Network Working Group
Request for Comments: 2660
Category: Experimental

E. Rescorla
RTFM, Inc.
A. Schiffman
Terisa Systems, Inc.

August 1999

The Secure HyperText Transfer Protocol

Status of this Memo

This memo defines an Experimental Protocol for the Internet community. It does not specify an Internet standard of any kind. Discussion and suggestions for improvement are requested. Distribution of this memo is unlimited.

Copyright Notice

Copyright (C) The Internet Society (1999). All Rights Reserved.

Abstract

This memo describes a syntax for securing messages sent using the Hypertext Transfer Protocol (HTTP), which forms the basis for the World Wide Web. Secure HTTP (S-HTTP) provides independently applicable security services for transaction confidentiality, authenticity/integrity and non-repudiability of origin.

The protocol emphasizes maximum flexibility in choice of key management mechanisms, security policies and cryptographic algorithms by supporting option negotiation between parties for each transaction.

Table of Contents

1. Introduction

The World Wide Web (WWW) is a distributed hypermedia system which has gained widespread acceptance among Internet users. Although WWW browsers support other, preexisting Internet application protocols, the native and primary protocol used between WWW clients and servers is the HyperText Transfer Protocol (HTTP) [RFC-2616]. The ease of use of the Web has prompted its widespread employment as a client/server architecture for many applications. Many such applications require the client and server to be able to authenticate each other and exchange sensitive information confidentially. The original HTTP specification had only modest support for the cryptographic mechanisms appropriate for such transactions.

Secure HTTP (S-HTTP) provides secure communication mechanisms between an HTTP client-server pair in order to enable spontaneous commercial transactions for a wide range of applications. Our design intent is to provide a flexible protocol that supports multiple orthogonal operation modes, key management mechanisms, trust models, cryptographic algorithms and encapsulation formats through option negotiation between parties for each transaction.

1.1. Summary of Features

Secure HTTP is a secure message-oriented communications protocol designed for use in conjunction with HTTP. It is designed to coexist with HTTP's messaging model and to be easily integrated with HTTP applications.

Secure HTTP provides a variety of security mechanisms to HTTP clients and servers, providing the security service options appropriate to the wide range of potential end uses possible for the World-Wide Web. The protocol provides symmetric capabilities to both client and server (in that equal treatment is given to both requests and replies, as well as for the preferences of both parties) while preserving the transaction model and implementation characteristics of HTTP.

Several cryptographic message format standards may be incorporated into S-HTTP clients and servers, particularly, but in principle not limited to, [CMS] and [MOSS]. S-HTTP supports interoperation among a variety of implementations, and is compatible with HTTP. S-HTTP aware clients can communicate with S-HTTP oblivious servers and vice-versa, although such transactions obviously would not use S-HTTP security features.

S-HTTP does not require client-side public key certificates (or public keys), as it supports symmetric key-only operation modes.

This is significant because it means that spontaneous private transactions can occur without requiring individual users to have an established public key. While S-HTTP is able to take advantage of ubiquitous certification infrastructures, its deployment does not require it.

S-HTTP supports end-to-end secure transactions, in contrast with the original HTTP authorization mechanisms which require the client to attempt access and be denied before the security mechanism is employed. Clients may be "primed" to initiate a secure transaction (typically using information supplied in message headers); this may be used to support encryption of fill-out forms, for example. With S-HTTP, no sensitive data need ever be sent over the network in the clear.

S-HTTP provides full flexibility of cryptographic algorithms, modes and parameters. Option negotiation is used to allow clients and servers to agree on transaction modes (e.g., should the request be signed or encrypted or both -- similarly for the reply?); cryptographic algorithms (RSA vs. DSA for signing, DES vs. RC2 for encrypting, etc.); and certificate selection (please sign with your "Block-buster Video certificate").

S-HTTP attempts to avoid presuming a particular trust model, although its designers admit to a conscious effort to facilitate multiply-rooted hierarchical trust, and anticipate that principals may have many public key certificates.

S-HTTP differs from Digest-Authentication, described in [RFC-2617] in that it provides support for public key cryptography and consequently digital signature capability, as well as providing confidentiality.

1.2. Changes

This document describes S-HTTP/1.4. It differs from the previous memo in that it differs from the previous memo in its support of the Cryptographic Message Syntax (CMS) [CMS], a successor to PKCS-7; and hence now supports the Diffie-Hellman and the (NIST) Digital Signature Standard cryptosystems. CMS used in RSA mode is bits on the wire compatible with PKCS-7.

