INTEGRATING A SIGNATURE MODULE IN SSL/TLS

Ibrahim Hajjeh, Ahmed Serhrouchni
Ecole Nationale Supérieure des Télécommunications - LTCI-UMR 5141 CNRS, France

Keywords: SSL/TLS, e-commerce, non-repudiation, data signature.

Abstract: SSL/TLS[1] is currently the most deployed security protocol on the Internet. SSL/TLS 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. In this paper, we describe a generic implementation of the non-repudiation service as an optional module in the SSL/TLS protocol. This approach provides both parties with evidence that the transaction has taken place and a clear separation with application design and development. We discuss the motivation for our approach and our proposed architecture.

1 INTRODUCTION

SSL stands for secure socket layer (Freier, 2000), was first developed by Netscape Corporation in 1994, and standardized by the Internet Engineering Task Force "IETF" in 1998 as we know the Transport Layer Security protocol or "TLS" (Dierks, 1999). SSL/TLS is designed to make use of TCP to provide a reliable end-to-end secure service with confidentiality, data integrity and authentication for one or both entities.

Today, SSL/TLS is the most deployed security protocol. This is due mainly to its native integration in browsers and web servers. However like other security protocol (IPSEC (Kent, 1998), SSH (Ylonen, 2003), etc.), SSL/TLS does not provide a non-repudiation service. This is left to the application layer. The application itself has to manage the exchange of signed data and its storage (Wichert, 1999).

In this paper we propose a generic implementation of the signature service in SSL/TLS called "SSL-SIGN" that can be used with any application, in a transparent method and with the minimum of programmer effort. Our approach is "generic" because different type of items, such as, data, signature format, or value can be exchanged during the Extended SSL/TLS handshake. To keep interoperability with existing SSL/TLS versions, SSL-SIGN will be negotiated using the Extended Client and Server Hello messages defined in (Blake-Wilson, 2003) and data signature begins at the end of Extended SSL/TLS handshake. In addition, SSL-SIGN requires both entities to be authenticated using their X.509 certificates (ITU-T, 1997). This will guarantee that the signer of messages is always the same as the pretended sender.

The rest of this paper is organized as follows. In section 2 we discuss the motivation behind this work. In section 3 we give a brief background description of the Standard and Extended SSL/TLS protocols. In section 4 we present the signature module. In section 5 we discuss related work. To conclude we propose an analysis of this solution and its prospects, in particular in experimentation and future deployment.

[1] There are some slight differences between SSL 3.0 and TLS 1.0, this paper will refer to the protocol as SSL/TLS

Hajjeh I. and Serhrouchni A. (2004).
INTEGRATING A SIGNATURE MODULE IN SSL/TLS.
In *Proceedings of the First International Conference on E-Business and Telecommunication Networks*, pages 305-310
DOI: 10.5220/0001395903050310

2 MOTIVATION

Today, e-commerce applications have more specific needs such as secure data storage and evidence management. Implementing a non-repudiation service in a security protocol will lead to more interoperability between applications and can provide standard evidence that critical transactions have taken place. The proof is an S/MIME (Dusse, 1998), PKCS7 (Kalishi, 1998) or XML-DSIG (W3C, 2000) signed data based on the proposed IETF Internet standards. The main reasons for integrating a signature module in the SSL/TLS security protocol are as follows:

1. Modularity of TLS protocol
The first motivation of our proposal is the modular nature of SSL/TLS protocol. Since SSL/TLS is developed in four independent protocols, our approach can be added without any change to the SSL/TLS protocol and with a total reuse of pre-existing SSL/TLS infrastructure and implementation. To demonstrate, we implemented SSL-SIGN as a single package that can be optionally used to deliver the non-repudiation service and without any change in other SSL/TLS protocols. Our work was also improved by the standardization of (Blake-Wilson, 2003) that we propose to use to keep interoperability with existing SSL/TLS versions.

2. Generic non-repudiation service
The first objective of SSL-SIGN is to provide a generic non-repudiation service that can be easily used with protocols. SSL-SIGN will minimize both design and implementation of the signature service and that of the designers and implementators who wish to use this module. Thus, we choose to implement the signature module with an SSL-like function (*ssl_sign_write & ssl_sign_read*) with a number of input parameters to simplify its integration with applications.

