Internet-Draft
November 2006
Expires: May 2007

M. Brown
RedPhone Security
R. Housley
Vigil Security

Transport Layer Security (TLS) Evidence Extensions
<draft-housley-evidence-extns-01.txt>

Status of this Memo

Copyright Notice

Abstract

This document specifies evidence creation extensions to the Transport Layer Security (TLS) Protocol. Extension types are carried in the client and server hello message extensions to confirm that both parties support the protocol features needed to perform evidence creation. The syntax and semantics of the evidence creation alerts and messages are described in detail.

1. Introduction

Transport Layer Security (TLS) protocol [TLS1.0][TLS1.1] is being used in an increasing variety of operational environments, including ones that were not envisioned when the original design criteria for TLS were determined. The extensions specified in this document support evidence creation in environments where the peers in the TLS session cooperate to create persistent evidence of the TLS-protected application data. Such evidence may be necessary to meet business requirements, including regulatory requirements. Also, evidence may be used in tandem with authorization information to make high assurance access control and routing decisions in military and government environments. Evidence created using this extension may also be used to audit various aspects of the TLS handshake, including the cipher suite negotiation and the use of other TLS extensions. In many cases, the evidence does not need to be validated by third parties; however, in other cases, the evidence might be validated by third parties. To accommodate all of these situations, the evidence is generated using a digital signature. Since validation of a digital signature requires only public information, evidence generated with this mechanism is suitable for sharing with third parties.

When digital certificates are to be employed in evidence creations, the client must obtain the public key certificate (PKC) for the server, and the server must obtain the PKC for the client. This is most easily accomplished by using the PKCs provided in the Handshake Protocol Certificate messages. Further, both parties SHOULD have an opportunity to validate the PKC that will be used by the other party before evidence creation. Again, this is naturally accomplished using the Handshake Protocol, where the TLS session can be rejected if the PKC cannot be validated.

This document describes evidence creation TLS extensions supporting both TLS 1.0 and TLS 1.1. These extensions observe the conventions defined for TLS Extensions [TLSEXT]. The extensions are designed to be backwards compatible, meaning that the protocol alerts and messages associated with the evidence creation extensions will be exchanged only if the client indicates support for them in the client hello message and the server indicates support for them in the server hello message.

Clients typically know the context of the TLS session before it is established. As a result, the client can request the use of the evidence creation extensions in sessions where they might be needed. Servers accept extended client hello messages, even if the server does not support the all of the listed extensions. However, the server will not indicate support for any extensions that are not

"understood" by the implementation. At the end of the hello message exchange, the client may reject communications with servers that do not support the evidence creation extensions, or the client may accept the situation and proceed, whichever is appropriate.

1.1. Conventions

The syntax for the evidence creation messages is defined using the TLS Presentation Language, which is specified in Section 4 of [TLS1.0].

