TLS Working Group I. Hajjeh
Internet Draft INEOVATION
M. Badra
LIMOS Laboratory
Intended status: Experimental December 15, 2007
Expires: June 2008

TLS Sign
draft-hajjeh-tls-sign-04.txt

Status of this Memo

This Internet-Draft will expire on June 15, 2007.

Copyright Notice

Abstract

TLS protocol provides authentication and data protection for communication between two entities. However, missing from the protocol is a way to perform non-repudiation service.

This document defines extensions to the TLS protocol to allow it to perform non-repudiation service. It is based on [TLSSIGN] and it provides the client and the server the ability to sign by TLS, handshake and applications data using certificates such as X.509.

Table of Contents

1. Introduction

Actually, TLS is the most deployed security protocol for securing exchanges. It provides end-to-end secure communications between two entities with authentication and data protection. However, what is missing from the protocol is a way to provide the non-repudiation service.

This document describes how the non-repudiation service may be integrated as an optional module in TLS. This is in order to provide both parties with evidence that the transaction has taken place and to offer a clear separation with application design and development.

TLS-Sign's design motivations included:

O TLS is application protocol-independent. Higher-level protocol can operate on top of the TLS protocol transparently.

O TLS is a modular nature protocol. Since TLS is developed in four independent protocols, the approach defined in this document can be added by extending the TLS protocol and with a total reuse of pre-existing TLS infrastructures and implementations.

O Several applications like E-Business require non-repudiation proof of transactions. It is critical in these applications to have the non-repudiation service that generates, distributes, validates and maintains the evidence of an electronic transaction. Since TLS is widely used to secure these applications exchanges, the non-repudiation should be offered by TLS.

O Generic non-repudiation with TLS. TLS Sign provides a generic non-repudiation service that can be easily used with protocols. TLS Sign minimizes both design and implementation of the signature service and that of the designers and implementators who wish to use this module.

1.1. Conventions used in this document

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. TLS Sign overview

TLS Sign is integrated as a higher-level module of the TLS Record protocol. It is optionally used if the two entities agree. This is negotiated by extending Client and Server Hello messages in the same way defined in [TLSEXT].

In order to allow a TLS client to negotiate the TLS Sign, a new extension type should be added to the Extended Client and Server Hellos messages. TLS clients and servers MAY include an extension of type 'signature' in the Extended Client and Server Hellos messages. The 'extension_data' field of this extension contains a 'signature_request' where:

```
   enum {
         pkcs7(0), smime(1), xmldsig(2), (255);
      } ContentFormat;

   struct {
```

```
             ContentFormat content_format;
             SignMethod sign_meth;
             SignType sign_type<2..2^16-1>;
          } SignatureRequest;

      enum {
            ssl_client_auth_cert(0), ssl_client_auth_cert_url(1), (255);
         } SignMethod;
```

```
      uint8 SignType[2];
```

The client initiates the TLS Sign module by sending the ExtendedClientHello including the 'signature' extension. This extension contains:

- the SignType carrying the type of the non repudiation proof. It can have one of these two values:

```
   SignType non_repudiation_with_proof_of_origin      = { 0x00, 0x01 };
   SignType non_repudiation_without_proof_of_origin   = { 0x00, 0x02 };
```

- the ContentFormat carrying the format of signed data. It can be PKCS7 [PKCS7], S/MIME [SMIME] or XMLDSIG [XMLDSIG]

```
            ContentFormat PKCS7   = { 0x00, 0xA1 };
            ContentFormat SMIME   = { 0x00, 0xA2 };
            ContentFormat XMLDSIG = { 0x00, 0xA3 };
```

o if the value of the ContentFormat is PKCS7, then the PKCS7 Content_type is of type signed-data.

o if the value of the ContentFormat is S/MIME, then S/MIME Content_type is of type SignedData

o if the value of the ContentFormat is XMLDSIG, then XMLDSIG signatureMethod algorithms.

- the SignMethod carrying the signature method that is used to sign the application data (e.g. X509 authentication certificate).

```
            SignMethod X509 = { 0x00, 0xB1 };
```

Actually, this document uses the same certificate used in client authentication. Any new signature method MAY be added in future versions (e.g. delegated attributes certificates).

The server MAY reject the connection by sending the error alert "unsupported_extension" [TLSEXT] and closing the connection.

The client and the server MAY use the same certificates used by the Handshake protocol. Several cases are possible:

- If the server has an interest in getting non-repudiation data from the client and that the cipher_suites list sent by the client does not include any cipher_suite with signature ability, the server MUST (upon reception of tls_sign_on_off protocol message not followed by a

certificate with a type equals to ExtendedServerHello.sign_method) close the connection by sending a fatal error.

