TLS Working Group
INTERNET-DRAFT
Expires May 16, 2001

Simon Blake-Wilson, Certicom
Magnus Nystrom, RSA Security
November 17, 2000

Wireless Extensions to TLS
<draft-ietf-tls-wireless-00.txt>

Status of this Memo

Abstract

This document suggests extensions to TLS designed to make TLS more amenable to use within wireless environments. The extensions may be used by TLS clients and servers. The extensions are backwards compatible - communication is possible between TLS 1.0 clients that support the extensions and TLS 1.0 servers that do not support the extensions, and vice versa.

The document suggests extensions of two types: generic extension mechanisms for the TLS client and server hellos, and specific extensions using these generic mechanisms. It is hoped that the structure of the document will allow each suggested extension to be evaluated independently.

This document is based on discussions at the TLS working group meeting during the Pittsburgh IETF meeting, and on discussions within the WAP security group.

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.

Please send comments on this document to the TLS mailing list.

Table of Contents

1. Introduction

Wireless environments often suffer from a number of constraints not commonly present in wired environments - these constraints may include bandwidth limitations, computational power limitations, memory limitations, and battery life limitations.

Use within wireless environments was not one of the initial design criteria of the TLS protocol. As a result, implementations of TLS within wireless environments face a number of challenges.

This document specifies extensions to the TLS 1.0 protocol designed to make TLS more amenable to use within wireless environments. The extensions described here focus on extending the functionality provided by the TLS protocol message formats. Other issues, such as the addition of "wireless-friendly" cipher suites, are deferred.

Specifically, the extensions described in this document are designed to:

- Allow TLS clients and servers to negotiate the maximum record size to be sent. This functionality is desirable as a result of memory constraints common among wireless clients, and bandwidth constraints common among wireless networks.

- Allow TLS clients and servers to negotiate the use of client certificate URLs. This functionality is desirable in order to conserve memory on wireless clients.

- Allow TLS clients to indicate to TLS servers which CA root keys they possess. This functionality is desirable in order to prevent multiple handshake failures involving TLS clients which are only able to store a small number of CA root keys due to memory limitations.

- Encourage TLS servers to use small session identifiers. This functionality is desirable in order to conserve memory on wireless clients.

- Allow TLS clients and servers to negotiate the use of truncated MACs. This functionality is desirable in order to conserve bandwidth in wireless networks.

- Allow TLS clients and servers to negotiate that the server sends the client an OCSP response during a TLS handshake. This functionality is desirable in order to avoid sending a CRL over a wireless network and therefore save bandwidth.

In order to support the extensions above, general extension mechanisms for the client hello message and the server hello message are introduced.

The extensions described in this document may be used by TLS 1.0 clients and TLS 1.0 servers. The extensions are designed to be backwards compatible - meaning that TLS 1.0 clients that support the extensions can talk to TLS 1.0 servers that do not support the extensions, and vice versa.

Backwards compatibility is primarily achieved via two considerations:

- Clients typically request the use of extensions via the extended client hello message described in Section 2.1. TLS 1.0 [TLS] requires servers to "accept" extended client hello messages, even if the server does not "understand" the extension.

- For the specific extensions described here, no mandatory server response is required when clients request extended functionality.

Note however, that although backwards compatibility is supported, some wireless clients may be forced to reject communications with servers that do not support the extensions as a result of the limited capabilities of the wireless clients.

The remainder of this document is organized as follows. Section 2 describes general extension mechanisms for the client hello and server hello handshake messages. Section 3 describes specific extensions to TLS 1.0. The final sections of the document address IPR, security considerations, acknowledgements, and references.

2. General Extension Mechanisms

This section presents general extension mechanisms for the TLS handshake client hello and server hello messages.

These general extension mechanisms are necessary in order to enable clients and servers to negotiate whether to use specific extensions, and how to use specific extensions. The extension formats described are based on [MAILING LIST].

Section 2.1 specifies the extended client hello message format, Section 2.2 specifies the extended server hello message format, and Section 2.3 describes the actual extension format used with the extended client and server hellos.

2.1. Extended Client Hello

The extended client hello message format MAY be sent in place of the client hello message format when clients wish to request extended functionality from servers. The extended client hello message format is:

```
struct {
    ProtocolVersion client_version;
    Random random;
    SessionID session_id;
    CipherSuite cipher_suites<2..2^16-1>;
    CompressionMethod compression_methods<1..2^8-1>;
    Extension client_hello_extension_list<0..2^16-1>;
} ClientHello;
```

Here the new "client_hello_extension_list" field contains a list of extensions. The actual "Extension" format is defined in Section 2.3.

In the event that clients request additional functionality using the extended client hello, and this functionality is not supplied by the server, clients MAY abort the handshake.