1.3. Processing Model

1.3.1. Message Preparation

The creation of an S-HTTP message can be thought of as a a function with three inputs:

1. The cleartext message. This is either an HTTP message or some other data object. Note that since the cleartext message is carried transparently, headers and all, any version of HTTP can be carried within an S-HTTP wrapper.
2. The receiver's cryptographic preferences and keying material. This is either explicitly specified by the receiver or subject to some default set of preferences.
3. The sender's cryptographic preferences and keying material. This input to the function can be thought of as implicit since it exists only in the memory of the sender.

In order to create an S-HTTP message, then, the sender integrates the sender's preferences with the receiver's preferences. The result of this is a list of cryptographic enhancements to be applied and keying material to be used to apply them. This may require some user intervention. For instance, there might be multiple keys available to sign the message. (See Section 3.2.4.9.3 for more on this topic.) Using this data, the sender applies the enhancements to the message clear-text to create the S-HTTP message.

The processing steps required to transform the cleartext message into the S-HTTP message are described in Sections 2 and 3. The processing steps required to merge the sender's and receiver's preferences are described in Sections 3.2.

1.3.2. Message Recovery

The recovery of an S-HTTP message can be thought of as a function of four distinct inputs:

1. The S-HTTP message.
2. The receiver's stated cryptographic preferences and keying material. The receiver has the opportunity to remember what cryptographic preferences it provided in order for this document to be dereferenced.
3. The receiver's current cryptographic preferences and keying material.
4. The sender's previously stated cryptographic options. The sender may have stated that he would perform certain cryptographic operations in this message. (Again, see sections 4 and 5 for details on how to do this.)

In order to recover an S-HTTP message, the receiver needs to read the headers to discover which cryptographic transformations were performed on the message, then remove the transformations using some combination of the sender's and receiver's keying material, while taking note of which enhancements were applied.

The receiver may also choose to verify that the applied enhancements match both the enhancements that the sender said he would apply (input 4 above) and that the receiver requested (input 2 above) as well as the current preferences to see if the S-HTTP message was appropriately transformed. This process may require interaction with the user to verify that the enhancements are acceptable to the user. (See Section 6.4 for more on this topic.)

1.4. Modes of Operation

Message protection may be provided on three orthogonal axes: signature, authentication, and encryption. Any message may be signed, authenticated, encrypted, or any combination of these (including no protection).

Multiple key management mechanisms are supported, including password-style manually shared secrets and public-key key exchange. In particular, provision has been made for prearranged (in an earlier transaction or out of band) symmetric session keys in order to send confidential messages to those who have no public key pair.

Additionally, a challenge-response ("nonce") mechanism is provided to allow parties to assure themselves of transaction freshness.

1.4.1. Signature

If the digital signature enhancement is applied, an appropriate certificate may either be attached to the message (possibly along with a certificate chain) or the sender may expect the recipient to obtain the required certificate (chain) independently.

1.4.2. Key Exchange and Encryption

In support of bulk encryption, S-HTTP defines two key transfer mechanisms, one using public-key enveloped key exchange and another with externally arranged keys.

In the former case, the symmetric-key cryptosystem parameter is passed encrypted under the receiver's public key.

In the latter mode, we encrypt the content using a prearranged session key, with key identification information specified on one of the header lines.

1.4.3. Message Integrity and Sender Authentication

Secure HTTP provides a means to verify message integrity and sender authenticity for a message via the computation of a Message Authentication Code (MAC), computed as a keyed hash over the document using a shared secret -- which could potentially have been arranged in a number of ways, e.g.: manual arrangement or 'inband' key management. This technique requires neither the use of public key cryptography nor encryption.

This mechanism is also useful for cases where it is appropriate to allow parties to identify each other reliably in a transaction without providing (third-party) non-repudiability for the transactions themselves. The provision of this mechanism is motivated by our bias that the action of "signing" a transaction should be explicit and conscious for the user, whereas many authentication needs (i.e., access control) can be met with a lighter-weight mechanism that retains the scalability advantages of public-key cryptography for key exchange.

1.4.4. Freshness

The protocol provides a simple challenge-response mechanism, allowing both parties to insure the freshness of transmissions. Additionally, the integrity protection provided to HTTP headers permits implementations to consider the Date: header allowable in HTTP messages as a freshness indicator, where appropriate (although this requires implementations to make allowances for maximum clock skew between parties, which we choose not to specify).