3. Digital signature of E-business transactions
E-business applications are being more exigent in security needs and there is an insistent demand for an electronic equivalent to the handwritten signature. The technologies needed for this purpose (S/MIME, PKCS7v1.5, CMS (Housley, 2002)) have been available for several years but are totally independent of the security protocols in use like SSL and IPSEC. Integrating a data signature module in a security protocol especially SSL/TLS will lead to more secure transaction between entities and can provide evidence that can be later presented to a third party to resolve any disputes between them.

3 THE SSL/TLS PROTOCOL

This section gives a short introduction into the SSL/TLS protocol. It also explains the specification of the SSL/TLS handshake protocol and the proposed SSL/TLS Extensions. However, a detailed specification of SSL/TLS and SSL/TLS Extensions is outside the scope of this paper. We only introduce the two concepts with enough detail to put the description of our design and architecture in context.

3.1 Background

The standard SSL/TLS Protocol consists of two layers: the TLS Record Protocol and the TLS Handshake Protocol. The TLS Record takes messages to be transmitted from various high level protocols and encapsulates them in a secure connection with data integrity and confidentiality. The TLS Handshake Protocol allows the peer entities located at both ends of the secure channel to authenticate one another, to negotiate encryption algorithms and to exchange secret session keys for encryption. Once this phase in finished, a protected connection is established.

3.2 The SSL/TLS Handshake Protocol

TLS handshake is the most complex part of the SSL/TLS protocol. In the TLS handshake, the two entities will create a shared secret and authenticate each other. However, in most cases, only the server is authenticated.

In a typical full handshake (figure 1), the client will begin the TLS exchange by sending a *ClientHello* message which contains a random number (R1), a session identifier (*S_ID*), a compression list (*compression_list*) and a list of supported cipher suites (*cipher_list*). The server sends the *ServerHello* message that contains a generated random number (R2), a session identifier and the selected cipher suite. The server also sends, his X509 certificate message containing the server's public key. The client verifies the server's public key and generates a 48-byte random number, called the *pre-master-secret*, encrypts it using the server public key and sends it to the server in the *ClientKeyExchange* message. Upon receiving the *ClientKeyExchange* message, the server decrypts the

pre-master secret using its server's private key. At this point, both client and server can calculate a *master-secret* computed from the pre-master-secret and the two exchanged random numbers. This secret will serve after in deriving the symmetric keys used in data encryption and authentication. In the last exchange, the two entities exchange the finished messages that contain a MAC of all exchanged messages.

If the client's certificate is required by the server (the server send the *CertificateRequest* message), then the client sends a C*ertificateVerify* message including its signature on the hash value of the *pre-master* key combined with all past messages exchanged in the current session.

With the present TLS Handshake, the server is not able to know the requested client service till a full handshake has occurred with the client. The main objective of the TLS Extensions is to permit the two entities to negotiate an optional service during the first TLS handshake exchange.

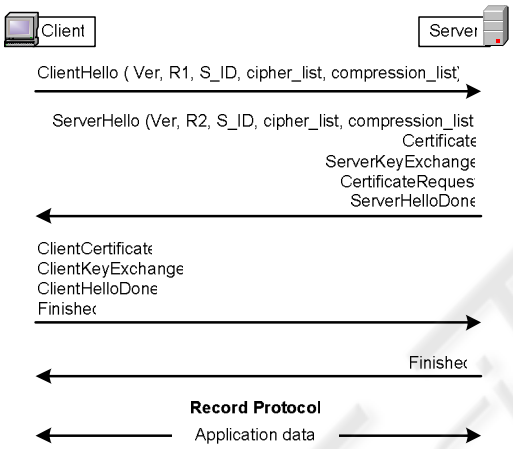

Figure 1: The SSL/TLS Handshake

3.3 The Extended TLS Handshake

TLS Extensions proposes a framework to control the TLS handshake, and extended functions (Blake-Wilson, 2003). It provides both generic extension mechanisms for the TLS handshake client and server hellos, and specific extensions using these generic mechanisms. (Blake-Wilson, 2003) Defines two generic messages called *ExtendedClientHello* and *ExtendedServerHello* messages. These two messages allow the two entities to negotiate new proposed services in the first SSL/TLS exchange and before opening a secured session. This work has also resolved all compatibility problems between TLS clients that want to negotiate our optional signature module and servers that do not support this option, and vice versa.