Note that TLS, Section 7.4.1.2, allows additional information to be added to the client hello message. Thus the use of the extended client hello defined above should not "break" existing TLS 1.0 servers.

2.2. Extended Server Hello

The extended server hello message format MAY be sent in place of the server hello message when the client has requested extended functionality via the extended client hello message specified in Section 2.1. The extended server hello message format is:

```
      struct {
          ProtocolVersion server_version;
          Random random;
          SessionID session_id;
          CipherSuite cipher_suite;
          CompressionMethod compression_method;
          Extension server_hello_extension_list<0..2^16-1>;
      } ServerHello;
```

Here the new "server_hello_extension_list" field contains a list of extensions. The actual "Extension" format is defined in Section 2.3.

Note that the extended server hello message is only sent in response to an extended client hello message. This prevents the possibility that the extended server hello message could "break" existing TLS 1.0 clients.

2.3. Hello Extensions

The extension format for extended client hellos and extended server hellos is:

```
          struct {
              ExtensionType extensionType;
              opaque unknown_extension<0..2^16-1>;
          } Extension;
```

Here:

- "extensionType" identifies the particular extension type.

- "unknown_extension" contains information specific to the particular extension type.

The extension types defined in this document are:

```
          enum {
                reserved(0), max_record_size(1),
                client_certificate_url(2), trusted_key_ids(3),
                truncated_MAC(4), status_request(5), (65535)
          } ExtensionType;
```

Note that for all the extension types defined in this document, the extension type should appear in the extended server hello only if the same extension type appeared in the corresponding client hello. Thus clients MUST abort the handshake if they receive an extension type in the extended server hello that they did not request in the associated (extended) client hello.

Also note that when multiple extensions are present in the extended client hello or the extended server hello, the extensions must appear in the order identified in "ExtensionType". Thus clients and servers MUST abort the handshake if they receive an extended hello message in which the extensions are not in the correct order.

Finally note that it is possible to alternatively define "unknown_extension" using a selection on "extensionType" (instead of just defining it as type "opaque"). This approach appears to have pros and cons - using a selection is more restrictive and thus less prone to implementation errors, but using "opaque" ensures inclusion of the length and thus ensures that users can skip extensions they don't understand. We would particularly welcome comments on this issue.

3. Wireless Extensions

This section describes the specific TLS extensions specified in this document.

Note that any messages associated with these extensions that is sent during the TLS handshake MUST be included in the hash calculations involved in "Finished" messages.

Section 3.1 describes the extension of TLS to provide maximum record size negotiation. Section 3.2 describes the extension to allow client certificate URLs. Section 3.3 describes the extension to allow clients to indicate which CA root keys they possess. Section 3.4 describes the extension to restrict the size of session identifiers. Section 3.5 describes the extension to allow the use of truncated MACs. Section 3.6 describes the extension to support integration of OCSP into TLS handshakes.

3.1. Maximum Record Size Negotiation

TLS specifies a fixed maximum record size of 2^14 bytes. It may be desirable for wireless clients to negotiate a smaller maximum record size due to memory limitations or bandwidth limitations.

In order to negotiate smaller maximum record sizes, clients MAY include an extension of type "max_record_size" in the (extended) client hello. The "unknown-extension" field of this extension shall contain:

```
enum{
  2^8(1), 2^9(2), 2^10(3), 2^11(4), 2^12(5), (255)
} Negotiated_max_record_size ;
```

whose value is the desired maximum record size. The allowed values for

this field are: 2^8, 2^9, 2^10, 2^11, and 2^12.

Servers that receive an extended client hello containing a "max_record_size" extension, MAY accept the requested maximum record size by including an extension of type "max_record_size" in the (extended) server hello. The "unknown_extension" field of this extension shall contain "Negotiated_max_record_size" whose value is the same as the requested maximum record size.

Servers receiving maximum record size negotiation requests for values other than the allowed values MUST abort the handshake. Similarly, clients receiving maximum record size negotiation responses that differ from the size they requested MUST also abort the handshake.

Once a maximum record size other than 2^14 has been successfully negotiated during a TLS handshake, both the client and server pass the negotiated maximum record size value to the TLS record layer along with the negotiated security parameters. (Note that it may be desirable in the future to update the TLS security parameters to include the maximum record size value.) During the subsequent session after exchange of change cipher spec messages, the client and server MUST ensure that no messages larger than the negotiated size are sent.

3.2. Client Certificate URLs

TLS specifies that when client authentication is performed, client certificates are sent by clients to servers during the TLS handshake. It may be desirable for wireless clients to send a certificate URL in place of a certificate so that they do not need to store their certificate and can therefore save memory.

In order to negotiate to send a certificate URL to a server, clients MAY include an extension of type "client_certificate_url" in the (extended) client hello. The "unknown_extension" field of this extension shall be empty.

