Network Working Group
Internet-Draft
Expires: April 26, 2012

A. Langley
Google Inc

October 24, 2011

Transport Layer Security (TLS) Encrypted Client Certificates
draft-agl-tls-encryptedclientcerts-00

Abstract

This document describes a Transport Layer Security (TLS) extension that allows client certificates to be encrypted in the initial TLS handshake.

Status of this Memo

This Internet-Draft is submitted to IETF in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

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."

The list of current Internet-Drafts can be accessed at http://www.ietf.org/ietf/1id-abstracts.txt.

The list of Internet-Draft Shadow Directories can be accessed at http://www.ietf.org/shadow.html.

This Internet-Draft will expire on April 26, 2012.

Copyright Notice

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 BSD License.

Table of Contents

1. Introduction

TLS [RFC5246] defines a handshake in which both the server's and client's certificates (if any) are sent in the clear during the initial handshake. Although the server's certificates are usually non-sensitive, client certificates may include email address or even full legal names. Even client certificates that contain nothing but a serial number provide a unique identifier that can be correlated across connections by an eavesdropper.

This motivates encrypting the client's certificates. One existing solution is to perform an initial handshake without client authentication and then to renegotiate with it. This solves the disclosure issue but at a significant cost in handshake overhead and latency. The solution presented below simply moves the client's certificates after the client's ChangeCipherSpec. This is fundamentally incompatable with DH or ECDH certificates but we note that such certificates are rarely used in our experience. This solution is also weak as it only defends against eavesdroppers, not active attackers. We still consider it worthwhile given the very low cost.

2. Requirements Notation

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].

3. Encrypted client certificates extension

A new extension type ("encrypted_client_certificates(provisionally 13180)") is defined and MAY be included by the client in its "ClientHello" message. If, and only if, the server sees this extension in the "ClientHello", it MAY choose to include the extension in its "ServerHello". The "extension_data" MUST be empty in each case.

```
enum {
  encrypted_client_certificates(provisionally 13180), (65535)
} ExtensionType;
```

If the extension is echoed by the server then encrypted client certificates are in effect for the handshake. This causes the client's second flow to be reordered so that the "Certificate" and "CertificateVerify" messages occur after the "ChangeCipherSpec".

Here is an example of the client's second flow without encrypted client certificates (taken from RFC 5246 [RFC5246]):

```
Certificate*
ClientKeyExchange
CertificateVerify*
[ChangeCipherSpec]
Finished
```

When client encrypted certificates are in effect, this becomes:

```
ClientKeyExchange
[ChangeCipherSpec]
Certificate*
CertificateVerify*
Finished
```

The "handshake_messages" value of the "CertificateVerify" is constructed using the new message order.

This extension does not imply that a "CertificateRequest" handshake message will be sent by the server, nor that a "Certificate" or "CertificateVerify" message will be sent by the client. It only affects the message ordering when a client certificate would have normally been sent in the clear.

4. Security considerations

In the course of a normal handshake, the use of this extension will protect the client certificate from eavesdroppers. An active attacker can perform a downgrade attack and expose the client's certificates at the cost of dooming the connection. In order to defend against the active attack, a strict client may refuse to send certificates is the server doesn't support this extension in the initial handshake.

5. IANA Considerations

This document requires IANA to update its registry of TLS extensions to assign an entry, referred herein as "encrypted_client_certificates".

6. Acknowledgements

Thanks to Wan-Teh Chang, Diana Smetters, Brian Smith, Adam Barth, Dirk Balfanz and Mayank Upadhyay for discussions around this design.

7. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, August 2008.

Appendix A. Changes

To be removed by RFC Editor before publication

Author's Address

Adam Langley
Google Inc

Email: agl@google.com

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: November 15, 2011

A. Langley
Google Inc
P. Hoffman
VPNC

May 14, 2011

Transport Layer Security (TLS) Next Protocol Extension
draft-agl-tls-nextproto-00

Abstract

This document describes a Transport Layer Security (TLS) extension for application layer protocol probing and announcement. This allows the application client to specify which protocol will be performed over the secure connection.

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 15, 2011.

Copyright Notice

Table of Contents

1. Introduction

As the Internet has evolved, it has become commonplace for hosts to initiate connections based on untrusted and possibly hostile data. HTTP [RFC2616] clients are currently the most widespread example of this as they will fetch URLs based on the contents of untrusted webpages.

Any time that a connection is initiated based on untrusted data there is the possibility of a cross-protocol attack. If the attacker can control the contents of the connection in any way (for example, the requested URL in an HTTP connection) they may be able to encode a valid message in another protocol. The connecting host believes that it is speaking one protocol but the server understands it to be another. The application of Postel's Law exacerbates the issue as many servers will permit gross violations of the expected protocol in order to achieve maximum compatibility with clients.

The WebSockets [websockets] protocol seeks to allow low-latency, full-duplex communication between browsers and HTTP servers. However, it also permits an unprecedented amount of attacker control over the contents of the connection. In order to prevent cross-protocol attacks, a mechanism to assure that both client and server are speaking the same protocol is required. To this end, the Next Protocol extension described in this document extends the TLS [RFC5246] handshake to allow the client to tell the server the intended application protocol.

1.1. Design of the Protocol

The basic design of the extension is that the client expresses that it knows how to specify which application protocol it will use after the TLS session is established. The server responds with some or all the types of application protocols that it knows. The client responds with the protocol that it intends to use; this might even be one that was not listed by the server.

Note that this protocol is not a negotiation in the classic sense of "client says what it wants and server picks one choice". Instead, it allows the client to not reveal in the clear which application protocol it intends to use after TLS is established. Intermediaries who might prevent TLS from being established for a particular application cannot determine which protocol will be used; this, in turn, leads to secure connections for more protocols.

2. Requirements Notation

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].

3. Next Protocol Extension

This document defines a new extension type, "next_protocol(TBD)".

```
enum {
  next_protocol(TBD), (65535)
} ExtensionType;
```

The next_protocol extension MAY be included by the client in its "ClientHello" message. If, and only if, the server sees this extension in the "ClientHello", it MAY choose to include the extension in its "ServerHello".

The "extension_data" field of a "next_protocol" in a "ClientHello" MUST be empty. The "extension_data" field of a "next_protocol" in a "ServerHello" contains the list of application-layer protocols that the server wishes to advertise that it supports. That list is a set of two-octet (uint16) values, in network byte order, taken from the ports numbers assigned by IANA; see <http://www.iana.org/assignments/port-numbers>.

This document also defines a new handshake message type, "next_protocol_ann(TBD)").

```
struct {
  uint16 announced_protocol;
} NextProtocolAnnounce;
```

If, and only if, the server included a "next_protocol" extension in its ServerHello message, the client MUST send a "NextProtocolAnnounce" message after its "ChangeCipherSpec" and before its "Finished" message. The NextProtocolAnnounce message contains the single application-layer protocol that the client will use in this connection after the TLS handshake completes; that value does not need to match any of those given by the server in the next_protocol extension.

3.1. Handshake Summary

```
-> ClientHello (contains next_protocol extension
       with empty extension_data )
<- ServerHello (contains next_protocol extension
       with list of protocols)
<- ...
<- ServerHelloDone
-> ClientKeyExchange
-> ...
-> ChangeCipherSpec
-> NextProtocolAnnounce (contains announced_protocol)
-> Finished
<- ChangeCipherSpec
<- Finished
```

3.2. Session Resumption and Renegotiation

Unlike many other TLS extensions, this extension does not establish properties of the session, only of the connection. When session resumption or session tickets [RFC5077] are used, the previous contents of this extension are irrelevant and only the values in the new handshake messages are considered.

For the same reasons, after a handshake has been performed for a given connection, renegotiations on the same connection MUST NOT

include the "next_protocol" extension.

4. Security Considerations

This extension sends the server's list of supported protocols in the clear. This may be undesirable for certain protocols (such as Tor [tor]) where one could imagine that hostile networks would terminate any TLS connection with a server that advertised such a capability. Thus, if a client knows through out-of-band methods that a server supports a particular protocol, it can specify that protocol in the NextProtocolAnnounce message and use that protocol after TLS is set up.

5. IANA Considerations

This document requires IANA to update its registry of TLS extensions to assign an entry, referred herein as "next_protocol".

This document also requires IANA to update its registry of TLS handshake types to assign an entry, referred herein as "next_protocol_ann".

6. Acknowledgements

This document benefitted specifically from discussions with Wan-Teh Chang and Nagendra Modadugu.

7. References

7.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[RFC2616] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., and T. Berners-Lee, "Hypertext Transfer Protocol -- HTTP/1.1", RFC 2616, June 1999.

[RFC5077] Salowey, J., Zhou, H., Eronen, P., and H. Tschofenig, "Transport Layer Security (TLS) Session Resumption without Server-Side State", RFC 5077, January 2008.

[RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, August 2008.

7.2. Informative References

[websockets] Fette, I., "The Web Socket protocol", draft-ietf-hybi-thewebsocketprotocol (work in progress), 2011.

[tor] "Tor Onion Router", www.tor.org , 2011.

Authors' Addresses

Adam Langley

Google Inc

Email: agl@google.com

Paul Hoffman
VPNC

Email: paul.hoffman@vpnc.org

Network Working Group
Internet-Draft
Expires: October 31, 2012

A. Langley
Google Inc
May 2012

Transport Layer Security (TLS) Next Protocol Negotiation Extension
draft-agl-tls-nextprotoneg-04

Abstract

This document describes a Transport Layer Security (TLS) extension for application layer protocol negotiation. This allows the application layer to negotiate which protocol should be performed over the secure connection.