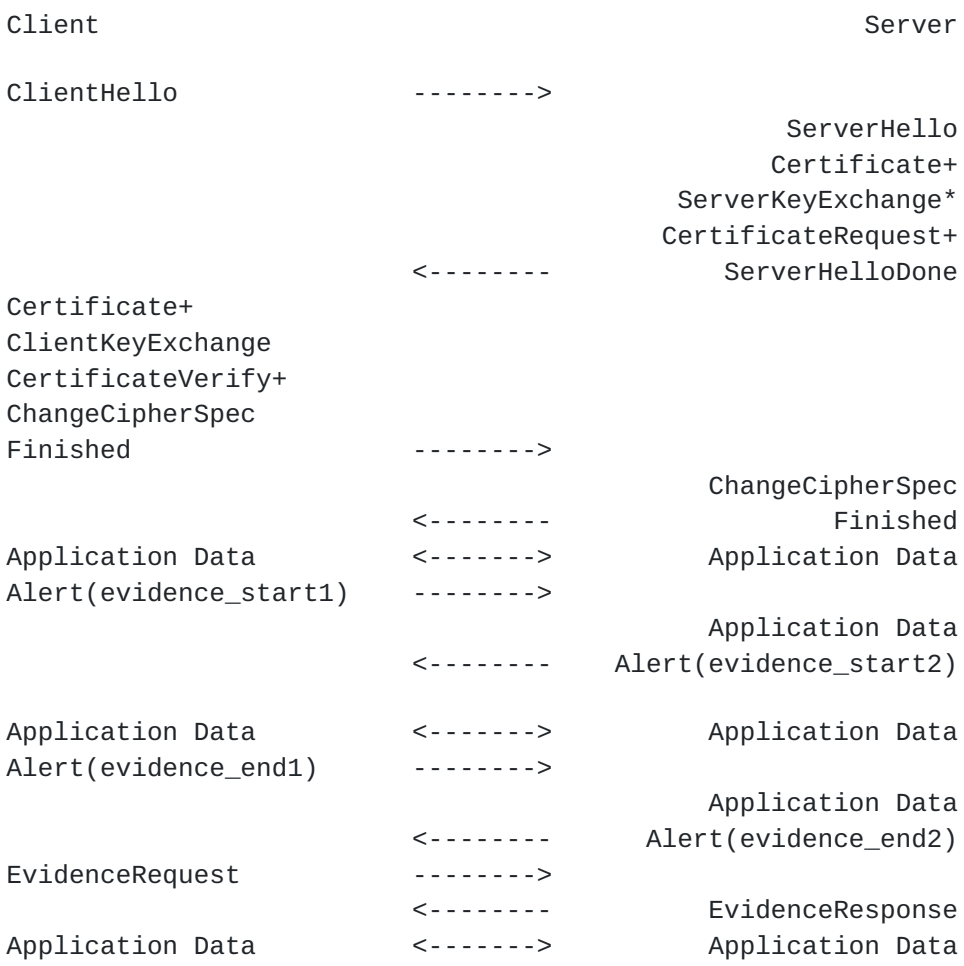
The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [STDWORDS].

1.2. Overview

Figure 1 illustrates the placement of the evidence creation alerts and messages in the TLS session. The first pair of evidence creation alerts indicates the beginning of the protected content that will be covered by the evidence. The first pair of alerts can appear any place after the TLS Handshake Protocol Finished messages, which ensures that they are integrity protected. The second pair of evidence creation alerts indicates the ending of the protected content that will be covered by the evidence. Immediately after the reception of the final alert, a pair of Evidence Protocol messages is exchanged to create the persistent evidence.

Generating evidence is not compatible with Diffie-Hellman Ephemeral (DHE) key exchanges. DHE does not permit the same keying material to be generated for validation of the evidence after the TLS session is over. Persistent evidence requires the use of a digital signature so that it can be validated well after the TLS session is over.

The ClientHello message includes an indication that the evidence creation messages are supported. The ServerHello message also includes an indication that the evidence creation messages are supported. Both the client and the server MUST indicate support for the evidence protocol alerts and messages; otherwise they MUST NOT be employed by either the client or the server.



* Indicates optional or situation-dependent messages that are not always sent.

\+ Indicates messages optional to the TLS handshake that MUST be sent when using TLS evidence.

Figure 1. Example TLS Session with Evidence Creation.

2. Evidence Extension Types

The general extension mechanisms enable clients and servers to negotiate whether to use specific extensions, and how to use specific extensions. As specified in [TLSEXT], the extension format used in the extended client hello message and extended server hello message within the TLS Handshake Protocol is:

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

The extension_type identifies a particular extension type, and the extension_data contains information specific to the particular extension type.

As specified in [TLSEXT], for all extension types, the extension type MUST NOT appear in the extended server hello message unless the same extension type appeared in the preceding client hello message. Clients MUST abort the handshake if they receive an extension type in the extended server hello message that they did not request in the preceding extended client hello message.

When multiple extensions of different types are present in the extended client hello message or the extended server hello message, the extensions can appear in any order, but there MUST NOT be more than one extension of the same type.