(Note that it is necessary to negotiate use of a client certificate URL in order to avoid "breaking" existing TLS 1.0 servers.)

Servers that receive an extended client hello containing a "client_certificate_url" extension, MAY indicate that they are willing to accept a certificate URL by including an extension of type "client_certificate_url" in the (extended) server hello. The "unknown_extension" field of this extension shall be empty.

After negotiation of the use of a client certificate URL has been successfully completed (by exchanging hellos including "client_certificate_url" extensions), clients send a "CertificateorURL" message in place of a "Certificate" message:

```
         struct{
              Certificate_or_URL certificate_transport_type;
              select (Certificate_or_URL) {
                  case certificate:
                      ASN.1Cert certificate_list<0..2^24-1>;
                  case url:
                      opaque url<0..2^8-1>;
              } certificate_transport_body;
         } CertificateorURL;

         enum{ certificate(1), url(2), (255) } Certificate_or_URL ;
```

(Nevertheless, the handshake message is identified as being of type "certificate".)

Here:

- "certificate_transport_type" indicates whether a certificate chain or URL is being sent.

- "certificate_transport_body" contains either a certificate chain, or a certificate URL.

Servers receiving "CertificateorURL" shall attempt to retrieve the client's certificate chain from the URL (if necessary), and then process the certificate chain as usual.

Note that "CertificateorURL" allows the client to send either a certificate or a URL. The option to send a certificate, even after successfully negotiating the possibility to send a URL, is included to provide flexibility to clients possessing multiple certificates.

3.3. Trusted CA Indication

Wireless clients which, due to memory limitations, possess only a small number of CA root keys, may wish to indicate to servers which root keys they possess, in order to avoid repeated handshake failures.

In order to indicate which CA root keys they possess, clients MAY include an extension of type "trusted_key_ids" in the (extended) client hello. The "unknown_extension" field of this extension shall contain "TrustedAuthorities" where:

```
          TrustedAuthority TrustedAuthorities<0..2^16-1>;
```

```
          struct {
              IdentifierType identifier_type;
              select (identifier_type) {
                  case null: struct {};
                  case key_hash_sha: KeyHash;
                  case x509_name: DistinguishedName;
              } identifier;
          } TrustedAuthority;

          enum { null(0), key_hash_sha(1), x509_name(2),(255)}
              IdentifierType;

          opaque DistinguishedName<1..2^16-1>;

          opaque KeyHash[20];
```

Here "TrustedAuthorities" provides a list of CA root key identifiers that the client possesses. Each CA root key is identified via either:

- "null" - no CA root key identity supplied.

- "key_hash_sha" - contains the SHA-1 hash of the CA root key. (For DSA and ECDSA keys, this is the hash of the "subjectPublicKey" value. For RSA keys, this is the hash of the byte string representation of the modulus.)

- "x509_name" - contains the X.509 distinguished name of the CA.

Note that clients may include none, some, or all of the CA root keys they possess in this extension.

(The option to include no CA root keys is included both to maintain syntactical similarity with Section 3.6, and to allow the client to indicate possession of some pre-defined set of CA root keys.)

Servers that receive a client hello containing the "trusted_key_ids" extension, MAY use the information contained in the extension to guide their selection of an appropriate certificate chain to return to the client.

3.4. Small Session Identifiers

TLS specifies that session identifiers can be up to 32 bytes in length. In order to save memory, it is desirable to restrict the size of session identifiers which are stored by wireless clients.

The syntax used by TLS for session identifiers is:

```
          opaque SessionID<0..32>;
```

In order to encourage use of small session identifiers, servers SHOULD select session identifiers whose length is 8 bytes or less.

3.5. Truncated MACs

TLS uses the MAC construction HMAC with either MD5 or SHA-1 [HMAC] to authenticate record layer communications. In TLS the entire output of the hash function is used as the MAC tag. However it may be desirable in wireless environments to save bandwidth by truncating the output of the hash function when forming MAC tags.

In order to negotiate the use of truncated MACs, clients MAY include an extension of type "truncated_MAC" in the extended client hello. The "unknown_extension" field of this extension shall contain "MACTruncations", where:

```
MACTruncation MACTruncations<0..2^8-1>;

struct {
    MACAlgorithm mac_type;
    uint16 trucation_size_in_bits
} MACTruncation;

enum { null(0), md5(1), sha1(2), (255) } MACAlgorithm;
```

Here "MACTruncations" contains a list of MAC truncation sizes suggested by the client. Allowed values for suggested truncation sizes are: for HMAC-with-SHA1 ("sha1") - 80 bits; and for HMAC-with-MD5 ("md5") - 80 bits.