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 October 31, 2012.

Table of Contents

1. Introduction

The Next Protocol Negotiation extension (NPN) is currently used to negotiate the use of SPDY [spdy] as an application level protocol on port 443, and to perform SPDY version negotiation. However, it is not SPDY specific in any way.

Designers of new application level protocols are faced with a problem: there are no good options for establishing a clean transport for a new protocol and negotiating its use. Negotiations on port 80 will run afoul of intercepting proxies. Ports other than 80 and 443 are likely to be firewalled without any fast method of detection, and are also unlikely to traverse HTTP proxies with CONNECT. Negotiating on port 443 is possible, but may run afoul of MITM proxies and also uses a round trip for negotiation on top of the round trips for establishing the TLS connection. Negotiation at that level is also dependent on the application level protocol, i.e. the real world tolerance of servers to HTTP Upgrade requests.

Next Protocol Negotiation allows application level protocols to be negotiated without additional round trips and with clean fallback in the case of an unsupportive MITM proxy.

2. Requirements Notation

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].

3. Next Protocol Negotiation Extension

A new extension type ("next_protocol_negotiation(TBD)") is defined and MAY be included by the client in its "ClientHello" message. If, and only if, the server sees this extension in the "ClientHello", it MAY choose to echo the extension in its "ServerHello".

```
enum {
  next_protocol_negotiation(TBD), (65535)
} ExtensionType;
```

The "extension_data" field of a "next_protocol_negotiation" extension in a "ClientHello" MUST be empty.

The "extension_data" field of a "next_protocol_negotiation" extension in a "ServerHello" contains an optional list of protocols advertised by the server. Protocols are named by opaque, non-empty byte strings and the list of protocols is serialized as a concatenation of 8-bit, length prefixed byte strings. Implementations MUST ensure that the empty string is not included and that no byte strings are truncated.

A new handshake message type ("encrypted_extensions(TBD)") is defined. If the server included a "next_protocol_negotiation" extension in its "ServerHello" message, the client MUST send a "EncryptedExtensions" message after its "ChangeCipherSpec" and before its "Finished" message.

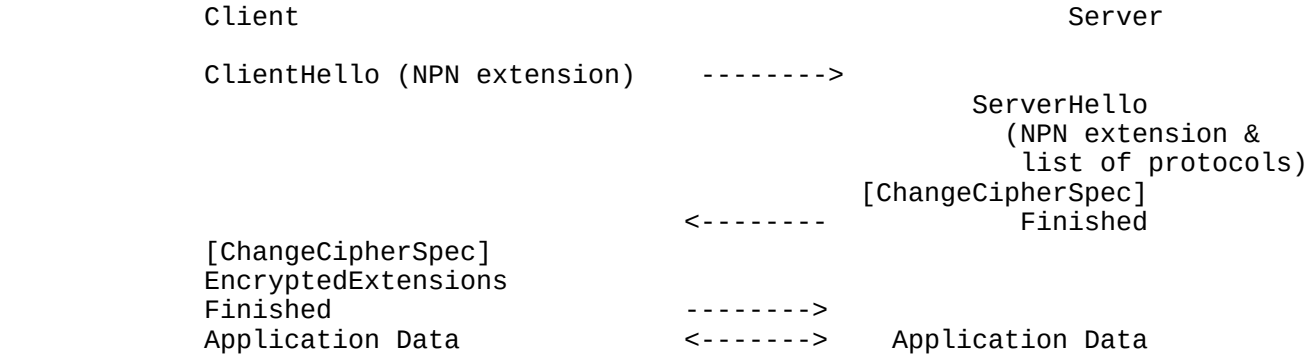
```
enum {
  encrypted_extensions(TBD), (65535)
} HandshakeType;
```

Therefore a full handshake with "EncryptedExtensions" has the following flow (contrast with section 7.3 of RFC 5246 [RFC5246]):

```
Client                                               Server

ClientHello (NPN extension)   -------->
                                                ServerHello
                                                  (NPN extension &
                                                   list of protocols)
                                               Certificate*
                                          ServerKeyExchange*
                                         CertificateRequest*
                              <--------      ServerHelloDone
Certificate*
ClientKeyExchange
CertificateVerify*
[ChangeCipherSpec]
EncryptedExtensions
Finished                      -------->
                                          [ChangeCipherSpec]
                              <--------             Finished
Application Data              <------->     Application Data
```

An abbreviated handshake with "EncryptedExtensions" has the following flow:



The "EncryptedExtensions" message contains a series of "Extension" structures (see section 7.4.1.4 of RFC 5246 [RFC5246]

If the server included a "next_protocol_negotiation" extension in its "ServerHello" message, the client MUST include an "Extension" with "extension_type" equal to "next_protocol_negotiation(TBD)". The "extension_data" of which has the following format:

```
struct {
  opaque selected_protocol<0..255>;
  opaque padding<0..255>;
} NextProtocolNegotiationEncryptedExtension;
```

The contents of "selected_protocol" are an opaque protocol string, but need not have been advertised by the server. The length of "padding" SHOULD be 32 - ((len(selected_protocol) + 2) % 32). Note that len(selected_protocol) does not include its length prefix.

Unlike many other TLS extensions, this extension does not establish properties of the session, only of the connection. When session resumption or session tickets [RFC5077] are used, the previous contents of this extension are irrelevant and only the values in the new handshake messages are considered.

For the same reasons, after a handshake has been performed for a given connection, renegotiations on the same connection MUST NOT include the "next_protocol_negotiation" extension.

4. Protocol selection

It’s expected that a client will have a list of protocols that it supports, in preference order, and will only select a protocol if the server supports it. In that case, the client SHOULD select the first protocol advertised by the server that it also supports. In the event that the client doesn’t support any of server’s protocols, or the server doesn’t advertise any, it SHOULD select the first protocol that it supports.

There may be cases where the client knows, via other means, that a server supports an unadvertised protocol. In these cases the client can simply select that protocol.

5. Design discussion

NPN is an outlier from TLS in several respects: firstly that it introduces a handshake message between the "ChangeCipherSpec" and "Finished" message, that the handshake message is padded, and that the negotiation isn't done purely with the hello messages. All these aspects of the protocol are intended to prevent middle-ware discrimination based on the negotiated protocol and follow the general principle that anything that can be encrypted, should be encrypted. The server's list of advertised protocols is in the clear as a compromise between performance and robustness.

6. Security considerations

The server's list of supported protocols is still advertised in the clear with this extension. This may be undesirable for certain protocols (such as Tor [tor]) where one could imagine that hostile networks would terminate any TLS connection with a server that advertised such a capability. In this case, clients may wish to opportunistically select a protocol that wasn't advertised by the server. However, the workings of such a scheme are outside the scope of this document.

7. IANA Considerations

This document requires IANA to update its registry of TLS extensions to assign an entry referred to here as "next_protocol_negotiation".

This document also requires IANA to update its registry of TLS handshake types to assign an entry referred to here as "encrypted_extensions".

This document also requires IANA to create a registry of TLS Next Protocol Negotiation protocol strings on a first come, first served basis, initially containing the following entries:

- "http/1.1": HTTP/1.1 [RFC2616]
- "spdy/1": (obsolete) SPDY version 1
- "spdy/2": SPDY version 2
- "spdy/3": SPDY version 3

8. Acknowledgments

This document benefited specifically from discussions with Wan-Teh Chang and Nagendra Modadugu.

9. References

9.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, August 2008.

9.2. Informative References

[RFC2616] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P. and T. Berners-Lee, "Hypertext Transfer Protocol -- HTTP/1.1", RFC 2616, June 1999.

[RFC5077] Salowey, J., Zhou, H., Eronen, P. and H. Tschofenig, "Transport Layer Security (TLS) Session Resumption without Server-Side State", RFC 5077, January 2008.

[tor] Dingledine, R., Matthewson, N. and P. Syverson, "Tor: The Second-Generation Onion Router", August 2004.

[spdy] Belshe, M. and R. Peon, "SPDY Protocol (Internet Draft)", Feb 2012.

Author’s Address

Adam Langley
Google Inc

Email: agl@google.com

Internet Engineering Task Force
Internet-Draft
Expires: May 16, 2012

D. Balfanz, Ed.
D. Smetters
M. Upadhyay
A. Barth
Google Inc.

November 13, 2011

TLS Origin-Bound Certificates
draft-balfanz-tls-obc-01

Abstract

This document specifies a Transport Layer Security (TLS) extension and associated semantics that allow clients and servers to negotiate the use of origin-bound, self-signed certificates for TLS client authentication.

Status of this Memo

This Internet-Draft is submitted to IETF in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

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."

The list of current Internet-Drafts can be accessed at http://www.ietf.org/ietf/1id-abstracts.txt.

The list of Internet-Draft Shadow Directories can be accessed at http://www.ietf.org/shadow.html.

This Internet-Draft will expire on May 16, 2012.

Copyright Notice

Table of Contents

1. Introduction

Transport Layer Security (TLS [RFC5246]) allows clients to authenticate themselves using Client Certificates. In practice, clients tend to ask for user consent before using Client Certificates. This is to allay privacy concerns about user-identifying information in the Client Certificate, and also to let the user choose among possibly many certificates that can be used by the client for the TLS session.

The user experience of obtaining this consent, along with that of obtaining the certificates in the first place, has traditionally presented a hurdle to user adoption. Additionally, operational constraints on the server side can make it difficult for service providers to switch from a cookie-based authentication scheme to certificate-based TLS client authentication.