This document specifies the use of the evidence_creation extension type. This specification adds one new type to ExtensionType:

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

The evidence_creation extension is relevant when a session is initiated and also for any subsequent session resumption. However, a client that requests resumption of a session does not know whether the server has maintained all of the context necessary to accept this request, and therefore the client SHOULD send an extended client hello message that includes the evidence_creation extension type. This indicates that the client requests the server's continued cooperation in creating evidence. If the server denies the resumption request, then the evidence_creation extension will be negotiated normally using the full Handshake protocol.

Clients MUST include the evidence_creation extension in the extended

client hello message to indicate their desire to send and receive evidence creation alerts and messages. The extension_data field indicates the evidence creation algorithms that are supported. The format is indicated with the EvidenceCreateList type:

```
uint16 EvidenceCreateSuite[2];

struct {
   EvidenceCreateSuite evidence_suites<2..2^16-1>;
} EvidenceCreateList;
```

The EvidenceCreateList enumerates the evidence creation algorithms that are supported by the client. The client MUST order the entries from most preferred to least preferred, but all of the entries MUST be acceptable to the client. Values are defined in Appendix A, and others can be registered in the future.

Servers that receive an extended client hello message containing the evidence_creation extension MUST respond with the evidence_creation extension in the extended server hello message if the server is willing to send and receive evidence creation alerts and messages. The evidence_creation extension MUST be omitted from the extended server hello message if the server is unwilling to send and receive using one of the evidence creation algorithm suites identified by the client. The extension_data field indicates the evidence creation algorithm suite that the server selected from the list provided by the client. The format is indicated with the EvidenceCreateSuite type defined above.

3. Alert Messages

This document specifies the use of four new alert message descriptions: the evidence_start1, evidence_start2, evidence_end1, and evidence_end2. These descriptions are specified in Sections 3.1, 3.2, 3.3, and 3.4, respectively. The alert descriptions are presented below in the order they MUST be sent; sending alerts in an unexpected order results in a fatal error. These descriptions are always used with the warning alert level. This specification adds four new types to AlertDescription:

```
enum {
   evidence_start1(TBD),
   evidence_start2(TBD),
   evidence_end1(TBD),
   evidence_end2(TBD),
   evidence_failure(TBD),
   (255)
} AlertDescription;
```

3.1. The evidence_start1 Description

The client and the server need to synchronize evidence creation. Either party may indicate the desire to start creating evidence by sending the evidence_start1 alert. If the other party is ready to begin creating evidence, then the other party MUST send an evidence_start2 alert in response to the evidence_start1 alert that was sent. If the other party is unwilling to begin creating evidence, the other party MUST respond with fatal evidence_failure(TBD) alert and terminate the TLS session.

Evidence may be collected more than once during a TLS session; however, evidence gathering MUST NOT be nested. That is, a party sending the a second evidence_start1 alert before evidence_end2 alert has occurred and the evidence protocol has completed is a protocol violation. Reception of an evidence_start1 alert that would result in evidence nesting MUST be responded to with a fatal evidence_failure(TBD) alert and terminating the TLS session.

Digital signatures are used in evidence creation. If an evidence_start1 alert is received before the other party has provided a valid PKC, then the evidence_start1 alert recipient MUST terminate the TLS session using a fatal certificate_unknown alert.

3.2. The evidence_start2 Description

The evidence_start2 alert is sent in response to the evidence_start1 alert. By sending the evidence_start2 alert, the sending party indicates that they are also ready to begin creating evidence. After this pair of alerts is exchanged, both the client and the server use the hash function indicated in the extended server hello message to start computing the evidence. Each party computes two independent hash values: one for each octet that is written, and one for each octet that is read.

Digital signatures are used in evidence creation. If an evidence_start2 alert is received before the other party has provided a valid PKC, then the evidence_start2 alert recipient MUST terminate the TLS session using a fatal certificate_unknown alert.