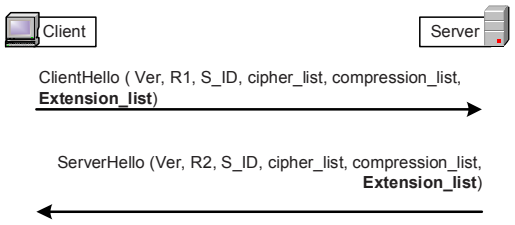

Figure 2: Negotiation of Extended Client and Server Hello messages

In an Extended TLS Handshake (figure 2), the client sends the *ExtendedClientHello* message that contains the standard *ClientHello* message and a proposed extension list (*Extension_list*). If the server accepts the proposed new functionalities, he will respond with the *ExtendedServerHello* message with the same *Extension_list* field sent by the client.

Since the *ClientHello* message defined in TLSv1.0 can contain additional information and to keep interoperability between TLS and the proposed TLS Extensions, the *ExtendedServerHello* message should be sent just in response to an *ExtendedClientHello* message and any error in message format will generate a fatal alert (Blake-Wilson, 2003). The rest of this exchange is similar to the standard TLS handshake.

In SSL-SIGN, a specific extension called signature will be used to negotiate the non-repudiation service between client and server.

4 OUR APPROACH: SSL-SIGN

4.1 Overview

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

The signature module or 'SSL-SIGN' is integrated as a higher-level module of the TLS Record protocol (figure 3). SSL-SIGN will be optionally used if the two entities agree on it during the Extended TLS handshake. When SSL-SIGN is negotiated, the following steps are involved:

1. Exchanging SSL-SIGN initiation messages and negotiates the signature format some other parameters.
2. Authenticating both TLS client and server with their X509 public key certificate

containing the X.509 signature and encryption extensions.

3. Storing all exchanged and signed data after the Extended TLS handshake.

The next sections will detail these steps

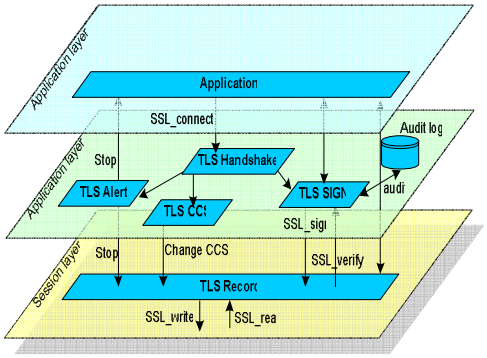

Figure 3: The SSL-SIGN Architecture

4.2 Client authentication

Because SSL-SIGN relies on the security properties of the SSL/TLS session, especially for checking data integrity, protecting against man-in-the-middle attacks and preventing replay attacks, and because the non-repudiation service is viewed as a stronger version of data authentication service, client authentications will be mandatory before allowing then to sign the data. This will also protect the server against Denial of Service (DOS) attacks, as we do not force them to verify signed data coming from anonymous users.

In SSL-SIGN, the server sends the *CertificateRequest* message to the client requesting from him a strong authentication with an X.509 certificate. Client will send his certificate in the Certificate message and sign all exchanged parameters with the SSL/TLS server. This can also be used by the server as a proof of client communication.

4.3 Digital signature and data storage

The objective of SSL-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. SSL-SIGN provides the two basic types of non-repudiation service:

Non-repudiation with proof of origin
And non-repudiation with proof of delivery

The *non-repudiation with proof of origin* provides the TLS server with evidence proving that the TLS client has sent him the signed data at a certain time. The *non-repudiation with proof of delivery* provides the TLS client with evidence that the server has received his signed data at a specific time. Because SSL/TLS add GMT time in its first exchange to protect negotiation against replay attacks and because all handshake TLS are signed by the client certificate, the time value 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. All signed data are enveloped in a new message with a 2-byte

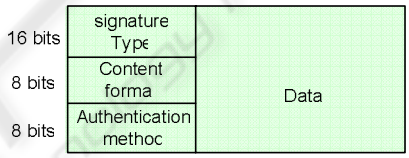

Figure 4: The SSL/TLS signature message

header containing: the signature type (ex. *sign_with_proof_of_receipt*), the content format (ex. *signed_smime_file*) and the authentication method (ex. *x509_tlsclient_cert*).