The TLS Origin-Bound Certificates extension (TLS-OBC) is a TLS extension [RFC6066] that allows clients to use certificate-based client authentication without having to obtain user consent before using certificates. A client creates at most one (self-signed) certificate of any given type (e.g., rsa_sign or ecdsa_sign) per web origin [WebOrigin], and does not include user-identifying information into such origin-bound certificates, thus making user consent and user-assisted certificate selection unnecessary.

1.1. Requirements Language

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].

1.2. Extension Dependencies

The client SHOULD use TLS-OBC during a TLS session only if it also uses the TLS Encrypted Client Certificates [EncryptedClientCerts] extension during that TLS session.

2. Origin-Bound Certificates

An origin-bound certificate is a self-issued certificate that the client uses for TLS client authentication for a particular web origin [WebOrigin]. Origin-bound certificates MUST be self-signed, i.e., a private key corresponding to the certified public key MUST be used to sign the certificate.

Clients MUST NOT re-use the same origin-bound certificate for more

than one web origin.

2.1. Certificate Creation

Clients create origin-bound certificates when asked to perform TLS client authentication after negotiating the "ob_cert" extension with a server from that origin (see Section 3.5), and they do not already have a valid origin-bound certificate for that origin available for use.

Clients SHOULD NOT re-use the same key pair for more than one origin-bound certificate. To do so would violate the privacy properties required by origin-bound certificates.

When creating an origin-bound certificate, it is RECOMMENDED that clients use the PrintableString representation of "anonymous.invalid" as the common name component of the Distinguished Name of both the certificate's Issuer and Subject, with no other name components present.

It is RECOMMENDED that clients pick a lifetime for the new certificate between one month and one year (perhaps depending on an assessment of how well the private key is protected on the host platform). The client SHALL use the notBefore and notAfter fields in the new certificate to record the lifetime of the new certificate.

2.1.1. Origin-Bound Certificate Extension

Origin-bound certificates MUST be X.509 v3 certificates, and MUST include the origin-bound certificate extension (OID 1.3.6.1.4.1.11129.2.1.6), shown below. This extension MUST be marked critical.

The data of the extension, "OriginBoundCertificate", will be the ASCII Serialization of the certificate's web origin [WebOrigin] as an IA5String.

```
 id-ce-originBoundCertificate OBJECT IDENTIFIER ::=
         { enterprises Google 2 1 6 }

 OriginBoundCertificate ::=
         IA5String
```

Figure 1

The server MAY reject the origin-bound certificate if it determines that the web origin included in the "OriginBoundCertificate" extension does not match the server's origin (for example, if it

doesn't match the host name specified in the Server Name Indication TLS extension). It is, however, RECOMMENDED that the server merely report the value of the web origin to the upper layers in the protocol stack, and leave it up to those layers to make use of the web origin value (e.g., to assist developers in debugging the feature).

2.2. Certificate and Key Rotation

After a client has created an origin-bound certificate for a certain web origin, the client SHOULD re-use the certificate for a period of time, and then discard it.

The client MUST discard existing origin-bound certificate at or before the notAfter time indicated in each certificate.

The client MAY opt to discard an existing origin-bound certificate at any time of its choosing prior to the notAfter time indicated in the certificate. In particular, the client may delete an existing origin-bound certificate in response to a user request to clear privacy-sensitive state from their user-agent, ensuring that a fresh certificate will be generated the next time a user visits that origin.

The client SHOULD use a new key pair when it generates a new origin-bound certificate.

3. Changes to The Handshake Message Contents

3.1. Extension Definition

This document defines a new TLS extension, "ob_cert" (with tentative extension type 13175, or 0x3377), which indicates that client and server agree to allow the client to use an origin-bound certificate to authenticate itself to the server.

To indicate support for origin-bound certificates, the client includes an extension of type "ob_cert" in the Extended Client Hello message. To indicate that a subsequent CertificateRequest message requests an origin-bound certificate, the server includes the same extension in its Extended Server Hello message.

The "ob_cert" TLS extension will be assigned a value from the TLS ExtensionType registry, and will contain zero length "extension_data". For testing, we will use the extension type value of 0x3377. For this type value, the entire encoding of the extension will be 33 77 00 00.

3.2. Client Hello

A client wishing to indicate support for origin-bound client certificates MUST include the "ob_cert" in the extended Client Hello message to a server from a particular web origin.

3.3. Server Hello

A server that receives a Client Hello message containing the "ob_cert" extension MAY choose to include the same extension in the extended Server Hello message. If it does, the client and server are considered to have negotiated origin-bound client certificates for the session.

3.4. Certificate Request

A server MAY choose to send a Certificate Request message to the client. If the session uses origin-bound client certificates, the server SHOULD use an empty "certificate_authorities" list.

A client that receives a Certificate Request during a session for which origin-bound client certificates were negotiated MUST ignore the "certificate_authorities" list.

3.5. Client Certificate

TLS [RFC5246] requires that a client that receives a Certificate Request message must send a Client Certificate message in response (which may or may not include client certificates). If client and server negotiated origin-bound client certificates, then the client SHOULD include an origin-bound certificate in the Client Certificate message.

If a client includes an origin-bound certificate in the Client Certificate message during a session for which origin-bound client certificates were negotiated, the included certificate MUST

- be an origin-bound certificate for the web origin of the server, and
- match a certificate type requested by the server in the Certificate Request message.

If a client doesn’t possess a certificate meeting the above requirements, it SHOULD self-issue a certificate of a type requested by the server in the Certificate Request message and include the newly-generated certificate in the Client Certificate message. Only if the client is not capable of providing a certificate of the

requested type SHOULD it include an empty "certificate_list" in the Client Certificate message.

A server verifying a Client Certificate message in a handshake that negotiated origin-bound client certificates MUST verify that the certificate is self-signed, rather than being signed by any particular CA. The server MUST verify that the public key in the certificate corresponds to the key used to authenticate the client in the handshake. The server SHOULD ignore the Issuer and Subject in the presented certificate. The server MUST ignore the notBefore and notAfter fields in the certificate.

Finally, the server MUST verify that the certificate contains the origin-bound extension, and MAY verify that the origin identified in that certificate matches the server (see Section 2.1.1).

4. Security Considerations

4.1. Third-Party Tracking

A third party might be able to track a client across multiple sessions by observing the use of the same origin-bound certificate. To alleviate this concern, clients SHOULD also implement the Encrypted Client Certificates Extension [EncryptedClientCerts], and use it concurrently with TLS-OBC. The Encrypted Client Certificates Extension [EncryptedClientCerts] ensures that the origin-bound certificate is sent after TLS established secrecy on the channel between client and server.

4.2. Server Tracking

A client can prevent server tacking by deleting the origin-bound certificate for the server’s Web origin. This could happen, for example, when the user elects to remove all cookies for that origin.

4.3. Cookie Hardening

One way TLS-OBC can be used to strengthen cookie-based authentication is by "binding" cookies to an origin-bound certificate. The server, when issuing a cookie for an HTTP session, would associate the client’s origin-bound certificate with the session (either by encoding information about the certificate unforgeably in the cookie, or by associating the certificate with the cookie’s session through some other means). That way, if and when a cookie gets stolen from a client, it cannot be used over a TLS connection initiated by a different client - the cookie thief would also have to steal the private key associated with the client’s origin-bound certificate, a

task considerably harder especially when we assume the existence of a Trusted Platform Module or other Secure Element that can store the origin-bound-certificate's private key.

4.4. Key Lengths

If the client uses RSA key pairs, the key length should be at least 1024 bits. For longer-lived origin-bound certificates, the key length should be 2048 bits.

If the client uses ECDSA key pairs, the key length should be at least 160 bits.

5. References

[EncryptedClientCerts]
Langley, A., "TLS Encrypted Client Certificates", <http://tools.ietf.org/html/draft-agl-tls-encryptedclientcerts>.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, August 2008.

[RFC6066] Eastlake, D., "Transport Layer Security (TLS) Extensions: Extension Definitions", RFC 6066, January 2011.

[WebOrigin]
Barth, A., "The Web Origin Concept", <http://tools.ietf.org/html/draft-abarth-origin-09>.

Authors' Addresses

Dirk Balfanz (editor)
Google Inc.
1600 Ampitheatre Parkway
Mountain View, CA
USA

Phone:
Email: balfanz@google.com

D K Smetters
Google Inc.

Mayank Upadhyay
Google Inc.

Adam Barth
Google Inc.

Internet Engineering Task Force
Internet-Draft
Intended status: Standards Track
Expires: March 11, 2013

D. Harkins, Ed.
Aruba Networks
D. Halasz, Ed.
Halasz Ventures

September 7, 2012

Secure Password Ciphersuites for Transport Layer Security (TLS)
draft-harkins-tls-pwd-03

Abstract

This memo defines several new ciphersuites for the Transport Layer Security (TLS) protocol to support certificate-less, secure authentication using only a simple, low-entropy, password. The ciphersuites are all based on an authentication and key exchange protocol that is resistant to off-line dictionary attack.

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 March 11, 2013.

Copyright Notice

Table of Contents

1. Background

1.1. The Case for Certificate-less Authentication

TLS usually uses public key certificates for authentication [RFC5246]. This is problematic in some cases:

- Frequently, TLS [RFC5246] is used in devices owned, operated, and provisioned by people who lack competency to properly use certificates and merely want to establish a secure connection using a more natural credential like a simple password. The proliferation of deployments that use a self-signed server certificate in TLS [RFC5246] followed by a PAP-style exchange over the unauthenticated channel underscores this case.