3.3. The evidence_end1 Description

Either party may initiate the closure of an evidence-creating interval and the exchange of evidence messages by sending the evidence_end1 alert. Upon sending evidence_end1, the sender MUST not send any more application data on this connection until the Evidence Protocol messages are exchanged.

The evidence_end1 alert sender MAY initiate the Evidence Protocol as described in Section 4 at any time following this alert. The evidence_end1 alert sender SHOULD ensure that it has received all pending application data writes from the other party before initiating the Evidence Protocol. One way to ensure that all of the application data has been received it to wait for the receipt of an evidence_end2 alert. If the Evidence Protocol begins before all of the application data is available, the result will be a fatal evidence_failure(TBD) alert when signature verification fails.

3.4. The evidence_end2 Description

The evidence_end2 alert is sent in response to the evidence_end1 alert. The evidence_end1 alert receiver SHOULD complete any pending writes. The intent is to include any application data that would be sent in response to application data that was received before the evidence_end1 alert as part of evidence creation. Once the pending writes are complete, the evidence_end1 alert receiver sends the evidence_end2 alert.

At this point, each party completes the hash value computations.

The evidence_end2 alert receiver MUST respond by initiating the Evidence Protocol as described in Section 4, if it has not already done so.

3.5. The evidence_failure Description

The evidence_failure fatal alert is sent to indicate a failure in evidence creation. During evidence synchronization, this alert indicates that the sending party is unwilling to begin evidence creation. During the Evidence Protocol, this alert indicates that the evidence provided by the other party is not acceptable or cannot be validated.

4. Evidence Protocol

This document specifies an additional TLS Protocol: the Evidence Protocol. It is used to create persistent evidence of the TLS session content. This specification adds one new Record layer ContentType:

```
enum {
   evidence(TBD),
   (255)
} ContentType;
```

Persistence evidence of the TLS session content is produced by the TLS Evidence Protocol. Evidence messages are supplied to the TLS Record Layer, where they are encapsulated within one or more TLSPlaintext structures, which are processed and transmitted as specified by the current active session state.

The Evidence Protocol structure follows:

```
enum {
   request(1), response(2), (255)
} EvidenceMsgType;

struct {
   EvidenceMsgType evidence_msg_type;
   uint24 length; /* number of octets in message */
   select (EvidenceMsgType) {
      case request:     EvidenceRequest;
      case response:    EvidenceResponse;
   } body;
} EvidenceProtocol;
```

The Evidence Protocol messages are presented below in the order they MUST be sent; sending evidence messages in an unexpected order results in a fatal unexpected_message(10) alert. The EvidenceRequest message and the EvidenceResponse message are specified in Section 4.2 and Section 4.3, respectively. Section 4.1 describes structures that are used in the EvidenceRequest and EvidenceResponse messages.

4.1. Certificates and Digital Signatures

The evidence Protocol makes use of the ASN.1Cert definition used elsewhere in TLS. It is repeated here for convenience.

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

The EvidenceSignature definition is very similar to the Signature definition used elsewhere in TLS. The EvidenceSignature definition signs hash[hash_size], but the Signature definition used elsewhere in TLS signs a combination of an md5_hash and a sha_hash. Also, the EvidenceSignature definition excludes the anonymous case.

```
enum { rsa, dsa, ecdsa } EvidenceSignatureAlgorithm;

select (EvidenceSignatureAlgorithm)
{   case rsa:
        digitally-signed struct {
            opaque hash[hash_size];
        };
    case dsa:
        digitally-signed struct {
            opaque hash[hash_size];
        };
    case ecdsa:
        digitally-signed struct {
            opaque hash[hash_size];
        };
} EvidenceSignature;
```

The hash algorithm and the hash_size depend on evidence create algorithm suite selected by the server in the evidence_creation extension.

