perform resumption based on the session_id field on ClientHello2.
However, if client withholds the session_id value and the server then
declines EH level two session resumption will not be possible. But
because session resumption depends on the client having had recent
prior interaction with the server anyway, this unpredictability is
thought unlikely to become an issue in practice.

Client                                                Server
ClientHello (with EH ext)    -------->                  ServerHello2a
                                  [ChangeCipherSpec]
                                  ServerHello2b
                                  <--------                  Finished
[ChangeCipherSpec]
ClientHello2                 -------->                  Application Data
                                  <--------                  Application Data

Figure 6. TLS abbreviated handshake with EH level two.
XXX TODO possible optimization: the client could indicate that it wants to allow resumption without involving the key exchange. This would seem to leave any ServerHello extensions completely in the clear. But many extensions seem to be ignored or are required to hold identical values on resumption, if they could be shown to be redundant perhaps they could simply be eliminated.

### 8. Client Behavior

The client initiates the connection and begins the handshake using the following sequence, after which the connection generally follows the TLS specification.

1. The client constructs and sends the Client Hello message and EH extension:

   * A client that simply wishes to query a server for its level of support SHOULD send a Client Hello containing an EH extension with a client_required value of ‘inquire’. The client SHOULD set client_requested to the highest level that it supports (which may be zero). Other than this specific case, a client SHOULD NOT set the client_required value higher than the client_requested value.

   * A client wishing not to use EH SHOULD send an EH extension on the Client Hello with the client_requested and client_required fields set to zero. This is semantically equivalent to sending no EH extension at all, except that the Server Hello may contain an EH extension in reply.

   * A client wishing to use level one EH support only SHOULD send an EH extension on the Client Hello with the client_requested field set to one. The client SHOULD set the client_required field to zero or one as appropriate.

   * A client which desires level two EH support but will accept level one as a fallback SHOULD send an EH extension on the Client Hello with the client_requested field set to two and the client_required field set to one. Since the protocol may fall back to level one, the client SHOULD continue send all necessary session_id and extensions on the initial Client Hello.

   * A client which requires level two EH support SHOULD set the client_requested and client_required fields to two. Such a client SHOULD NOT set any session_id or extension data on the
ClientHello, or send extensions other than EH and those necessary for the key exchange and the integrity of the connection. Renegotiation Info [RFC 5746] is an example of such an extension.

2. If a client, requesting level one or higher, anticipates that sensitive data could be returned via some ServerHello extension in reply, and would prefer that extension not be evaluated at all rather than have that reply transmitted in the clear, the client MUST put the requesting ClientHello extension inside the conditional_extensions section of the EH extension instead of putting it in the usual ClientHello extension section.

3. The server’s response to the ClientHello informs the client of the EH level that will be in effect for the remainder of the handshake:

* If the server’s response is a standard Server Hello message, the server is providing EH level zero. If any EH extension is present, it MUST be ignored.

* If the server’s response is a ServerHello2a message, this the handshake is proceeding at EH level one or higher. The client MUST proceed with the handshake at the level indicated by the server_accepted field. (Level zero here represents an unrecoverable protocol error.)

4. Now, depending on the negotiated level:

* If the client required an EH level higher than what was accepted by the server, the client SHOULD continue with the connection at least to the point of validating the server’s Finished message (in order to authenticate this information). As the client continues the handshake it MUST avoid disclosing any potentially interesting info (such as a client certificate). After the server’s Finished message has been received and validated the client MUST close the connection gracefully without transmitting any application data. The remainder of this section does not apply.

* If EH level zero was negotiated (and no higher value was required by the client) the client continues the TLS connection as usual and the remainder of this section does not apply.

* If EH level one or two was negotiated, the client handles the TLS-defined fields and extensions on the ServerHello2a in the usual way. Additionally, the client MUST examine the server-selected cipher suite and confirm that it was (a) a cipher suite the client proposed and (b) a cipher suite for which the client had provided a ClientDHParamSet in the ClientHello EH extension.
5. By combining data from the Client Hello and the ServerHello2a, the client now has everything necessary to generate the key_block.