- A password is a more natural credential than a certificate (from early childhood people learn the semantics of a shared secret), so a password-based TLS ciphersuite can be used to protect an HTTP-based certificate enrollment scheme-- e.g. an [RFC5967] -style request and an [RFC5751] -style response-- to parlay a simple password into a certificate for subsequent use with any certificate-based authentication protocol. This addresses a significant "chicken-and-egg" dilemma found with certificate-only use of [RFC5246].

- Some PIN-code readers will transfer the entered PIN to a smart card in clear text. Assuming a hostile environment, this is a bad practice. A password-based TLS ciphersuite can enable the establishment of an authenticated connection between reader and card based on the PIN.

1.2. Resistance to Dictionary Attack

It is a common misconception that a protocol that authenticates with a shared and secret credential is resistent to dictionary attack if the credential is assumed to be an N-bit uniformly random secret, where N is sufficiently large. The concept of resistence to dictionary attack really has nothing to do with whether that secret can be found in a standard collection of a language’s defined words (i.e. a dictionary). It has to do with how an adversary gains an advantage in attacking the protocol.

For a protocol to be resistant to dictionary attack any advantage an adversary can gain must be a function of the amount of interactions she makes with an honest protocol participant and not a function of the amount of computation she uses. The adversary will not be able to obtain any information about the password except whether a single guess from a single protocol run which she took part in is correct or

incorrect.

It is assumed that the attacker has access to a pool of data from which the secret was drawn-- it could be all numbers between 1 and 2^N, it could be all defined words in a dictionary. The key is that the attacker cannot do a an attack and then enumerate through the pool trying potential secrets (computation) to see if one is correct. She must do an active attack for each secret she wishes to try (interaction) and the only information she can glean from that attack is whether the secret used with that particular attack is correct or not.

2. Keyword Definitions

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].

3. Introduction

3.1. Notation

The following notation is used in this memo:

password
: a secret, and potentially low-entropy word, phrase, code or key used as a credential for authentication. The password is shared between the TLS client and TLS server.

y = H(x)
: a binary string of arbitrary length, x, is given to a function H which produces a fixed-length output, y.

a | b
: denotes concatenation of string a with string b.

[a]b
: indicates a string consisting of the single bit "a" repeated "b" times.

x mod y
: indicates the remainder of division of x by y. The result will be between 0 and y.

LSB(x)
    returns the least-significant bit of the bitstring "x".

3.2. Discrete Logarithm Cryptography

The ciphersuites defined in this memo use discrete logarithm cryptography (see [SP800-56A]) to produce an authenticated and shared secret value that is an element in a group defined by a set of domain parameters. The domain parameters can be based on either Finite Field Cryptography (FFC) or Elliptic Curve Cryptography (EEC).

Elements in a group, either an FFC or EEC group, are indicated using upper-case while scalar values are indicated using lower-case.

3.2.1. Elliptic Curve Cryptography

The authenticated key exchange defined in this memo uses fundamental algorithms of elliptic curves defined over GF(p) as described in [RFC6090].

Domain parameters for the ECC groups used by this memo are:

- A prime, p, determining a prime field GF(p). The cryptographic group will be a subgroup of the full elliptic curve group which consists points on an elliptic curve-- elements from GF(p) that satisfy the curve’s equation-- together with the "point at infinity" that serves as the identity element.

- Elements a and b from GF(p) that define the curve’s equation. The point (x,y) in GF(p) x GF(p) is on the elliptic curve if and only if (y^2 - x^3 - a*x - b) mod p equals zero (0).

- A point, G, on the elliptic curve, which serves as a generator for the ECC group. G is chosen such that its order, with respect to elliptic curve addition, is a sufficiently large prime.

- A prime, q, which is the order of G, and thus is also the size of the cryptographic subgroup that is generated by G.

- A co-factor, f, defined by the requirement that the size of the full elliptic curve group (including the "point at infinity") is the product of f and q.

This memo uses the following ECC Functions:

- Z = elem-op(X,Y) = X + Y: two points on the curve X and Y, are sumed to produce another point on the curve, Z. This is the group operation for ECC groups.

- Z = scalar-op(x,Y) = x * Y: an integer scalar, x, acts on a point on the curve, Y, via repetitive addition (Y is added to itself x times), to produce another EEC element, Z.

- Y = inverse(X): a point on the curve, X, has an inverse, Y, which is also a point on the curve, when their sum is the "point at infinity" (the identity for elliptic curve addition). In other words, R + inverse(R) = "0".

- z = F(X): the x-coordinate of a point (x, y) on the curve is returned. This is a mapping function to convert a group element into an integer.

Only ECC groups over GF(p) can be used with TLS-PWD. ECC groups over GF(2^m) SHALL NOT be used by TLS-PWD. In addition, ECC groups with a co-factor greater than one (1) SHALL NOT be used by TLS-PWD.

A composite (x, y) pair can be validated as an a point on the elliptic curve by checking whether: 1) both coordinates x and y are greater than zero (0) and less than the prime defining the underlying field; 2) the x- and y- coordinates satisfy the equation of the curve; and 3) they do not represent the point-at-infinity "0". If any of those conditions are not true the (x, y) pair is not a valid point on the curve.

3.2.2. Finite Field Cryptography

Domain parameters for the FFC groups used by this memo are:

- A prime, p, determining a prime field GF(p), the integers modulo p. The FFC group will be a subgroup of GF(p)*, the multiplicative group of non-zero elements in GF(p).

- An element, G, in GF(p)* which serves as a generator for the FFC group. G is chosen such that its multiplicative order is a sufficiently large prime divisor of ((p-1)/2).

- A prime, q, which is the multiplicative order of G, and thus also the size of the cryptographic subgroup of GF(p)* that is generated by G.

This memo uses the following FFC Functions:

- Z = elem-op(X,Y) = (X * Y) mod p: two FFC elements, X and Y, are multiplied modulo the prime, p, to produce another FFC element, Z. This is the group operation for FFC groups.

- Z = scalar-op(x,Y) = Y^x mod p: an integer scalar, x, acts on an FFC group element, Y, via exponentiation modulo the prime, p, to produce another FFC element, Z.

- Y = inverse(X): a group element, X, has an inverse, Y, when the product of the element and its inverse modulo the prime equals one (1). In other words, (X * inverse(X)) mod p = 1.

- z = F(X): is the identity function since an element in an FFC group is already an integer. It is included here for consistency in the specification.

Many FFC groups used in IETF protocols are based on safe primes and do not define an order (q). For these groups, the order (q) used in this memo shall be the prime of the group minus one divided by two--(p-1)/2.

An integer can be validated as being an element in an FFC group by checking whether: 1) it is between one (1) and the prime, p, exclusive; and 2) if modular exponentiation of the integer by the group order, q, equals one (1). If either of these conditions are not true the integer is not an element in the group.

3.3. Instantiating the Random Function

The protocol described in this memo uses a random function, H, which is modeled as a "random oracle". At first glance, one may view this as a hash function. As noted in [RANDOR], though, hash functions are too structured to be used directly as a random oracle. But they can be used to instantiate the random oracle.

The random function, H, in this memo is instantiated by using the hash algorithm defined by the particular TLS-PWD ciphersuite in HMAC mode with a key whose length is equal to block size of the hash algorithm and whose value is zero. For example, if the ciphersuite is TLS_ECCPWD_WITH_AES_128_GCM_SHA256 then H will be instantiated with SHA256 as:

```
H(x) = HMAC-SHA256([0]32, x)
```

3.4. Passwords

The authenticated key exchange used in TLS-PWD requires each side to have a common view of a shared credential. To protect a database of stored passwords, though, the password SHALL be salted and the result, called the base, SHALL be used as the authentication credential.

The salting function is defined as:

    base = HMAC-SHA256(salt, username | password)

The password used for generation of the base SHALL be represented as a UTF-8 encoded character string processed according to the rules of the [RFC4013] profile of [RFC3454] and the salt SHALL be a 32 octet random number. The server SHALL store a triplet of the form:

    { username, base, salt }

And the client SHALL generate the base upon receiving the salt from the server.

3.5. Assumptions

The security properties of the authenticated key exchange defined in this memo are based on a number of assumptions:

1. The random function, H, is a "random oracle" as defined in [RANDOR].

2. The discrete logarithm problem for the chosen group is hard. That is, given g, p, and y = g^x mod p, it is computationally infeasible to determine x. Similarly, for an ECC group given the curve definition, a generator G, and Y = x * G, it is computationally infeasible to determine x.

3. Quality random numbers with sufficient entropy can be created. This may entail the use of specialized hardware. If such hardware is unavailable a cryptographic mixing function (like a strong hash function) to distill enropy from multiple, uncorrelated sources of information and events may be needed. A very good discussion of this can be found in [RFC4086].

4. Specification of the TLS-PWD Handshake

The authenticated key exchange is accomplished by each side deriving a password-based element, PE, in the chosen group, making a "committment" to a single guess of the password using PE, and generating the Premaster Secret. The ability of each side to produce a valid finished message authenticates itself to the other side.

The authenticated key exchange is dropped into the standard TLS message handshake by modifying some of the messages.