- If the server has an interest in getting non-repudiation data from the client and that the cipher_suites list sent by the client includes at least a cipher_suite with signature ability, the server SHOULD select a cipher_suite with signature ability and MUST provide a certificate (e.g., RSA) that MAY be used for key exchange. Further, the server MUST request a certificate from the client using the TLS certificate request message (e.g., an RSA or a DSS signature-capable certificate). If the client does not send a certificate during the TLS Handshake, the server MUST close the TLS session by sending a fatal error in the case where the client sends a tls_sign_on_off protocol message not followed by a certificate with a type equals to ExtendedServerHello.sign_method.

- The client or the server MAY use a certificate different to these being used by TLS Handshake. This MAY happen when the server agrees in getting non-repudiation data from the client and that the type of the client certificate used by TLS Handshake and the type selected by the server from the list in ExtendedClientHello.sign_method are different, or when the ExtendedServerHello.cipher_suite does not require client and/or server certificates. In these cases, the client or the server sends a new message called certificate_sign, right after sending the tls_sign_on_off protocol messages. The new message contains the sender's certificate in which the type is the same type selected by the server from the list in ExtendedClientHello.sign_method. The certificate_sign is therefore used to generate signed data. It is defined as follows:

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

   struct {
           ASN.1Cert certificate_list<1..2^24-1>;
        } CertificateSign;
```

The certificate_list, as defined in [TLS], is a sequence (chain) of certificates. The sender's certificate MUST come first in the list. If the server has no interest in getting non-repudiation data from the client, it replays with an ordinary TLS ServerHello or return a handshake failure alert and close the connection [TLS].

```
    Client                                                 Server

    ClientHello                -------->
                                                      ServerHello
                                                     Certificate*
```

```
                                               ServerKeyExchange*
                                              CertificateRequest*
                              <--------           ServerHelloDone
    Certificate*
    ClientKeyExchange
    CertificateVerify*
    ChangeCipherSpec
    Finished                   -------->
                                                 ChangeCipherSpec
                            <--------                    Finished
    TLSSignOnOff   <-------------------------->     TLSSignOnOff
    CertificateSign* <---------------------->  CertificateSign*
    (Signed) Application Data <-----> (Signed) Application Data
```

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

2.1. tls sign on off protocol

To manage the generation of evidence, new sub-protocol is added by this document, called tls_sign_on_off. This protocol consists of a single message that is encrypted and compressed under the established connection state. This message can be sent at any time after the TLS session has been established. Thus, no man in the middle can replay or inject this message. It consists of a single byte of value 1 (tls_sign_on) or 0 (tls_sign_off).

```
enum {
      change_cipher_spec(20), alert(21), handshake(22),
      application_data(23), tls_sign(TBC), (255)
   } ContentType;

struct {
        enum { tls_sign_off(0), tls_sign_on(1), (255) } type;
     } TLSSignOnOff;
```

The tls_sign_on_off message is sent by the client and/or server to notify the receiving party that subsequent records will carry data signed under the negotiated parameters.

Note: TLSSignOnOff is an independent TLS Protocol content type, and is not actually a TLS handshake message.

2.1.1 TLS sign packet format

This document defines a new packet format that encapsulates signed data, the TLSSigntext. The packet format is shown below. The fields are transmitted from left to right.

```
0                   1                   2                   3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| Content-Type  |     Flag      |     Version                   |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|              Length           |  Signed Data ...
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Content-Type

Same as TLSPlaintext.type.

Flag

```
      0 1 2 3 4 5 6 7 8
      +-+-+-+-+-+-+-+-+
      |A R R R R R R R|
      +-+-+-+-+-+-+-+-+
```

A = acknowledgement of receipt
R = Reserved.

When the whole signed data is delivered to the receiver, the TLS Sign will check the signature. If the signature is valid and that the sender requires a proof of receipt, the receiver MUST generate a TLSSigntext packet with the bit A set to 1 (acknowledgement of receipt). This helps the receiver of the acknowledgment of receipt in storing the data-field for later use (see section 2.2). The data field of that message contains the digest of the whole data receiver by the generator of the acknowledgement of receipt. The digest is signed before sending the result to the other side.

2.1.1. bad_sign alert

This alert is returned if a record is received with an incorrect signature. This message is always fatal.

2.2. Storing signed data

The objective of TLS Sign is to provide both parties with evidence that can be stored and later presented to a third party to resolve disputes that arise if and when a communication is repudiated by one

of the entities involved. This document provides the two basic types of non-repudiation service:

- Non-repudiation with proof of origin: provides the TLS server with evidence proving that the TLS client has sent it the signed data at a certain time.

- Non-repudiation with proof of delivery: provides the TLS client with evidence that the server has received the client's signed data at a specific time.