4.2. EvidenceRequest Message

The EvidenceRequest message contains the evidence_end1 alert sender's contribution to the persistent evidence. It employs the evidence create algorithm suite selected by the server in the evidence_creation extension in the extended server hello message.

```
struct {
   Evidence evidence<1..2^16-1>;
   ASN.1Cert party1_certificate;
   EvidenceSignature party1_signature;
} EvidenceRequest;
```

```
struct {
   EvidenceCreateSuite evidence_suite;
   uint64 gmt_unix_time;
   uint64 app_data_sent_offset;
   uint64 app_data_received_offset;
   opaque handshake_protocol_hash<1..512>;
   opaque app_data_sent_hash<1..512>;
   opaque app_data_received_hash<1..512>;
} Evidence;
```

The elements of the EvidenceRequest structure are described below:

evidence
: Contains an evidence create algorithm identifier, a timestamp, and three hash values in the Evidence structure as described below.

party1_certificate
: Contains the X.509 certificate of the signer. While this certificate was probably exchanged and validated in the Handshake Protocol, inclusion here makes it clear which certificate was employed by the signer when the evidence is validated in the future, possibly by a third party.

party1_signature
: Contains the digital signature computed by the sender of the evidence_end1 alert using the evidence creation algorithm suite identified in evidence_create_suite. The hash value is computed as:

  Hash(evidence)

The elements of the Evidence structure are described below:

evidence_suite
: Indicates the evidence creation algorithm suite selected by the server in the evidence_creation extension in the Handshake Protocol. This value determines the structure of the hash values and digital signatures.

gmt_unix_time
: Indicates the current date and time according to the local system clock used by the sender of the evidence_end1 alert. This time value is intended to represent the moment in time that evidence_end1 was sent. Unlike other places in the TLS protocol, a 64-bit value is used to ensure that time values do not wrap in 2038.

app_data_sent_offset
Indicates the number of octets that were sent as part of this TLS session before evidence collection began.

app_data_received_offset
Indicates the number of octets that were received as part of this TLS session before evidence collection began.

handshake_protocol_hash
Compute as:

Hash(handshake_messages),

where handshake_messages refers to all Handshake Protocol messages sent or received, beginning with the most recent client hello message. If the double handshake mechanism described in the security considerations of [TLSAUTHZ] is used to encrypt the Handshake Protocol, the plaintext form of these messages is used in calculating this hash value.

Verification of the handshake_protocol_hash is performed using the plaintext form of the Handshake protocol messages. For this hash value to be validated at a later time, this information must be saved as part of the overall evidence. Any attempt to save this data must ensure that it is not inappropriately disclosed by employing suitable physical protection or cryptographic mechanisms that are at least as strong as the selected TLS ciphersuite. Suitable controls are discussed further in the Security Considerations; see Section 6.

In the case of successful TLS session resumption, the most recent client hello message will contain a valid ClientHello.session_id value as well as extensions, and these extensions may include sensitive data. The TLS Authorization Extension [AUTHZ] is one example where an extension might contain sensitive information. Thus, even when session resumption is employed, the content of the Handshake protocol messages ought to be protected.

TLS users should ensure that the content of the Handshake protocol messages contain sufficient evidence to determine the intent of the signers, where "signers" are defined as the subject identities in the exchanged X.509 certificates. Clients and servers MAY record the protocol messages containing an expression of the intent of the signers using a suitable TLS extension [TLSEXT], such as [TLSAUTHZ]. For example, a client may request access to a resource provided by the server,

provide sufficient authentication and authorization information grounds to convince the server to grant the requested access, and receive an affirmative response from the server. A record of TLS Handshake protocol messages representing this example may provide a sufficient record of the intent of both the client and the server.

app_data_sent_hash
Compute as:

Hash(sent_application_data),

where sent_application_data refers to all of the Application Data messages sent since the most recent evidence_start1 or evidence start2 alert was sent, and ending with the sending of the evidence_end1 alert. The alerts are not application data, and they are not included in the hash computation.

app_data_received_hash
Compute as:

Hash(received_application_data),

where received_application_data refers to all of the Application Data messages received since the most recent evidence_start1 or evidence start2 alert was received, and ending with the receipt of the evidence_end2 alert. The alerts are not application data, and they are not included in the hash computation.