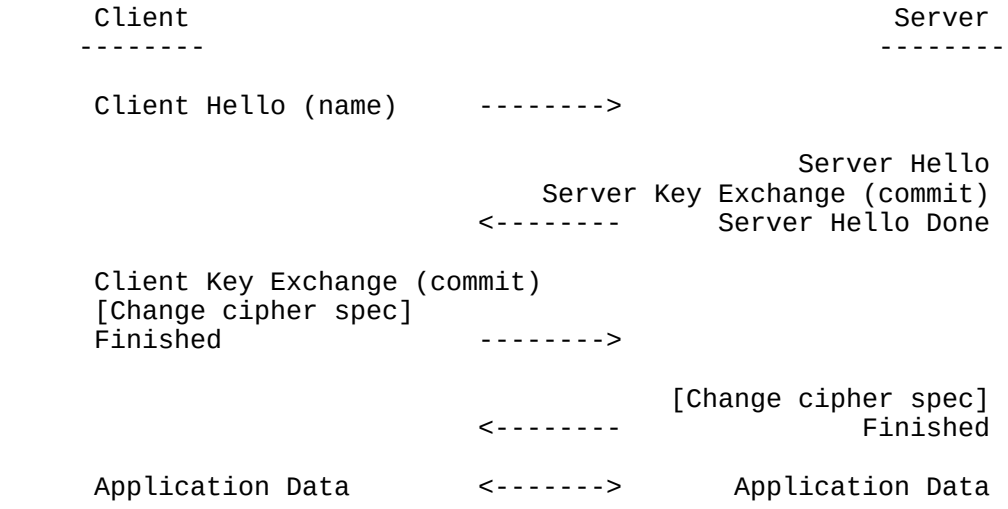


Figure 1

4.1. Fixing the Password Element

Prior to making a "committment" both sides must generate a secret element, PE, in the chosen group using the common password-derived base. The server generates PE after it receives the Client Hello and chooses the particular group to use, and the client generates PE upon receipt of the Server Key Exchange.

Fixing the password element involves an iterative "hunting and pecking" technique using the prime from the negotiated group's domain parameter set and an ECC- or FFC-specific operation depending on the negotiated group.

To thwart side channel attacks which attempt to determine the number of iterations of the "hunting-and-pecking" loop are used to find PE for a given password, a security parameter, k, is used to ensure that at least k iterations are always performed. This technique need only be used with ECC groups.

First, an 8-bit counter is set to the value one (1). Then, H is used to generate a password seed from the a counter, the prime of the selected group, and the base (which is derived from the username, password, and salt):

pwd-seed = H(base | counter | p)

Then, the pwd-seed is expanded using the PRF to the length of the prime from the negotiated group's domain parameter set, to create

pwd-value:

```
pwd-value = PRF(pwd-seed, "TLS-PWD Hunting And Pecking",
                ClientHello.random | ServerHello.random) [0..p];
```

If the pwd-value is greater than or equal to the prime, p, the counter is incremented, and a new pwd-seed is generated and the hunting-and-pecking continues. If pwd-value is less than the prime, p, it is passed to the group-specific operation which either returns the selected password element or fails. If the group-specific operation fails, the counter is incremented, a new pwd-seed is generated, and the hunting-and-pecking continues. This process continues until the group-specific operation returns the password element. For FCC groups, this terminates the hunting-and-pecking process. For ECC groups, after the password element has been chosen, the base is changed to a random number, the counter is incremented and the hunting-and-pecking continues until the counter is greater than the security parameter, k.

When PE has been discovered, pwd-seed and pwd-value SHALL be irretrievably destroyed.

4.1.1. Computing an ECC Password Element

The group-specific operation for ECC groups uses pwd-value, pwd-seed, and the equation for the curve to produce PE. First, pwd-value is used directly as the x-coordinate, x, with the equation for the elliptic curve, with parameters a and b from the domain parameter set of the curve, to solve for a y-coordinate, y. If there is no solution to the quadratic equation, this operation fails and the hunting-and-pecking process continues. If a solution is found, then an ambiguity exists as there are technically two solutions to the equation and pwd-seed is used to unambiguously select one of them. If the low-order bit of pwd-seed is equal to the low-order bit of y, then a candidate PE is defined as the point (x, y); if the low-order bit of pwd-seed differs from the low-order bit of y, then a candidate PE is defined as the point (x, p - y), where p is the prime over which the curve is defined. The candidate PE becomes PE, a random number is used instead of the base, and the hunting and pecking continues until it has looped through k iterations.

Algorithmically, the process looks like this:

```
found = 0
counter = 0
base = H(username | password | salt)
do {
  counter = counter + 1
  pwd-seed = H(base | counter | p)
  pwd-value = PRF(pwd-seed, "TLS-PWD Hunting And Pecking",
                  ClientHello.random | ServerHello.random) [0..p]
  if (pwd-value < p)
  then
    x = pwd-value
    if ( (y = sqrt(x^3 + ax + b)) != FAIL)
      then
      if (found == 0)
      then
        if (LSB(y) == LSB(pwd-seed))
        then
          PE = (x, y)
        else
          PE = (x, p-y)
        fi
        found = 1
      else
        base = random()
      fi
    fi
  fi
} while ((found == 0) || (counter <= k))
```

Figure 2: Fixing PE for ECC Groups

The probability that one requires more than "n" iterations of the "hunting and pecking" loop to find PE is roughly (q/2p)^n which rapidly approaches zero (0) as "n" increases. Therefore the security parameter, k, SHOULD be set sufficiently large such that the probability that finding PE would take more than k iterations is sufficiently small.

4.1.2. Computing an FFC Password Element

The group-specific operation for FFC groups takes pwd-value, and the prime, p, and order, q, from the group's domain parameter set (see Section 3.2.2 when the order is not part of the defined domain parameter set) to directly produce a candidate password element, by exponentiating the pwd-value to the value ((p-1)/q) modulo the prime. If the result is greater than one (1), the candidate password element becomes PE, and the hunting and pecking terminates successfully.

Algorithmically, the process looks like this:

```
found = 0
counter = 0
do {
  counter = counter + 1
  pwd-seed = H(base | counter | p)
  pwd-value = PRF(pwd-seed, "TLS-PWD Hunting And Pecking",
                  ClientHello.random | ServerHello.random) [0..p]
  if (pwd-value < p)
  then
    PE = pwd-value ^ ((p-1)/q) mod p
    if (PE > 1)
    then
      found = 1
    fi
  fi
} while (found == 0)
```

Figure 3: Fixing PE for FFC Groups

4.2. Changes to Handshake Message Contents

4.2.1. Client Hello Changes

The client is required to identify herself to the server by adding a PWD extension to the Client Hello message. The PWD extension uses the standard mechanism defined in [RFC5246]. The "extension data" field of the PWD extension SHALL contain a PWD_name which is used to identify the password shared between the client and server.

```
enum { pwd(TBD) } ExtensionType;

opaque PWD_name<1..2^8-1>;
```

The PWD_name SHALL be UTF-8 encoded character string processed according to the rules of the [RFC4013] profile of [RFC3454].

A client offering a PWD ciphersuite MUST include the PWD extension in her Client Hello.

If a server does not have a password identified by the PWD_name in the PWD extension of the Client Hello, the server SHOULD hide that fact by simulating the protocol-- putting random data in the PWD-specific components of the Server Key Exchange-- and then rejecting the client's finished message with a "bad_record_mac" alert. To properly effect a simulated TLS-PWD exchange, an appropriate delay SHOULD be inserted between receipt of the Client Hello and response

of the Server Hello. Alternately, a server MAY choose to terminate the exchange if a password identified by the PWD_name in the PWD extension of the Client Hello is not found.

The server decides on a group to use with the named user (see Section 9 and generates the password element, PE, according to Section 4.1.2.

4.2.2. Server Key Exchange Changes

The domain parameter set for the selected group MUST be specified in the ServerKeyExchange, either explicitly or, in the case of some elliptic curve groups, by name. In addition to the group specification, the ServerKeyExchange also contains the server's "committment" in the form of a scalar and element, and the salt which was used to store the user's password.

Two new values have been added to the enumerated KeyExchangeAlgorithm to indicate TLS-PWD using finite field cryptography, ff_pwd, and TLS-PWD using elliptic curve cryptography, ec_pwd.

```
enum { ff_pwd, ec_pwd } KeyExchangeAlgorithms;

struct {
  opaque salt<1..2^8-1>;
  opaque pwd_p<1..2^16-1>;
  opaque pwd_g<1..2^16-1>;
  opaque pwd_q<1..2^16-1>;
  opaque ff_selement<1..2^16-1>;
  opaque ff_sscalar<1..2^16-1>;
} ServerFFPWDParams;

struct
  opaque salt<1..2^8-1>;
  ECParameters curve_params;
  ECPoint ec_selement;
  opaque ec_sscalar<1..2^8-1>;
} ServerECPWDParams;

struct {
  select (KeyExchangeAlgorithm) {
    case ec_pwd:
      ServerECPWDParams params;
    case ff_pwd:
      ServerFFPWDParams params;
  };
} ServerKeyExchange;
```

4.2.2.1. Generation of ServerKeyExchange

The scalar and Element that comprise the server's "committment" are generated as follows.

First two random numbers, called private and mask, between zero and the order of the group (exclusive) are generated. If their sum modulo the order of the group, q, equals zero the numbers must be thrown away and new random numbers generated. If their sum modulo the order of the group, q, is greater than zero the sum becomes the scalar.

```
scalar = (private + mask) mod q
```

The Element is then calculated as the inverse of the group's scalar operation (see the group specific operations in Section 3.2) with the mask and PE.

```
Element = inverse(scalar-op(mask, PE))
```

After calculation of the scalar and Element the mask SHALL be irretrievably destroyed.

4.2.2.1.1. ECC Server Key Exchange

EEC domain parameters are specified, either explicitly or named, in the ECParameters component of the EEC-specific ServerKeyExchange as defined in [RFC4492]. The scalar SHALL become the ec_sscalar component and the Element SHALL become the ec_selement of the ServerKeyExchange. If the client requested a specific point format (compressed or uncompressed) with the Support Point Formats Extension (see [RFC4492]) in its Client Hello, the Element MUST be formatted in the ec_selement to conform to that request.