   * If EH level one is in use the client SHOULD NOT send his own Change Cipher Spec record at this time, even though he could. This somewhat artificial rule is to simply to promote consistency among implementations. However, the client MUST send his CCS and switch to the new cipher state before sending any other handshake records.

   * If EH level two is in use, the client MUST send a Change Cipher Spec record, switch to the new cipher state, and generate and send a ClientHello2 handshake message. The session_id field SHOULD be populated with the appropriate value if it was withheld from the original ClientHello, otherwise it MUST be empty. Likewise, any ClientHello extensions not sent in the original ClientHello (either in the ClientHello extension area or in the EH extension conditional_extensions area) SHOULD be included. The normal rule of at most one of any specific type of extension MUST be followed, but it now applies to the union of all three possible ClientHello-type extension areas.

6. The client receives a ChangeCipherSpec record from the server and the new cipher state becomes active on the server-to-client channel.

7. The client receives a ServerHello2b handshake message. The client MUST interpret the session_id and extensions the usual way, as defined for a normal ServerHello. The client MAY look at the server_max_supported value for informational purposes, but there is no guarantee that this value will remain valid for any length of time. If any EH extension is present, it MUST be ignored.

8. The remainder of the handshake proceeds as usual, with just these annotations:

   * We’ve already received the Change Cipher Spec from the server, so we don’t expect to receive another.

   * In level one, the client will need to send a Change Cipher Spec before his next handshake message, typically a Certificate or Client Key Exchange message. In level two the client will have already sent it.
* When the (EC)DH parameters are received again (at the expected point in a non-EH handshake, e.g. the Server Key Exchange message), they MUST be compared with the server params used in the early key exchange. If the data covered by the server’s signature is not byte-for-byte the same as that used to establish the current effective cipher state, the client MUST treat this condition as it would an invalid signature on the signed data itself. Even if the handshake was anonymous-anonymous, these parameters MUST be compared.

9. Server Behavior

From the server’s perspective, the handshake proceeds as follows:

1. The server receives a Client Hello message from the client and looks for an EH extension:

   * If no EH extension is present, the negotiated EH level is zero and the server continues the TLS connection as usual. The remainder of this section does not apply.

   * If an EH extension is present, the server uses the smaller of the client_requested value and the server’s own maximum supported implementation level as the EH level for the handshake. The server MAY consider the client_required level in this process. For example, a server might claim to only support level zero or one, but could actually use level two if the client “required” it, i.e., threatened to waste the handshake.

2. If the negotiated EH level is zero and an EH extension was present on the Client Hello, the server SHOULD include an EH extension on the Server Hello in reply. The server SHOULD set the server_max_supported field honestly. The server MUST NOT process any extensions in the conditional_extensions. The TLS handshake continues as usual and the remainder of this section does not apply.

3. If the received client_required value is higher than the client_requested value, or it is higher than that supported by the server, the server SHOULD continue the handshake at the smaller of the client_requested and max_server_supported values. In such a case, the server’s goal should be to transmit its Finished message while disclosing a minimum of information. For example, the server SHOULD NOT transmit a nonempty session_id (except to indicate session resumption) and it MUST NOT include extensions other than EH and those necessary for the integrity of the connection. Renegotiation Info is an example of such an extension. The server MUST NOT request client authentication. Immediately after transmitting the Server Finished message the server MUST close the connection gracefully without transferring any application data.
4. At this point, the EH level is known to be at least one. The server selects a cipher suite from those that (a) the server supports, (b) are listed in the Client Hello cipher_suites, (c) permit this type of key exchange, and (c) the client has provided a valid set of (EC)DH parameters in the EH extension. If no cipher suites of this type can be selected, the handshake fails.

5. The server generates its (EC)DH parameters and transmits a ServerHello2a. Most of the fields correspond directly to those of ServerHello, the others carry the server’s (EC)DH params. The server MUST NOT include extensions other than EH and those necessary for the integrity of the connection. Renegotiation Info is an example of such an extension.

6. The server transmits its Change Cipher Spec record and switches to the new cipher state in the transmit direction.

7. If the negotiated EH level is two, the server receives a Change Cipher Spec record from the client, switches to the new cipher state in the receive direction and receives and processes a ClientHello2 message from the client.

