INTERNET-DRAFT Brian Tung
draft-tung-transsec-sts-01.txt ISI
expires January 31, 1998

Simple Transaction Security (STS)

0. Status of this Memo

This document is an Internet-Draft. 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."

To learn the current status of any Internet-Draft, please check the "1id-abstracts.txt" listing contained in the Internet-Drafts Shadow Directories on ds.internic.net (US East Coast), nic.nordu.net (Europe), ftp.isi.edu (US West Coast), or munnari.oz.au (Pacific Rim).

The distribution of this memo is unlimited. It is filed as draft-tung-transsec-sts-01.txt, and expires January 31, 1998. Please send comments to the authors.

1. Abstract

This document describes Simple Transaction Security (STS), a protocol for specifying services and protocols used to secure an enclosed message. The framework is flexible enough to allow a wide variety of protocols to be used, at the cost of some negotiation and the optimizations present in protocols such as S-HTTP.

2. Introduction and Motivation

STS provides some of the same services as existing protocols, such as MOSS and S-HTTP. However, STS was developed for use in the LSAM (Large Scale Active Middleware) project, which involves a variety of protocol modifications to enhance web performance, and both MOSS and S-HTTP exhibit limitations that make them less than perfectly suited for this application. In this section, we will describe these protocols and their limitations in some detail.

2.1. MOSS

MIME Object Security Services (MOSS) is a protocol that applies digital signature and encryption services to MIME objects. As such, it is more general than S-HTTP (see below). These services are

provided through the use of end-to-end cryptography between two participants at the application layer.

It requires participants to have at least one public/private key pair, unlike S-HTTP, which allows the services to be specified in relation to symmetric keys (which may be exchanged out of band). In scenarios in which components are in close collaboration, the requirement that keys be exchanged using public key cryptography is an extensive performance liability.

MOSS places no inherent limitations on which services it can provide support for, but currently only the multipart/signed and multipart/encrypted MIME types have been defined. No definitions have been made for simple transport of security components which do not directly affect the content of the message (e.g., payment instruments or authorization tokens).

2.2. S-HTTP

Secure HTTP (S-HTTP) is a message-oriented communications protocol for securing messages sent using HTTP. It allows applications to independently apply security services for transaction confidentiality, data integrity, and non-repudiation. S-HTTP emphasizes flexibility in the choice of algorithms by supporting option negotiation for each transaction.

S-HTTP is designed to operate in a general environment, where components are not necessarily known to support S-HTTP or its associated security services. Much of the negotiation supported in S-HTTP is thus unrequired overhead in cooperative situations in which trusted components have agreed on a security context, and where the transaction need only be secured against external attackers (including spoofers).

S-HTTP also only applies to HTTP transactions. Many of the headers defined in the S-HTTP draft apply specifically to HTTP constructs. It also does not attempt to provide secure transport for security components.

3. Protocol Specification

An STS message consists of a primary, divided into a preamble and a body, and an appendix. The preamble contains parameters relating to the securing of this message. The body contains the payload of the message, and may be encrypted or unencrypted. The appendix contains a signature of the message, though this signature may be computed using a null algorithm. In this section, we describe the syntax for each of these sections.

3.1. Primary

The primary is structured as follows:

```
--BEGIN STS PRIMARY--
Version: 1.0

--BEGIN STS PREAMBLE--
...
--END STS PREAMBLE--

--BEGIN STS BODY--
...
--END STS BODY--

--END STS PRIMARY--
```

3.1.1. Preamble

The header fields in the preamble are formatted as per RFC 822, and are limited to the following:

```
Association-ID:
Certificate:
In-Band-Key:
Encryption-Key:
Auxiliary:
```

The Association-ID field is mandatory and defines a unique identifier for this association. The value of this field is implementation-defined.

The syntax of the Certificate field is as follows:

```
Certificate: Key-ID=<String>; Type=<String>; Encoding=<String>;
        <Certificate>
```

The Key-ID subfield defines the identifier to be used to refer to this key. The Type (e.g., X.509) and Encoding define the format for the certificate.