As mentioned in Section 3.2.1, elliptic curves over GF(2^m), so called characteristic-2 curves, and curves with a co-factor greater than one (1) SHALL NOT be used with TLS-PWD.

4.2.2.1.2. FFC Server Key Exchange

FFC domain parameters sent in the ServerKeyExchange are for the group's prime, generator (which is only used for verification of the group specification), and the order of the group's generator. The scalar SHALL become the ff_sscalar component and the Element SHALL become the ff_selement in the FFC-specific ServerKeyExchange.

As mentioned in Section 3.2.2 if the prime is a safe prime and no order is included in the domain parameter set, the order added to the

ServerKeyExchange SHALL be the prime minus one divided by two-- (p-1)/2.

4.2.2.2. Processing of ServerKeyExchange

Upon receipt of the ServerKeyExchange, the client decides whether to support the indicated group or not. Named elliptic curves are easy to validate-- either they are supported or they are not, but care must be taken with FFC groups and explicitly specified ECC groups. As mentioned in Section 3.5, the discrete logarithm problem MUST be hard for any group used with this memo. The specific steps taken to come to this assurance for a particular group are outside the scope of this memo but they are the same steps to take when using the Diffie-Hellman key exchange with TLS. If the client decides not to support the group indicated in the ServerKeyExchange, she MUST abort the exchange.

If the client decides to support the indicated group the server's "commitment" MUST be validated by ensuring that: 1) the server's scalar value is greater than zero (0) and less than the order of the group, q; and 2) that the Element is valid for the chosen group (see Section 3.2.2 and Section 3.2.1 for how to determine whether an Element is valid for the particular group. Note that if the Element is a compressed point on an elliptic curve it MUST be uncompressed before checking its validity).

If the group is acceptable, the client extracts the salt from the ServerKeyExchange and generates the password element, PE, according to Section 3.4 and Section 4.1.2.

4.2.3. Client Key Exchange Changes

When the value of KeyExchangeAlgorithm is either ff_pwd or ec_pwd, the ClientKeyExchange is used to convey the client's "committment" to the server. It, therefore, contains a scalar and an Element.

```
struct {
  opaque ff_celement<1..2^16-1>;
  opaque ff_cscalar<1..2^16-1>;
} ClientFFPWDParams;

struct
  ECPoint ec_celement;
  opaque ec_cscalar<1..2^8-1>;
} ClientECPWDParams;

struct {
  select (KeyExchangeAlgorithm) {
    case ff_pwd: ClientFFPWDParams;
    case ec_pwd: ClientECPWDParams;
  } exchange_keys;
} ClientKeyExchange;
```

4.2.3.1. Generation of Client Key Exchange

The client’s scalar and Element are generated in the manner described in Section 4.2.2.1.

For an FFC group, the scalar SHALL become the ff_cscalar component and the Element SHALL become the ff_celement in the FFC-specific ClientKeyExchange.

For an ECC group, the scalar SHALL become the ec_cscalar component and the ELement SHALL become the ec_celement in the ECC-specific ClientKeyExchange. If the client requested a specific point format (compressed or uncompressed) with the Support Point Formats Extension in its ClientHello, then the Element MUST be formatted in the ec_celement to conform to its initial request.

4.2.3.2. Processing of Client Key Exchange

The server MUST validate the client’s "committment" by ensuring that: 1) the client’s scalar value is greater than zero (0) and less than the order of the group, q; and 2) that the Element is valid for the chosen group (see Section 3.2.2 and Section 3.2.1 for how to determin whether an Element is valid for a particular group. Note that if the Element is a compressed point on an elliptic curve it MUST be uncompressed before checking its validity.

4.3. Computing the Premaster Secret

The client uses her own scalar and Element, denoted here ClientKeyExchange.scalar and ClientKeyExchange.Element, the server’s scalar and Element, denoted here as ServerKeyExchange.scalar and

ServerKeyExchange.Element, and the random private value, denoted here as client.private, she created as part of the generation of her "commit" to compute an intermediate value, z, as indicated:

```
z = F(scalar-op(client.private,
                element-op(ServerKeyExchange.Element,
                           scalar-op(ServerKeyExchange.scalar, PE))))
```

With the same notation as above, the server uses his own scalar and Element, the client's scalar and Element, and his random private value, denoted here as server.private, he created as part of the generation of his "commit" to compute the premaster secret as follows:

```
z = F(scalar-op(server.private,
                element-op(ClientKeyExchange.Element,
                           scalar-op(ClientKeyExchange.scalar, PE))))
```

The intermediate value, z, is then used as the premaster secret after any leading bytes of z that contain all zero bits have been stripped off.

5. Ciphersuite Definition

This memo adds the following ciphersuites:

```
    CipherSuite TLS_FFCPWD_WITH_3DES_EDE_CBC_SHA = ( TBD, TBD );

    CipherSuite TLS_FFCPWD_WITH_AES_128_CBC_SHA = (TBD, TBD );

    CipherSuite TLS_ECCPWD_WITH_AES_128_CBC_SHA = (TBD, TBD );

    CipherSuite TLS_ECCPWD_WITH_AES_128_GCM_SHA256 = (TBD, TBD );

    CipherSuite TLS_ECCPWD_WITH_AES_256_GCM_SHA384 = (TBD, TBD );

    CipherSuite TLS_FFCPWD_WITH_AES_128_CCM_SHA = (TBD, TBD );

    CipherSuite TLS_ECCPWD_WITH_AES_128_CCM_SHA = (TBD, TBD );

    CipherSuite TLS_ECCPWD_WITH_AES_128_CCM_SHA256 = (TBD, TBD );

    CipherSuite TLS_ECCPWD_WITH_AES_256_CCM_SHA384 = (TBD, TBD );
```

Implementations conforming to this specification MUST support the TLS_ECCPWD_WITH_AES_128_CCM_SHA ciphersuite; they SHOULD support TLS_FFCPWD_WITH_AES_128_CCM_SHA, TLS_FFCPWD_WITH_AES_128_CBC_SHA,

TLS_ECCPWD_WITH_AES_128_CBC_SHA, TLS_ECCPWD_WITH_AES_128_GCM_SHA256, TLS_ECCPWD_WITH_AES_256_GCM_SHS384; and MAY support the remaining ciphersuites.

When negotiated with a version of TLS prior to 1.2, the Pseudo-Random Function (PRF) from that version is used; otherwise, the PRF is the TLS PRF [RFC5246] using the hash function indicated by the ciphersuite. Regardless of the TLS version, the TLS-PWD random function, H, is always instantiated with the hash algorithm indicated by the ciphersuite.

For those ciphersuites that use Cipher Block Chaining (CBC) [SP800-38A] mode, the MAC is HMAC [RFC2104] with the hash function indicated by the ciphersuite.

6. Acknowledgements

The authenticated key exchange defined here has also been defined for use in 802.11 networks, as an EAP method, and as an authentication method for IKE. Each of these specifications has elicited very helpful comments from a wide collection of people that have allowed the definition of the authenticated key exchange to be refined and improved.

The authors would like to thank Scott Fluhrer for discovering the "password as exponent" attack that was possible in an early version of this key exchange and for his very helpful suggestions on the techniques for fixing the PE to prevent it. The authors would also like to thank Hideyuki Suzuki for his insight in discovering an attack against a previous version of the underlying key exchange protocol. Special thanks to Lily Chen for helpful discussions on hashing into an elliptic curve and to Jin-Meng Ho for suggesting the countermeasures to protect against a small sub-group attack. Rich Davis suggested the defensive checks that are part of the processing of the ServerKeyExchange and ClientKeyExchange messages, and his various comments have greatly improved the quality of this memo and the underlying key exchange on which it is based.

Martin Rex, Peter Gutmann, Marsh Ray, and Rene Struik, discussed the possibility of a side-channel attack against the hunting-and-pecking loop on the TLS mailing list. That discussion prompted the addition of the security parameter, k, to the hunting-and-pecking loop.

7. IANA Considerations

IANA SHALL assign a value for a new TLS extention type from the TLS

ExtensionType Registry defined in [RFC5246] with the name "pwd". The RFC editor SHALL replace TBD in Section 4.2.1 with the IANA-assigned value for this extension.

IANA SHALL assign nine new ciphersuites from the TLS Ciphersuite Registry defined in [RFC5246] for the following ciphersuites:

```
CipherSuite TLS_FFCPWD_WITH_3DES_EDE_CBC_SHA = ( TBD, TBD );

CipherSuite TLS_FFCPWD_WITH_AES_128_CBC_SHA = (TBD, TBD );

CipherSuite TLS_ECCPWD_WITH_AES_128_CBC_SHA = (TBD, TBD );

CipherSuite TLS_ECCPWD_WITH_AES_128_GCM_SHA256 = (TBD, TBD );

CipherSuite TLS_ECCPWD_WITH_AES_256_GCM_SHA384 = (TBD, TBD );

CipherSuite TLS_FFCPWD_WITH_AES_128_CCM_SHA = (TBD, TBD );

CipherSuite TLS_ECCPWD_WITH_AES_128_CCM_SHA = (TBD, TBD );

CipherSuite TLS_ECCPWD_WITH_AES_128_CCM_SHA256 = (TBD, TBD );

CipherSuite TLS_ECCPWD_WITH_AES_256_CCM_SHA384 = (TBD, TBD );
```

The RFC editor SHALL replace (TBD, TBD) in all the ciphersuites defined in Section 5 with the appropriate IANA-assigned values.