Figure 5 explains the different events for proving and storing signed data. (Ford, 1994) Uses the term “critical action” to refer to the act of communication between the two entities. For a complete non-repudiation deployment, 5 steps should be respected:

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.

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

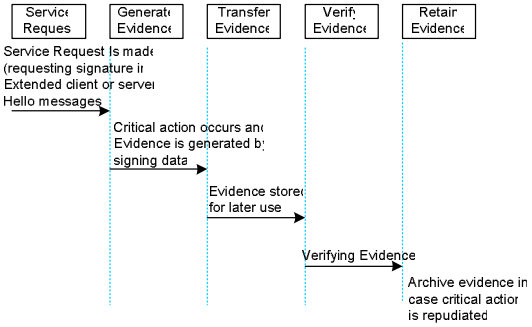

Figure 5: Different events for establishing a complete non-repudiation service between the TLS client and server.

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

4.4 Initialising the SSL-SIGN module

In order to allow a TLS client to negotiate the signature module, a new extension type should be added to the Extended Client Hello and Server Hello messages. TLS clients and servers may include an extension of type 'signature' in the Extended Client Hello and Server Hello messages. The '*extension_data*' field of this extension will contain a '*signature_request*' where:

```
enum {
        pkcs7_1.5(0), smime (2), xmldsig(255);
      }ContentFormat;
struct {
      ContentFormat          content_format;
       AuthMethode             auth_meth;
       Boolean                 bool;
       Signature_type
sign_type<1..2^16-1>;
      } signature_request;
enum {
        x509cert(0), x509cert_url(1), (255);
      } AuthMethode;
enum {
       false(0), true(1);
      } Boolean;
opaque Signature_type<1..2^16-1>;
```

The client initiates the SSL-SIGN module by sending the *ExtendedClientHello* with the 'signature' extension containing the signature type (non-repudiation with proof of origin, etc), the content format (PKCS7, S/MIME, XMLDSIG etc.), a Boolean value set to *true* if the client wants to negotiate the signature and *false* when he wants to stop this service. The client sends his authentication method (in this proposition, the client will use his X509 authentication certificate to sign the exchanged data after the TLS handshake phase, in other scenarios, a client can use a new certificate, an RSA public key or a delegated attribute certificates. The server can reject the connection by sending a fatal alert and closing the connection or accepting the negotiation of this module. In the case where the server accepts the requested service, it should specify a list of cipher suite that supports data signature (using RSA for example) and re-sends the client extension in the *ExtendedServerHello* message. Because the client authentication is compulsory when negotiating SSL-SIGN, the server sends his certificate, the *certificateRequest* and the *ServerHelloDone* messages. The client will then send his certificate and all necessary TLS parameters to finish the TLS handshake negotiation with the server.

4.5 Resuming SSL/TLS Handshake

SSL Resumed session is a fast SSL Handshake defined to minimize the SSL cryptographic operations and a significant number of SSL messages (Kambourakis, 2002). In the SSL Resumed session, the client will begin by sending in its *ClientHello* message a non-null session ID. If the server agrees to the resumed handshake, it will open a secure session based on the old security keys negotiated in a previous full handshake, otherwise, the server will generate a different session ID value and a full handshake will then takes place.

With SSL-SIGN, the SSL Resumed handshake will also be used to manage the non-repudiation service. In a real scenario, a client or an enterprise *A* connects to a secure site using the standard TLS handshake to purchase an order. After submitting all necessary information, it will arrive at a new site where payment should be confirmed and signature is requested. At this point a new connection in opened using the resumed handshake and non-repudiation service is negotiated using the Extended client and server hellos messages named respectively *ClientHelloSIGN* and *ServerHelloSIGN*. If *A* was not authenticated in his first SSL/TLS exchange or does not support a signature algorithm, the server will reject the connection and sends the *HelloRequest* message. *HelloRequest* is a simple notification that the client should begin the negotiation process anew by sending a *ClientHello* message when convenient (Dierks, 1999).