4.3. EvidenceResponse Message

The EvidenceResponse message contains the complete persistent evidence. The value is saved by one or both parties as evidence of the TLS session content identified by the evidence_start1, evidence_start2, evidence_end1, and evidence_end2 alerts.

```
struct {
   Evidence evidence<1..2^16-1>;
   ASN.1Cert party1_certificate;
   EvidenceSignature party1_signature;
   ASN.1Cert party2_certificate;
   EvidenceSignature party2_signature;
} EvidenceResponse;
```

The elements of the EvidenceResponse structure are described below:

evidence
: Contains an evidence create algorithm identifier, a timestamp, and three hash values in the Evidence structure as described in section 4.2. The evidence creation algorithm MUST match the evidence create algorithm suite selected by the server in the evidence_creation extension in the extended server hello message. The timestamp MUST be acceptable to the EvidenceRequest recipient as the same value is provided in the EvidenceResponse message. The three hash values received in the EvidenceRequest message MUST match locally computed values over the same data. Note that the app_data_sent_hash and the app_data_received_hash values represent the session from the perspective of the EvidenceRequest originator, and the values are not swapped to represent the EvidenceRequest recipient perspective. If any of these conditions is not met, then the EvidenceResponse message MUST NOT be sent, and the TLS session MUST be closed immediately after sending a fatal evidence_failure(TBD) alert.

party1_certificate
: Contains the X.509 certificate of the sender of the evidence_end1 alert. If this certificate cannot be validated, then TLS session must be closed immediately after sending one of the following fatal alerts: bad_certificate(42), unsupported_certificate(43), certificate_revoked(44), or certificate_expired(45). These alerts are described in Section 7.2.2 of [TLS1.1].

party1_signature
: Contains the digital signature computed by the sender of the evidence_end1 alert. If this signature cannot be validated, then TLS session must be closed immediately after sending a fatal evidence_failure(TBD) alert.

party2_certificate
: Contains the X.509 certificate of the sender of the evidence_end2 alert. While this certificate was probably exchanged and validated in the Handshake Protocol, inclusion here make it clear which certificate was employed by the signer when the evidence is validated in the future, possibly by a third party.

Party2_signature
Contains the digital signature computed by the sender of the evidence_end2 alert using the evidence creation algorithm suite identified in evidence_suite. The hash value is computed as:

Hash(evidence)

5. IANA Considerations

This document defines one TLS extension: evidence_creation(TBD). This extension type value is assigned from the TLS Extension Type registry defined in [TLSEXT].

This document defines five TLS alert descriptions: the evidence_start1(TBD), evidence_start2(TBD), evidence_end1(TBD), evidence_end2(TBD), and evidence_failure(TBD). These alert descriptions are assigned from the TLS Alert registry defined in [TLS1.1].

This document defines one TLS ContentType: evidence(TBD). This ContentType value is assigned from the TLS ContentType registry defined in [TLS1.1].

This document establishes a registry for TLS Evidence Protocol EvidenceMsgType. The first two entries in the registry are request(1) and response(2). All additional TLS Evidence Protocol EvidenceMsgType values are assigned by Standards Action as described in [IANA].

This document establishes a registry for Evidence Create Algorithm suite identifiers. Appendix A lists the initial values for this registry. Evidence Create Algorithm suite identifier values with the first byte in the range 0-191 (decimal) inclusive are assigned by Standards Action as described in [IANA]. Values with the first byte in the range 192-254 (decimal) are assigned by Specification Required as described in [IANA]. Values with the first byte 255 (decimal) are reserved for Private Use as described in [IANA].

6. Security Considerations

This document describes an extension to the TLS protocol, and the security considerations in [TLS1.1] and [TLSEXT] apply. Additionally, temporal and storage security considerations are discussed below.