8. Security Considerations

A passive attacker against this protocol will see the ServerKeyExchange and the ClientKeyExchange containing the server's scalar and Element, and the client's scalar and Element, respectively. The client and server effectively hide their secret private value by masking it modulo the order of the selected group. If the order is "q", then there are approximately "q" distinct pairs of numbers that will sum to the scalar values observed. It is possible for an attacker to iterate through all such values but for a large value of "q", this exhaustive search technique is computationally infeasible. The attacker would have a better chance in solving the discrete logarithm problem, which we have already assumed (see Section 3.5) to be an intractable problem.

A passive attacker can take the Element from either the ServerKeyExchange or the ClientKeyExchange and try to determine the random "mask" value used in its construction and then recover the other party's "private" value from the scalar in the same message.

But this requires the attacker to solve the discrete logarithm problem which we assumed was intractable.

Both the client and the server obtain a shared secret, the premaster secret, based on a secret group element and the private information they contributed to the exchange. The secret group element is based on the password. If they do not share the same password they will be unable to derive the same secret group element and if they don't generate the same secret group element they will be unable to generate the same premaster secret. Seeing a finished message along with the ServerKeyExchange and ClientKeyExchange will not provide any additional advantage of attack since it is generated with the unknowable premaster secret.

An active attacker impersonating the client can induce a server to send a ServerKeyExchange containing the server's scalar and Element. It can attempt to generate a ClientKeyExchange and send to the server but the attacker is required to send a finished message first so the only information she can obtain in this attack is less than the information she can obtain from a passive attack, so this particular active attack is not very fruitful.

An active attacker can impersonate the server and send a forged ServerKeyExchange after receiving the ClientHello. The attacker then waits until it receives the ClientKeyExchange and finished message from the client. Now the attacker can attempt to run through all possible values of the password, computing PE (see Section 4.1), computing candidate premaster secrets (see Section 4.3), and attempting to recreate the client's finished message.

But the attacker committed to a single guess of the password with her forged ServerKeyExchange. That value was used by the client in her computation of the premaster secret which was used to produce the finished message. Any guess of the password which differs from the one used in the forged ServerKeyExchange would result in each side using a different PE in the computation of the premaster secret and therefore the finished message cannot be verified as correct, even if a subsequent guess, while running through all possible values, was correct. The attacker gets one guess, and one guess only, per active attack.

Instead of attempting to guess at the password, an attacker can attempt to determine PE and then launch an attack. But PE is determined by the output of the random function, H, which is indistinguishable from a random source since H is assumed to be a "random oracle" (Section 3.5). Therefore, each element of the finite cyclic group will have an equal probability of being the PE. The probability of guessing PE will be 1/q, where q is the order of the

group. For a large value of "q" this will be computationally infeasible.

The implications of resistance to dictionary attack are significant. An implementation can provision a password in a practical and realistic manner-- i.e. it MAY be a character string and it MAY be relatively short-- and still maintain security. The nature of the pool of potential passwords determines the size of the pool, D, and countermeasures can prevent an attacker from determining the password in the only possible way: repeated, active, guessing attacks. For example, a simple four character string using lower-case English characters, and assuming random selection of those characters, will result in D of over four hundred thousand. An attacker would need to mount over one hundred thousand active, guessing attacks (which will easily be detected) before gaining any significant advantage in determining the pre-shared key.

Countermeasures to deal with successive active, guessing attacks are only possible by noticing a certain username is failing repeatedly over a certain period of time. Attacks which attempt to find a password for a random user are more difficult to detect. For instance, if a device uses a serial number as a username and the pool of potential passwords is sufficiently small, a more effective attack would be to select a password and try all potential "users" to disperse the attack and confound countermeasures. It is therefore RECOMMENDED that implementations of TLS-pwd keep track of the total number of failed authentications regardless of username in an effort to detect and thwart this type of attack.

The benefits of resistance to dictionary attack can be lessened by a client using the same passwords with multiple servers. An attacker could re-direct a session from one server to the other if the attacker knew that the intended server stored the same password for the client as another server.

An adversary that has access to, and a considerable amount of control over, a client or server could attempt to mount a side-channel attack to determine the number of times it took for a certain password (plus client random and server random) to select a password element. Each such attack could result in a successive paring-down of the size of the pool of potential passwords, resulting in a manageably small set from which to launch a series of active attacks to determine the password. A security parameter, k, is used to normalize the amount of work necessary to determine the password element (see Section 4.1). The probability that a password will require more than k iterations is roughly (q/2p)^k so it is possible to mitigate a side channel attack at the expense of a constant cost per connection attempt.

9. Implementation Considerations

The selection of the ciphersuite and selection of the particular finite cyclic group to use with the ciphersuite are divorced in this memo but they remain intimately close.

It is RECOMMENDED that implementations take note of the strength estimates of particular groups and to select a ciphersuite providing commensurate security with its hash and encryption algorithms. A ciphersuite whose encryption algorithm has a keylength less than the strength estimate, or whose hash algorithm has a blocksize that is less than twice the strength estimate SHOULD NOT be used.

For example, the elliptic curve named secp256r1 (whose IANA-assigned number is 23) provides an estimated 128 bits of strength and would be compatible with an encryption algorithm supporting a key of that length, and a hash algorithm that has at least a 256-bit blocksize. Therefore, a suitable ciphersuite to use with secp256r1 could be TLS_ECCPWD_WITH_AES_128_GCM_SHA256.

Resistance to dictionary attack means that the attacker must launch an active attack to make a single guess at the password. If the size of the pool from which the password was extracted was D, and each password in the pool has an equal probability of being chosen, then the probability of success after a single guess is 1/D. After X guesses, and removal of failed guesses from the pool of possible passwords, the probability becomes 1/(D-X). As X grows so does the probability of success. Therefore it is possible for an attacker to determine the password through repeated brute-force, active, guessing attacks. Implementations SHOULD take note of this fact and choose an appropriate pool of potential passwords-- i.e. make D big. Implementations SHOULD also take countermeasures, for instance refusing authentication attempts by a particular username for a certain amount of time, after the number of failed authentication attempts reaches a certain threshold. No such threshold or amount of time is recommended in this memo.

10. References

10.1. Normative References

[RFC2104] Krawczyk, H., Bellare, M., and R. Canetti, "HMAC: Keyed-Hashing for Message Authentication", RFC 2104, February 1997.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[RFC3454] Hoffman, P. and M. Blanchet, "Preparation of Internationalized Strings ("stringprep")", RFC 3454, December 2002.

[RFC4013] Zeilenga, K., "SASLprep: Stringprep Profile for User Names and Passwords", RFC 4013, February 2005.

[RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, August 2008.

[SP800-38A] National Institute of Standards and Technology, "Recommendation for Block Cipher Modes of Operation-- Methods and Techniques", NIST Special Publication 800-38A, December 2001.

10.2. Informative References

[RANDOR] Bellare, M. and P. Rogaway, "Random Oracles are Practical: A Paradigm for Designing Efficient Protocols", Proceedings of the 1st ACM Conference on Computer and Communication Security, ACM Press, 1993.

[RFC4086] Eastlake, D., Schiller, J., and S. Crocker, "Randomness Requirements for Security", BCP 106, RFC 4086, June 2005.

[RFC4492] Blake-Wilson, S., Bolyard, N., Gupta, V., Hawk, C., and B. Moeller, "Elliptic Curve Cryptography (ECC) Cipher Suites for Transport Layer Security (TLS)", RFC 4492, May 2006.

[RFC5751] Ramsdell, B. and S. Turner, "Secure/Multipurpose Internet Mail Extensions (S/MIME) Version 3.2 Message Specification", RFC 5751, January 2010.

[RFC5967] Turner, S., "The application/pkcs10 Media Type", RFC 5967, August 2010.

[RFC6090] McGrew, D., Igoe, K., and M. Salter, "Fundamental Elliptic Curve Cryptography Algorithms", RFC 6090, February 2011.

[SP800-56A] Barker, E., Johnson, D., and M. Smid, "Recommendations for Pair-Wise Key Establishment Schemes Using Discrete Logarithm Cryptography", NIST Special Publication 800-56A, March 2007.

Authors' Addresses

Dan Harkins (editor)
Aruba Networks
1322 Crossman Avenue
Sunnyvale, CA 94089-1113
United States of America

Email: dharkins@arubanetworks.com

Dave Halasz (editor)
Halasz Ventures
8401 Chagrin Road, Suite 10A
Chagrin Falls, OH 44023
United States of America

Email: david.e.halasz@gmail.com

IETF
Internet-Draft
Intended status: Standards Track
Expires: May 20, 2012

P. Wouters
Xelerance
J. Gilmore

S. Weiler
SPARTA, Inc.
T. Kivinen
AuthenTec
H. Tschofenig
Nokia Siemens Networks

November 17, 2011

TLS out-of-band public key validation
draft-wouters-tls-oob-pubkey-02

Abstract

This document specifies a new TLS certificate type for exchanging raw public keys in Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS) for use with out-of-band authentication. Currently, TLS authentication can only occur via PKIX or OpenPGP certificates. By specifying a minimum resource for raw public key exchange, implementations can use alternative authentication methods.

One such method is using DANE Resource Records secured by DNSSEC, Another use case is to provide authentication functionality when used with devices in a constrained environment that use whitelists and blacklists, as is the case with sensors and other embedded devices that are constrained by memory, computational, and communication limitations where the usage of PKIX is not feasible.