1.5. Implementation Options

In order to encourage widespread adoption of secure documents for the World-Wide Web in the face of the broad scope of application requirements, variability of user sophistication, and disparate implementation constraints, Secure HTTP deliberately caters to a variety of implementation options. See Section 8 for implementation recommendations and requirements.

2. Message Format

Syntactically, Secure HTTP messages are the same as HTTP, consisting of a request or status line followed by headers and a body. However, the range of headers is different and the bodies are typically

cryptographically enhanced.

2.1. Notational Conventions

This document uses the augmented BNF from HTTP [RFC-2616]. You should refer to that document for a description of the syntax.

2.2. Request Line

In order to differentiate S-HTTP messages from HTTP messages and allow for special processing, the request line should use the special Secure" method and use the protocol designator "Secure-HTTP/1.4". Consequently, Secure-HTTP and HTTP processing can be intermixed on the same TCP port, e.g. port 80. In order to prevent leakage of potentially sensitive information Request-URI should be "*". For example:

```
Secure * Secure-HTTP/1.4
```

When communicating via a proxy, the Request-URI should be consist of the AbsoluteURI. Typically, the rel path section should be replaced by "*" to minimize the information passed to in the clear. (e.g. http://www.terisa.com/*); proxies should remove the appropriate amount of this information to minimize the threat of traffic analysis. See Section 7.2.2.1 for a situation where providing more information is appropriate.

2.3. The Status Line

S-HTTP responses should use the protocol designator "Secure-HTTP/1.4". For example:

```
Secure-HTTP/1.4 200 OK
```

Note that the status in the Secure HTTP response line does not indicate anything about the success or failure of the unwrapped HTTP request. Servers should always use 200 OK provided that the Secure HTTP processing is successful. This prevents analysis of success or failure for any request, which the correct recipient can determine from the encapsulated data. All case variations should be accepted.

2.4. Secure HTTP Header Lines

The header lines described in this section go in the header of a Secure HTTP message. All except ’Content-Type’ and ’Content-Privacy-Domain’ are optional. The message body shall be separated from the header block by two successive CRLFs.

All data and fields in header lines should be treated as case insensitive unless otherwise specified. Linear whitespace [RFC-822] should be used only as a token separator unless otherwise quoted. Long header lines may be line folded in the style of [RFC-822].

This document refers to the header block following the S-HTTP request/response line and preceding the successive CRLFs collectively as "S-HTTP headers".

2.4.1. Content-Privacy-Domain

The two values defined by this document are 'MOSS' and 'CMS'. CMS refers to the privacy enhancement specified in section 2.6.1. MOSS refers to the format defined in [RFC-1847] and [RFC-1848].

2.4.2. Content-Type for CMS

Under normal conditions, the terminal encapsulated content (after all privacy enhancements have been removed) would be an HTTP message. In this case, there shall be a Content-Type line reading:

```
Content-Type: message/http
```

The message/http content type is defined in RFC-2616.

If the inner message is an S-HTTP message, then the content type shall be 'application/s-http'. (See Appendix for the definition of this.)

It is intended that these types be registered with IANA as MIME content types.

The terminal content may be of some other type provided that the type is properly indicated by the use of an appropriate Content-Type header line. In this case, the header fields for the encapsulation of the terminal content apply to the terminal content (the 'final headers'). But in any case, final headers should themselves always be S-HTTP encapsulated, so that the applicable S-HTTP/HTTP headers are never passed unenhanced.

S-HTTP encapsulation of non-HTTP data is a useful mechanism for passing pre-enhanced data (especially presigned data) without requiring that the HTTP headers themselves be pre-enhanced.

2.4.3. Content-Type for MOSS

The Content-Type for MOSS shall be an acceptable MIME content type describing the cryptographic processing applied. (e.g. multipart/signed). The content type of the inner content is described in the content type line corresponding to that inner content, and for HTTP messages shall be 'message/http'.

2.4.4. Prearranged-Key-Info

This header line is intended to convey information about a key which has been arranged outside of the internal cryptographic format. One use of this is to permit in-band communication of session keys for return encryption in the case where one of the parties does not have a key pair. However, this should also be useful in the event that the parties choose to use some other mechanism, for instance, a one-time key list.