The syntax of the In-Band-Key field is as follows:

```
In-Band-Key: Key-ID=<String>; Distribution-Algorithm=<String>;
        [Encrypting-Key-ID=<String>;] Encoding=<String>
        <Distributed-Key>
```

The Key-ID subfield defines the identifier to be used to refer to this key. The Distribution-Algorithm defines the algorithm used in distributing this key. If key distribution is based on encryption using a key established elsewhere (possibly in this preamble), then the optional Encrypting-Key-ID subfield contains the key used to encrypt the distributed key. Otherwise, if the distribution is based on a trusted third party, then the Distributed-Key is simply the credentials supplied by that third party (e.g., Kerberos ticket

and authenticator). In either case, the key is encoded using the algorithm defined in the Encoding subfield.

The syntax of the Encryption-Key field is as follows:

```
Encryption-Key: Key-ID=<String>; Encryption-Algorithm=<String>;
        Encoding=<String>
```

The Key-ID subfield identifies the key (if any) to be used to encrypt the body. The Encryption-Algorithm subfield defines the algorithm used to encrypt the body.

The Encoding field defines the encoding mechanism for the body of the message. This field is mandatory, even if encryption is not used.

The Auxiliary field is used for miscellaneous related payload, such as authorization or payment. The syntax for this field will be described later.

3.1.2. Body

The body of the message is constructed as follows. The payload is first optionally encrypted using the parameters described in the Encryption-Key field in the preamble. Then it is encoded using the algorithm identified in the Encoding field in the preamble. Then it is enclosed in the delimiters described above.

3.2. Appendix

The appendix is structured as follows:

```
--BEGIN STS APPENDIX--
Version: 1.0
Association-ID | Key-ID: <Value>
[Algorithm-ID: <Value>]
[Encoding: <Value>]

<Signature>
--END STS APPENDIX--
```

If the signature is to be based on an established association, then the second header is the Association-ID field, and its value is implementation-defined. If, instead, the signature is based on a key distributed in the preamble, then the second header is the Key-ID field, and its value is the identifier assigned to the key in the preamble.

Normally, the Association-ID determines the signature algorithm and encoding; if it does not, then the algorithm and encoding are named in the optional Algorithm-ID and Encoding fields. These two fields are not optional in the case of in-band key distribution.

The Signature begins with the first non-white-space character following the last appendix header value, and ends with the last non-white-space character before the end of the appendix. The structure of the Signature is dependent on the algorithm and encoding used.

4. Response

Normally, the response is sent to the initiator of the transaction in the same way as the request. However, this is not possible if the STS wrapping could not be properly decoded. In this case, the reply indicates the error discovered in decoding the STS request.

4.1. Error Reply

The STS error reply is structured exactly as the request, except that an additional field is used to indicate the error:

    Error-ID:

If applicable, additional information about the error is enclosed in the body of the primary. Currently defined errors are as follows:

    Key(s) not recognized. (List of Key-IDs.)

    Certificate type(s) not recognized. (List of Types.)

    Can't decode certificate(s). (List of Key-IDs.)

    Can't extract key(s). (List of Key-IDs.)

    Distribution algorithm(s) not supported. (List of Algorithms.)

    Encoding(s) not supported. (List of Encodings.)

    Body doesn't decrypt.

    Can't verify signature.

Where shown, the message body contains the listed information.

5. Supported Algorithms and Types

Currently supported certificate formats are X.509.

Currently supported encryption algorithms are DES and RSA.

Currently supported signature algorithms are MD5-DES, MD5-RSA, SHA-DES, and SHA-RSA.

Currently supported key distribution algorithms are KerberosV5.

Currently supported encodings are BASE64, 7-BIT, and 8-BIT.

6. Expiration Date

This draft expires on January 31, 1998.

7. Author

Brian Tung
USC Information Sciences Institute
4676 Admiralty Way Suite 1001
Marina del Rey CA 90292-6695
Phone: +1 310 822 1511
Fax: +1 310 823 6714
E-mail: brian@isi.edu