6.1. Temporal Considerations

Generating evidence that covers Application Data values that do not explicitly or implicitly indicate the point in time at which the Application Data was transferred over the TLS session might give rise to replay attacks, post-dating, pre-dating, or other temporal disputes. To avoid these concerns, the evidence includes an indication of the date and time. The TLS implementation MUST NOT attempt to extract date and time values from the Application Data; doing so is likely to be error prone. Instead, the date and time SHOULD come from a local clock or a trustworthy time server. Date and time are provided by one of the parties, and the other party determines that the date and time value is sufficiently accurate.

When a more highly trusted time source is needed, the Time-Stamp Protocol [TSP] can be used to obtain a time-stamp on the evidence from a trusted third party.

6.2. Storage Considerations

Parties that choose to preserve a plaintext record of Application Data or Handshake Protocol messages for evidence verification at a later time must ensure must ensure that this data is not inappropriately disclosed by employing suitable physical protection or cryptographic mechanisms that are at least as strong as the selected TLS ciphersuite.

Suitable physical controls for the protection of Application Data or Handshake Protocol messages containing keying material or sensitive data should use removable storage media in conjunction with durable, locking storage containers. If the removable media is transferred from one location to another or backup copies are made, secure handling protections ought to be employed, which might include the use of insured or bonded couriers.

A suitable cryptographic mechanism provides confidentiality protection, since the hash value in the evidence itself provides integrity protection. One reasonable solution is to encrypt the Handshake Protocol messages and Application Data messages with a fresh symmetric encryption key using the same algorithm that was negotiated for the selected TLS ciphersuite. The key generation should follow the recommendations in [RANDOM]. Then, the symmetric key is encrypted for storage with the party's RSA public key or long-lived key-encryption key. The Cryptographic Message Syntax (CMS) [CMS] offers a convenient way to keep all of this information together.

An alternative cryptographic mechanism is to save the TLS session itself. The negotiated TLS ciphersuite was already used to protect the Application Data messages, and the Handshake Protocol messages contain the keying material necessary to decrypt them if the party retains the private keys and/or pre-shared secrets.

7. References

7.1. Normative References

[IANA] Narten, T., and H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs", RFC 3434, October 1998.

[DSS] Federal Information Processing Standards Publication (FIPS PUB) 186, Digital Signature Standard, 2000.

[PKCS1] Kaliski, B., "PKCS #1: RSA Encryption Version 1.5", RFC 2313, March 1998.

[PKIX1] Housley, R., Polk, W., Ford, W. and D. Solo, "Internet Public Key Infrastructure - Certificate and Certificate Revocation List (CRL) Profile", RFC 3280, April 2002.

[TLS1.0] Dierks, T., and C. Allen, "The TLS Protocol, Version 1.0", RFC 2246, January 1999.

[TLS1.1] Dierks, T., and E. Rescorla, "The Transport Layer Security (TLS) Protocol, Version 1.1", RFC 4346, April 2006.

[TLSEXT] Blake-Wilson, S., Nystrom, M., Hopwood, D., Mikkelsen, J., and T. Wright, "Transport Layer Security (TLS) Extensions", RFC 4366, April 2006.

[SHA] Federal Information Processing Standards Publication (FIPS PUB) 180-2, Secure Hash Algorithm, 2002.

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

[X9.62] X9.62-1998, "Public Key Cryptography For The Financial Services Industry: The Elliptic Curve Digital Signature Algorithm (ECDSA)", January 7, 1999.

7.2. Informative References

[AUTHZ] Brown, M., and R. Housley, "Transport Layer Security (TLS) Authorization Extensions", work in progress, draft-housley-tls-authz-extns.

[CMS] Housley, R., "Cryptographic Message Syntax (CMS)", RFC 3852, July 2004.

[TSP] Adams, C., Cain, P., Pinkas, D., and R. Zuccherato, "Internet X.509 Public Key Infrastructure: Time-Stamp Protocol (TSP)", RFC 3161,August 2001.