8. Any not-yet-processed extensions from the ClientHello and ClientHello2, including those in the conditional_extensions field, should be processed at this time.

9. The server constructs and transmits a ServerHello2b message. The server SHOULD set the server_max_supported field honestly. The server SHOULD include any ServerHello-type extensions in reply that were not already sent on the ServerHello2a.

10. The remainder of the handshake proceeds as usual, with just these annotations:

* In level two, we’ve already received the Change Cipher Spec from the client so we don’t expect to receive another.

* When the (EC)DH parameters are received again (at the expected point in a non-EH handshake, e.g. the Client Key Exchange message), they MUST be compared with the client params used in the early key exchange. If the data covered here is not byte-for-byte the same as that used to establish the current effective cipher state, the server MUST treat this condition as it would an invalid signature on a Certificate Verify or an invalid verify_data in the client’s Finished message. Even if the handshake did not involve client authentication, these parameters MUST be compared.

10. Other Considerations
Implementers of TLS libraries that allow configuration of cipher suites (e.g. a permitted cipher suites list ordered by preference) are encouraged provide a means to configure this list separately for EH-requesting clients. This is to prevent a situation where server support of this extension is restricted by a simple server preference for non-DHE cipher suites.

There are reportedly some defective servers [XXX cite?] that fail in an ungraceful manner if sent a Client Hello message larger than a certain size which is smaller than the defined maximum. Clients wishing to minimize the risk of interoperability with such servers may consider limiting the number of parameter sets they include in the Client Hello, or sending only the smaller ECDH types. It is possible that these noncompliant servers are older and less well-maintained implementations and tend not to support newer features such as ECDH.

11. IANA Considerations

XXX allocation of encrypted_handshake Hello extension
XXX allocation of ServerHello2a handshake message id
XXX allocation of ServerHello2b handshake message id
XXX allocation of ClientHello2 handshake message id

12. Security Considerations

Insofar as TLS is inherently a data security protocol, this entire document is about security. However, a few points of concern are probably worth mentioning.

- It cannot be repeated often enough that the early encryption negotiated by the EH extension *only* provides protection from passive eavesdropper. It *can not* resist a man-in-the-middle type active attacker who wishes to steal a sample of the plaintext the client and server intended to exchange (doing so will break the handshake however). Like TLS without EH, full protection from an active attacker only begins after the Finished messages are received and validated by each side.

- It could happen that the handshake encryption so obvious in a packet capture will give users a false sense of security. The difference between unauthenticated encryption and an authenticated (and properly bound) encrypted channel is a subtle distinction even to protocol professionals. We must be vigilant to inform users of this extension of these limits and to not rely on it for protections beyond what it can deliver.
This extension is not expected to introduce any significant new computational Denial of Service (DoS) vulnerabilities. TLS without this extension allows an attacker to obligate the server to three or more useless asymmetric crypto operations (ephemeral key exchange, server DH parameter signing, client cert signature verification, and possibly client cert signature chain validation) with no significant computation required on the part of the attacker. This extension adds no new crypto operations to an existing DHE handshake. In fact, an attacker who negotiates its use finds himself now obligated to perform an asymmetric crypto operation in order to proceed further.

However, EH level two introduces an additional round trip from server to client and back. Under conditions of severe packet loss (e.g., during a DDoS event) these additional handshake packets may translate into a reduced success rate for clients. Facing a worst case scenario, a server operator may feel desperate to reduce load by disabling any excess features. Forward secrecy and client privacy being relatively abstract benefits, in the face of a very concrete service outage features such as Diffie-Hellman and EH level two will be tempting targets.

Although this extension does not modify the TLS cryptographic calculations, it does change the order of certain messages. The author does not (currently :-) believe that this introduces any new weaknesses (and it may even mitigate some) but this is certainly in need of careful expert review.  

13. References


Author’s Address
Marsh Ray
PhoneFactor, Inc.
7301 W 129th Street
Overland Park, KS 66213
USA

Email: marsh@extendedsubset.com