Servers that receive an extended hello containing a "truncated_MAC" extension, MAY agree to use a truncated MAC by including an extension of type "truncated_MAC" in the extended server hello. The "unknown_extension" field of this extension shall contain "MACTruncation". Here "MACTruncation" shall contain the agreed MAC truncation size, select from the list suggested by the client. Note that the "MACAlgorithm" identified in "MACTruncation" must match the MAC used by the established cipher suite.

Servers receiving MAC truncation negotiation requests requesting values other than the allowed values, MUST abort the handshake. Similarly clients receiving MAC truncation negotiation responses that differ from the values they suggested, or that do not match the established cipher suite, MUST abort the handshake.

Once MAC truncation has been successfully negotiated during a TLS handshake, both the client and the server pass the negotiated truncation size to the TLS record layer along with the other negotiated

security parameters. Subsequently during the session, clients and

servers MUST use truncated MACs. (Truncated MACs are calculated as specified in [HMAC].)

3.6. OCSP

Wireless clients may wish to use OCSP [OCSP] to check the validity of server certificates, in order to avoid transmission of CRLs and therefore save bandwidth on wireless networks.

In order to indicate their desire to use OCSP, clients MAY include an extension of type "status_request" in the (extended) client hello. The "unknown_extension" field of this extension shall contain "TrustedAuthorities" as defined in Section 3.3.

Here "TrustedAuthorities" provides a list of OCSP responders that the client trusts. The "null" alternative in "TrustedAuthorities" can be used to indicate that the responders are implicitly known to the server - e.g. by prior arrangement.

Servers that receive a client hello containing the "status_request" extension, MAY return an OCSP response to the client along with their certificate, and MAY use the information contained in the extension when selecting an OCSP responder.

Servers return an OCSP response along with their certificate by sending "CertificateAndOCSPResponse" in place of the "Certificate" message. If a server returns an OCSP response, then the server MUST include an extension of type "status_request" with empty "unknown_extensions" in the extended server hello.

```
struct {
    ASN.1Cert    certificate_list<0..2^24-1>;
    OCSPResponse ocsp_response;
} CertificateAndOCSPResponse;

opaque ASN.1Cert<1..2^24-1>;

opaque OCSPResponse<1..2^24-1>;
```

(Nevertheless, the handshake message is identified as being of type "certificate".)

Here "ocsp_response" contains a complete, DER-encoded OCSP response. Note that only one OCSP response may be sent.

Note that a server MAY also choose not to send the "CertificateAndOCSPResponse" message, and instead send the "Certificate" message, even if it receives a "status_request" extension in the client hello message.

Note in addition that servers MUST NOT send the
"CertificateAndOCSPResponse" message unless it received a
"status_request" extension in the client hello message.

Clients requesting an OCSP response, and receiving an OCSP response in
a "CertificateAndOCSPResponse" field:

- MUST process the certificate as if it was received in a "Certificate"
  message, and;

- MAY check the OCSP response and abort the handshake if the response
  is not satisfactory.

4. Security Considerations

The use of the extensions specified in this document may introduce security concerns for TLS clients and servers.

In particular, it is possible that truncated MACs are weaker than "un-truncated" MACs. No such weaknesses are currently known for the truncation sizes specified in this document.

In addition, it is possible that which CA root keys a client possesses could be regarded as confidential information. As a result, the CA root key indication extension should be used with care.

Also, TLS entities must be aware of the fact that until the handshake has been authenticated, active attackers can modify messages and insert, remove, or replace extensions.

In general, implementers should continue to monitor the state of the art, and address any weaknesses identified.

Additional security considerations are described in the TLS RFC [TLS].

5. Intellectual Property Rights

The IETF invites any interested party to bring to its attention any copyrights, patents or patent applications, or other proprietary rights which may cover technology that may be required to practice this document. Please address the information to the IETF Executive Director.

6. Acknowledgments

The authors wish to thank the WAP Security Group. This document is based on discussion within the WAP Security Group.

7. References

[HMAC] Krawczyk, H., Bellare, M., and Canetti, R. - HMAC: Keyed-hashing for message authentication. IETF RFC 2104, February 1997.

[MAILING LIST] Mikkelsen, J. Eberhard, R., and J. Kistler, "General ClientHello extension mechanism and virtual hosting," Ietf-tls mailing list posting, August 14, 2000.

[OCSP] Myers, M., Ankney, R., Malpani, A., Galperin, S., and C. Adams, "Internet X.509 Public Key Infrastructure: Online Certificate Status Protocol - OCSP," IETF RFC 2560, June 1999.

[TLS] Dierks, T., and C. Allen, "The TLS Protocol - Version 1.0,"
IETF RFC 2246, January 1999.

8. Authors' Addresses

Simon Blake-Wilson
Certicom Corp.
sblake-wilson@certicom.com

Magnus Nystrom
RSA Security
magnus@rsasecurity.com