This specification defines two methods for exchanging named keys, Inband, Outband. Inband indicates that the session key was exchanged previously, using a Key-Assign header of the corresponding method. Outband arrangements imply that agents have external access to key materials corresponding to a given name, presumably via database access or perhaps supplied immediately by a user from keyboard input. The syntax for the header line is:

```
  Prearranged-Key-Info =
   "Prearranged-Key-Info" ":" Hdr-Cipher "," CoveredDEK "," CoverKey-ID
  CoverKey-ID = method ":" key-name
  CoveredDEK = *HEX
  method = "inband" |  "outband"
```

While chaining ciphers require an Initialization Vector (IV) [FIPS-81] to start off the chaining, that information is not carried by this field. Rather, it should be passed internal to the cryptographic format being used. Likewise, the bulk cipher used is specified in this fashion.

<Hdr-Cipher> should be the name of the block cipher used to encrypt the session key (see section 3.2.4.7)

<CoveredDEK> is the protected Data Encryption Key (a.k.a. transaction key) under which the encapsulated message was encrypted. It should be appropriately (randomly) generated by the sending agent, then encrypted under the cover of the negotiated key (a.k.a. session key) using the indicated header cipher, and then converted into hex.

In order to avoid name collisions, cover key namespaces must be maintained separately by host and port.

Note that some Content-Privacy-Domains, notably likely future revisions of MOSS and CMS may have support for symmetric key management.

The Prearranged-Key-Info field need not be used in such circumstances. Rather, the native syntax is preferred. Keys exchanged with Key-Assign, however, may be used in this situation.

2.4.5. MAC-Info

This header is used to supply a Message Authenticity Check, providing both message authentication and integrity, computed from the message text, the time (optional -- to prevent replay attack), and a shared secret between client and server. The MAC should be computed over the encapsulated content of the S-HTTP message. S-HTTP/1.1 defined that MACs should be computed using the following algorithm ('||' means concatenation):

```
MAC = hex(H(Message||[<time>]||<shared key>))
```

The time should be represented as an unsigned 32 bit quantity representing seconds since 00:00:00 GMT January 1, 1970 (the UNIX epoch), in network byte order. The shared key format is a local matter.

Recent research [VANO95] has demonstrated some weaknesses in this approach, and this memo introduces a new construction, derived from [RFC-2104]. In the name of backwards compatibility, we retain the previous constructions with the same names as before. However, we also introduce a new series of names (See Section 3.2.4.8 for the names) that obey a different (hopefully stronger) construction. (^ means bitwise XOR)

```
HMAC = hex(H(K' ^ pad2 || H(K' ^ pad1 ||[<time>]|| Message)))
pad1 = the byte 0x36 repeated enough times to fill out a
            hash input block. (I.e. 64 times for both MD5 and SHA-1)
pad2 = the byte 0x5c repeated enough times to fill out a
            hash input block.
K' = H(<shared key>)
```

The original HMAC construction is for the use of a key with length equal to the length of the hash output. Although it is considered safe to use a key of a different length (Note that strength cannot be increased past the length of the hash function itself, but can be reduced by using a shorter key.) [KRAW96b] we hash the original key

to permit the use of shared keys (e.g. passphrases) longer than the length of the hash. It is noteworthy (though obvious) that this technique does not increase the strength of short keys.

The format of the MAC-Info line is:

```
MAC-Info =
"MAC-Info" ":"  [hex-time],
hash-alg, hex-hash-data, key-spec
hex-time = <unsigned seconds since Unix epoch represented as HEX>
hash-alg = <hash algorithms from section 3.2.4.8>
hex-hash-data = <computation as described above represented as HEX>
Key-Spec = "null" | "dek" | Key-ID
```

Key-Ids can refer either to keys bound using the Key-Assign header line or those bound in the same fashion as the Outband method described later. The use of a 'Null' key-spec implies that a zero length key was used, and therefore that the MAC merely represents a hash of the message text and (optionally) the time. The special key-spec 'DEK' refers to the Data Exchange Key used to encrypt the following message body (it is an error to use the DEK key-spec in situations where the following message body is unencrypted).

If the time is omitted from the MAC-Info line, it should simply not be included in the hash.

Note that this header line can be used to provide a more advanced equivalent of the original HTTP Basic authentication mode in that the user can be asked to provide a username and password. However, the password remains private and message integrity can be assured. Moreover, this can be accomplished without encryption of any kind.