TLS Handshake exchanges the current time and date according to the entities internal clock. Thus, the time and date can be stored with the signed data as a proof of communication. For B2C or B2B transactions, non-repudiation with proof of origin and non-repudiation with proof of receipt are both important. If the TLS client requests a non-repudiation service with proof of receipt, the server SHOULD verify and send back to client a signature on the hash of signed data.

The following figure explains the different events for proving and storing signed data [RFC4949]. RFC 4949 uses the term "critical action" to refer to the act of communication between the two entities. For a complete non-repudiation deployment, 6 phases should be respected:

```
--------   --------   --------   --------   --------   . --------
Phase 1:   Phase 2:   Phase 3:   Phase 4:   Phase 5:   . Phase 6:
Request    Generate   Transfer   Verify     Retain     . Resolve
Service    Evidence   Evidence   Evidence   Evidence   . Dispute
--------   --------   --------   --------   --------   . --------

Service    Critical   Evidence   Evidence   Archive    . Evidence
Request => Action  => Stored  => Is      => Evidence   . Is
Is Made    Occurs     For Later  Tested     In Case    . Verified
           and        Use |          ^      Critical   .     ^
           Evidence       v          |      Action Is  .     |
           Is         +-------------------+ Repudiated .     |
           Generated  |Verifiable Evidence|------> ... . ----+
                      +-------------------+
```

1- Requesting explicit transaction evidence before sending data. Normally, this action is taken by the SSL/TLS client

2- If the server accepts, the client will generate evidence by signing data using his X.509 authentication certificate. Server will go through the same process if the evidence of receipt is requested.

3 - The signed data is then sent by the initiator (client or server) and stored it locally, or by a third party, for a later use if needed.

4 - The entity that receive the evidence process to verify the signed data.

5- The evidence is then stored by the receiver entity for a later use if needed.

6- In this phase, which occurs only if the critical action is repudiated, the evidence is retrieved from storage, presented, and verified to resolve the dispute.

With this method, the stored signed data (or evidence) can be retrieved by both parties, presented and verified if the critical action is repudiated.

3. Security Considerations

Security issues are discussed throughout this memo.

4. IANA Considerations

This document defines a new TLS extension "signature", assigned the value TBD from the TLS ExtensionType registry defined in [TLSEXT].

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

This document defines a new handshake message, certificate_sign, whose value is to be allocated from the TLS HandshakeType registry defined in [TLS].

The bad_sign alert that is defined in this document is assigned to the TLS Alert registry defined in [TLS].

5. References

5.1. Normative References

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

[TLS] Dierks, T. and E. Rescorla, "The TLS Protocol Version 1.1", RFC 4346, April 2005.

[TLSEXT] Blake-Wilson, S., et. al., "Transport Layer Security TLS) Extensions", RFC 4366, April 2006.

[PKCS7] RSA Laboratories, "PKCS #7: RSA Cryptographic Message Syntax Standard," version 1.5, November 1993.

[SMIME] Ramsdell, B., "S/MIME Version 3 Message Specification", RFC 3851, July 2004.

[XMLDSIG] Eastlake, D., et. al, "(Extensible Markup Language) XML Signature Syntax and Processing", RFC 3275, March 2002.

5.2. Informative References

[RFC4949] Shirey, R., "Internet Security Glossary", RFC 4949, August 2007.

[TLSSIGN] Hajjeh, I., Serhrouchni, A., "Integrating a signature module in SSL/TLS, ICETE2004., ACM/IEEE, First International Conference on E-Business and Telecommunication Networks, Portugal, August 2004.

Author's Addresses

Ibrahim Hajjeh
INEOVATION
France

Email: hajjeh@ineovation.com

Mohamad Badra
LIMOS Laboratory - UMR6158, CNRS
France

Email: badra@isima.fr

Appendix Changelog

Changes from -01 to -02:

o Add an IANA section.

o Small clarifications to section 2.

o Add the bad_sign alert and the certificate_sign message.

Changes from -00 to -01:

o Clarifications to the format of the signed data in Section 2.

o Small clarifications to TLS SIGN negotiation in Section 2.

o Added Jacques Demerjian and Mohammed Achemlal as contributors/authors.

Intellectual Property Statement

Disclaimer of Validity

This document and the information contained herein are provided on an "AS IS" basis and THE CONTRIBUTOR, THE ORGANIZATION HE/SHE REPRESENTS OR IS SPONSORED BY (IF ANY), THE INTERNET SOCIETY, THE IETF TRUST AND THE INTERNET ENGINEERING TASK FORCE DISCLAIM ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY WARRANTY THAT THE USE OF THE INFORMATION HEREIN WILL NOT INFRINGE ANY RIGHTS OR ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

Copyright Statement

Acknowledgment

Funding for the RFC Editor function is currently provided by the Internet Society.