To stop the generation of evidence, client should just negotiate a resumed handshake with

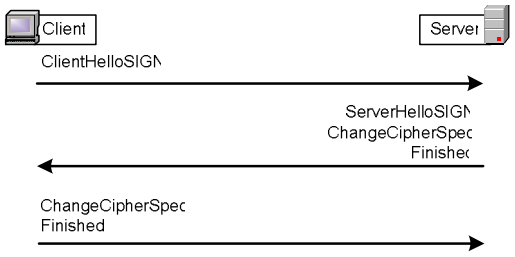

Figure 6: The SSL-SIGN Resumed Handshake client and server.

ClientHelloSIGN message with the value *false* in the Boolean field. After the Extended TLS handshake, all signed data is encapsulated in a new message with a signature header containing the negotiated parameters. This phase begin before data fragmentation by the TLS Record protocol.

5 CONCLUSION AND FUTUR WORK

We have presented the benefit of integrating a signature module in SSL/TLS protocol to protect exchanged data with a generic non-repudiation service. We have implemented a first generic module using the GNUTLS Transport security Library (GNUTLS) that supports the TLS Extensions standard.

There are several advantages of our module as opposed to leaving the non-repudiation service to applications. Our design adds signature before data encryption this should not be considered when seeing the low cost of the symmetric encryption and decryption at the TLS record layer.

There are still some things that remain to be added to our prototype in order to offer more complete functionality:

- Our architecture needs to be integrated in client web browsers like Mozilla web browser.
- The optional *close* of SIGN module is currently not supported.

Finally, our architecture needs to be extended to deal with X.509 attribute certificate. With the SSL-SIGN module, attribute certificate will be used for signature delegating, authentication and access control mechanisms.

REFERENCES

Kambourakis, G., Rouskas, A.N. and Gritzalis, S., (2002). Using SSL/TLS in Authentication and Key Agreement Procedures of Future Mobile Networks. In IEEE MWCN'02, *4th IEEE Int Conf on Mobile and Wireless Communications Networks 2002, Stockholm, Sweden.*

Wichert, M., Ingham, D. et al., (1999). Non-repudiation Evidence Generation for CORBA using XML. In ACSAC'99, *15th Annual Computer Security Applications Conference, Scottsdale, AZ, USA.*

Jackson K., Tuecke S. and al., (2001). TLS Delegation Protocol, In GGF1'01, *First Global Grid Forum & European Datagrid Conference, Amsterdam.*

Kalishi B., (1998). Cryptographic Message Syntax Version 1.5, *[Request for Comments], IETF, No. 2315.*

Dierks, T., (1999). The TLS Protocol Version 1.0, *[Request for Comments], IETF, No. 2246*

Housley R., (2002). Cryptographic Message Syntax (CMS), *[Request for Comments], IETF, No. 3369.*

Kent, S. and Atkinson, R., (1998). Security Architecture for the Internet Protocol, *[Request for Comments], IETF, No. 2401.*

Dusse, S., Hoffman, P. and al., (1998). S/MIMEv2 Message Spec, *[Request for Comments], IETF, No. 2311.*

Housley R., (2002). Cryptographic Message Syntax (CMS), *[Request for Comments], IETF, No. 3369.*

Freier, A., Karlton, P. and Kocker, P., (1996). The SSL Protocol, Version 3.0.

Blake-Wilson S., Nystrom, M. and al., (2003). Transport Layer Security (TLS) Extensions, *[Request for Comments], IETF, No. 3546*

Ford, W. and Baum M., (1994). Secure Electronic Commerce: Building the Infrastructure for Digital Signatures and Encryption, *ISBN 0-13-476342-4.*

GNUTLS project,
Available at: http://www.gnu.org/software/gnutls/

ITU-T Recommendation X.509, (1997). *Information Technology – Open Systems Interconnection – The Directory: Authentication Framework.11*

Ylonen, T. and Moffat, D., (2003). SSH Prot. Arch. *[Draft]"draft-ietf-secsh-architecture-15.txt", IETF.*