The new certificate type specified can also be used to reduce the latency of a TLS client that is already in possession of a validated public key of the TLS server before it starts a (non-resumed) TLS handshake.

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 May 20, 2012.

Copyright Notice

Table of Contents

1. Introduction

1.1. Motivation

Traditionally, TLS server public keys are obtained in PKIX containers in-band using the TLS connection and validated using trust anchors based on a [PKIX] certification authority (CA). This method can add a complicated trust relationship that is difficult to validate. Examples of such complexity can be seen in [Defeating-SSL].

Alternative methods are available that allow a TLS client to obtain the TLS server public key:

- The TLS server public key is obtained from a DNSSEC secured RRset using [DANE]
- The TLS server public key is obtained from a [PKIX] certificate chain from an [LDAP] server
- The TLS server public key is provisioned by the operating system and updated via software updates
- A TLS client has connected to the TLS server before and has cached the TLS server certificate chain or TLS server public key for re-use

[RFC5246] does not provide a mechanism for a TLS client to tell the TLS server it is already in possession of the authenticated public key. Therefore, a TLS server must always send a list of trusted CA keys and its EE certificate containing its public key, even when the TLS client does not require or desire that data for authentication.

[RFC6066] allows suppression of the certificate trust anchor chain, but not suppression of the PKIX EE certificate container. These certificate chains are large opaque blocks of data containing much more than the public key of the TLS server. Since the TLS client might only be able to validate the PKIX SubjectPublicKeyInfo via an out-of-band method such as [DANE], it has to ignore any additional information received that was sent by the server that it could not validate. Furthermore, information that comes in via these certificate chains could contain contradicting or additional information that the TLS client cannot validate or trust, such as an expiry date that conflicts with information obtained from DNS or LDAP. This document specifies a method to suppress sending this additional information.

Some small embedded devices use the UDP based [CoAP], a specialized constrained networks and nodes for machine-to-machine applications.

These devices interact with a Web server to upload data such as temperature sensor readings at a regular intervals. Constrained Application Protocol (CoAP) [CoAP] can utilize DTLS for its communication security. As part of the provisioning procedure, the embeded device is configured with the address and public key of a dedicated CoAP server to upload sensor data. Receiving PKIX information [PKIX] from a webserver would be an unneccesarry burden on a sensor networking deployment environment that requires pre-configured client-server public keys. These devices often also lack a real-time clock to perform any PKIX epixry checks.

1.2. Applicability

The Transport Layer Security (TLS) Protocol Version 1.2 is specified in [RFC5246] and provides a framework for extensions to TLS as well as considerations for designing such extensions. [RFC6066] defines several new TLS extensions. This document extends the specifications of those RFCs with one new TLS Certificate Type to facilitate suppressing unneeded [PKIX] information from being sent during the TLS handshake when this information is not required to authenticate the TLS server.

1.3. Terminology

Most security-related terms in this document are to be understood in the sense defined in [SECTERMS]; such terms include, but are not limited to, "attack", "authentication", "authorization", "certification authority", "certification path", "certificate", "credential", "identity", "self-signed certificate", "trust", "trust anchor", "trust chain", "validate", and "verify".

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. Changes to the Handshake Message Contents

This section describes the changes to the TLS handshake message contents when raw public keys are to be used for authentication. Figure 1 illustrates the exchange of messages as described in the sub-sections below. The new "RawPublicKey" value in the cert_type extension indicates the ability and desire to exchange raw public keys, which are then exchanged as part of the certificate payloads.

```
    client_hello,
    cert_type="RawPublicKey" ->

                               <-  server_hello,
                                   cert_type="RawPublicKey",
                                   certificate,
                                   server_key_exchange,
                                   certificate_request,
                                   server_hello_done

    certificate,
    client_key_exchange,
    certificate_verify,
    change_cipher_spec,
    finished                   ->

                               <- change_cipher_spec,
                                  finished

   Application Data        <------->     Application Data
```

Figure 1: Example Message Flow

2.1. Client Hello

In order to indicate the support of out-of-bound raw public keys, clients MUST include an extension of type "cert_type" to the extended client hello message. The "cert_type" TLS extension, which is defined in [RFC6091], is assigned the value of 9 from the TLS ExtensionType registry. This value is used as the extension number for the extensions in both the client hello message and the server hello message. The hello extension mechanism is described in [RFC5246].

The "cert_type" TLS extension carries a list of supported certificate types the client can use, sorted by client preference. This extension MUST be omitted if the client only supports X.509 certificates. The "extension_data" field of this extension contains a CertificateTypeExtension structure. Note that the CertificateTypeExtension structure is being used both by the client and the server, even though the structure is only specified once in this document.

The [RFC6091] defined CertificateTypeExtension is extended as follows:

```
enum { client, server } ClientOrServerExtension;

enum { X.509(0), OpenPGP(1),
   RawPublicKey([TBD]),
   (255) } CertificateType;

struct {
   select(ClientOrServerExtension)
       case client:
         CertificateType certificate_types<1..2^8-1>;
       case server:
         CertificateType certificate_type;
   }
} CertificateTypeExtension;
```

No new cipher suites are required to use raw public keys. All existing cipher suites that support a key exchange method compatible with the defined extension can be used.

2.2. Server Hello

If the server receives a client hello that contains the "cert_type" extension and chooses a cipher suite then two outcomes are possible. The server MUST either select a certificate type from the certificate_types field in the extended client hello or terminate the session with a fatal alert of type "unsupported_certificate".

The certificate type selected by the server is encoded in a CertificateTypeExtension structure, which is included in the extended server hello message using an extension of type "cert_type". Servers that only support X.509 certificates MAY omit including the "cert_type" extension in the extended server hello.

If the negotiated certificate type is RawPublicKey the TLS server MUST send a CertificateTypeExtension structure with a PKIX [PKIX] certificate containing ONLY the SubjectPublicKeyInfo. The public key MUST match the selected key exchange algorithm.

2.3. Certificate Request

The semantics of this message remain the same as in the TLS specification. However, if this message is sent, and the negotiated certificate type is RawPublicKey, the "certificate_authorities" list MUST be empty.

2.4. Other Handshake Messages

All the other handshake messages are identical to the TLS specification.

3. Security Considerations

The TLS cert_type extension defined here lets a TLS client attempt to supress the sending of server certificate as well as the certification chain for that certificate.

A client using this cert_type needs to be confident in the authenticity of the public key it is using. Since those public keys were obtained out-of-band extension), the authentication must also be out-of-band.

Depending on exactly how the public keys were obtained, it may be appropriate to use authentication mechanisms tied to the public key transport. For example, if public keys were obtained using [DANE] it is appropriate to use DNSSEC to authenticate the public keys.

4. IANA Considerations

We request that IANA assign a TLS cert_type value for RawPublicKey.

5. Contributors

The following individuals made important contributions to this document: Paul Hoffman.

6. Acknowledgements

This document is based on material from RFC 6066 for which the author is Donald Eastlake 3rd. Contributions to that document also include Joseph Salowey, Alexey Melnikov, Peter Saint-Andre, and Adrian Farrel.

The second version of this document was made after feedback from the TLS working group at IETF#81. The support for hashes of public keys has been removed after the discussions at the IETF#82 meeting.

7. References

7.1. Normative References

[PKIX] Cooper, D., Santesson, S., Farrell, S., Boeyen, S., Housley, R., and W. Polk, "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile", RFC 5280, May 2008.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, August 2008.

[SECTERMS] Shirey, R., "Internet Security Glossary, Version 2", RFC 4949, August 2007.

7.2. Informative References

[CoAP] Shelby, Z., Hartke, K., Bormann, C., and B. Frank, "Constrained Application Protocol", draft-ietf-core-coap-07 (work in progress), July 2011.

[DANE] Hoffman, P. and J. Schlyter, "Using Secure DNS to Associate Certificates with Domain Names For TLS", draft-ietf-dane-protocol-12 (work in progress), September 2011.

[Defeating-SSL] Marlinspike, M., "New Tricks for Defeating SSL in Practice", February 2009, <http://www.blackhat.com/presentations/bh-dc-09/Marlinspike/BlackHat-DC-09-Marlinspike-Defeating-SSL.pdf>.

[LDAP] Sermersheim, J., "Lightweight Directory Access Protocol (LDAP): The Protocol", RFC 4511, June 2006.

[RFC6066] Eastlake, D., "Transport Layer Security (TLS) Extensions: Extension Definitions", RFC 6066, January 2011.

[RFC6091] Mavrogiannopoulos, N. and D. Gillmor, "Using OpenPGP Keys for Transport Layer Security (TLS) Authentication", RFC 6091, February 2011.

Authors' Addresses

Paul Wouters
Xelerance
4130 Ramsayville Road
Ottawa, On K1G 3N4
Canada

Phone: +1-647-722-5653
Email: paul@xelerance.com
URI: https://www.xelerance.com/

John Gilmore
PO Box 170608
San Francisco, California 94117
USA

Phone: +1 415 221 6524
Email: gnu@toad.com
URI: https://www.toad.com/

Samuel Weiler
SPARTA, Inc.
7110 Samuel Morse Drive
Columbia, Maryland 21046
US

Email: weiler@tislabs.com

Tero Kivinen
AuthenTec
Eerikinkatu 28
HELSINKI FI-00180
FI

Email: kivinen@iki.fi

Hannes Tschofenig
Nokia Siemens Networks
Linnoitustie 6
Espoo 02600
Finland

Phone: +358 (50) 4871445
Email: Hannes.Tschofenig@gmx.net
URI: http://www.tschofenig.priv.at