8. Acknowledgements

Thanks to C. Robert Beattie, J.D. and Randy V. Sabett, J.D., CISSP for their observations and comparisons between the Uniform Electronic Transactions Act (1999) prepared by the National Conference of Commissioners on Uniform State Laws versus the American Bar Association's Digital Signature Guidelines (1996), their observations regarding the strengths and weaknesses of these two approaches, and their desire to promote trust and reduce potential for litigation in online transactions. Their pro bono contribution of time and expertise deserves recognition.

This material is based, in part, upon work supported by the United States Navy Space and Naval Warfare Systems Command under Contract No. N00039-06-C-0097.

Appendix A. Evidence Create Algorithms

The following values define the EvidenceCreateSuite identifiers used in the TLS Evidence Extensions.

An EvidenceCreateSuite defines a cipher specification supported by TLS Evidence Extensions. A suite identifier names a public key signature algorithm and an associated one-way hash function. A registration is named as follows:

<SignatureAlgorithm>_WITH_<HashFunction>

These components have the following meaning:

<SignatureAlgorithm>
Specifies the digital signature algorithm and key length associated with the algorithm. It is used to digitally sign the evidence. The "RSA_1024" value indicates the use of the PKCS#1 v1.5 [PKCS1] digital signature algorithm using a 1024-bit public key. The "DSS_1024" value indicates the use of the DSS digital signature algorithm [DSS] with a 1024-bit public key. The "ECDSA_P384" value indicates the use of the ECDSA digital signature algorithm [X9.62] using the P-384 named elliptic curve.

<HashFunction>
Specifies the one-way hash function used as part of the digital signature. The "SHA_1", "SHA_224", "SHA_256", "SHA_384", and "SHA_512" values identify members of the Secure Hash Algorithm family of one-way- hash functions [SHA].

In most cases it will be appropriate to use the same algorithms and certified public keys that were negotiated in the TLS Handshake Protocol. The following additional steps are required in order to employ the digital signature aspects of a TLS CipherSuite to a valid EvidenceCreateSuite:

1) CipherSuites that do not include signature-capable certificates cannot be used as EvidenceCreateSuite.

2) CipherSuites that specify the use of MD5 one-way hash function should not be used as EvidenceCreateSuite.

Of course, any aspect of a CipherSuite that deals with symmetric ciphers and symmetric cipher key lengths is not relevant to the EvidenceCreateSuite.

All public key certificate profiles used in TLS are defined by the IETF PKIX working group in [PKIX1]. When a key usage extension is present, then either the digitalSignature bit or the nonRepudiation bit MUST be set for the public key to be eligible for signing evidence. If both bits are set, then this requirement is satisfied.

The following EvidenceCreateSuite definitions are made at this time. Section 5 specifies the procedures for registering additional EvidenceCreateSuite definitions.

```
EvidenceCreateSuite RSA_1024_WITH_SHA_1       = { 0x00, 0x01 };
EvidenceCreateSuite RSA_1024_WITH_SHA_256     = { 0x00, 0x02 };
EvidenceCreateSuite RSA_2048_WITH_SHA_256     = { 0x00, 0x03 };

EvidenceCreateSuite DSS_1024_WITH_SHA_1       = { 0x00, 0x11 };
EvidenceCreateSuite DSS_2048_WITH_SHA_256     = { 0x00, 0x12 };

EvidenceCreateSuite ECDSA_P256_WITH_SHA_256   = { 0x00, 0x21 };
EvidenceCreateSuite ECDSA_P384_WITH_SHA_384   = { 0x00, 0x22 };
EvidenceCreateSuite ECDSA_P521_WITH_SHA_512   = { 0x00, 0x23 };
```

Author's Address

Mark Brown
RedPhone Security
2019 Palace Avenue
Saint Paul, MN 55105
USA
mark <at> redphonesecurity <dot> com

Russell Housley
Vigil Security, LLC
918 Spring Knoll Drive
Herndon, VA 20170
USA
housley <at> vigilsec <dot> com

Full Copyright Statement