In addition, MAC-Info permits fast message integrity verification (at the loss of non-repudiability) for messages, provided that the participants share a key (possibly passed using Key-Assign in a previous message).

Note that some Content-Privacy-Domains, notably likely future revisions of MOSS and CMS may have support for symmetric integrity protection The MAC-Info field need not be used in such circumstances. Rather, the native syntax is preferred. Keys exchanged with Key-Assign, however, may be used in this situation.

2.5. Content

The content of the message is largely dependent upon the values of the Content-Privacy-Domain and Content-Transfer-Encoding fields.

For a CMS message, with '8BIT' Content-Transfer-Encoding, the content should simply be the CMS message itself.

If the Content-Privacy-Domain is MOSS, the content should consist of a MOSS Security Multipart as described in RFC1847.

It is expected that once the privacy enhancements have been removed, the resulting (possibly protected) contents will be a normal HTTP request. Alternately, the content may be another Secure-HTTP message, in which case privacy enhancements should be unwrapped until clear content is obtained or privacy enhancements can no longer be removed. (This permits embedding of enhancements, such as sequential Signed and Enveloped enhancements.) Provided that all enhancements can be removed, the final de-enhanced content should be a valid HTTP request (or response) unless otherwise specified by the Content-Type line.

Note that this recursive encapsulation of messages potentially permits security enhancements to be applied (or removed) for the benefit of intermediaries who may be a party to the transaction between a client and server (e.g., a proxy requiring client authentication). How such intermediaries should indicate such processing is described in Section 7.2.1.

2.6. Encapsulation Format Options

2.6.1. Content-Privacy-Domain: CMS

Content-Privacy-Domain 'CMS' follows the form of the CMS standard (see Appendix).

Message protection may proceed on two orthogonal axes: signature and encryption. Any message may be either signed, encrypted, both, or neither. Note that the 'auth' protection mode of S-HTTP is provided independently of CMS coding via the MAC-Info header of section 2.3.6 since CMS does not support a 'KeyDigestedData' type, although it does support a 'DigestedData' type.

2.6.1.1. Signature

This enhancement uses the 'SignedData' type of CMS. When digital signatures are used, an appropriate certificate may either be attached to the message (possibly along with a certificate chain) as specified in CMS or the sender may expect the recipient to obtain its certificate (and/or chain) independently. Note that an explicitly allowed instance of this is a certificate signed with the private component corresponding to the public component being attested to. This shall be referred to as a self-signed certificate. What, if any, weight to give to such a certificate is a purely local matter. In

either case, a purely signed message is precisely CMS compliant.

2.6.1.2. Encryption

2.6.1.2.1. Encryption -- normal, public key

This enhancement is performed precisely as enveloping (using either ' EnvelopedData' types) under CMS. A message encrypted in this fashion, signed or otherwise, is CMS compliant. To have a message which is both signed and encrypted, one simply creates the CMS SignedData production and encapsulates it in EnvelopedData as described in CMS.

2.6.1.2.2. Encryption -- prearranged key

This uses the 'EncryptedData' type of CMS. In this mode, we encrypt the content using a DEK encrypted under cover of a prearranged session key (how this key may be exchanged is discussed later), with key identification information specified on one of the header lines. The IV is in the EncryptedContentInfo type of the EncryptedData element. To have a message which is both signed and encrypted, one simply creates the CMS SignedData production and encapsulates it in EncryptedData as described in CMS.

2.6.2. Content-Privacy-Domain: MOSS

The body of the message should be a MIME compliant message with content type that matches the Content-Type line in the S-HTTP headers. Encrypted messages should use multipart/encrypted. Signed messages should use multipart/signed. However, since multipart/signed does not convey keying material, is is acceptable to use multipart/mixed where the first part is application/mosskey-data and the second part is multipart/mixed in order to convey certificates for use in verifying the signature.

Implementation Note: When both encryption and signature are applied by the same agent, signature should in general be applied before encryption.

2.6.3. Permitted HTTP headers

2.6.3.1. Overview

In general, HTTP [RFC-2616] headers should appear in the inner content (i.e. the message/http) of an S-HTTP message but should not appear in the S-HTTP message wrapper for security reasons. However, certain headers need to be visible to agents which do not have access to the encapsulated data. These headers may appear in the S-HTTP headers as well.

Please note that although brief descriptions of the general purposes of these headers are provided for clarity, the definitive reference is [RFC-2616].

2.6.3.2. Host

The host header specificies the internet host and port number of the resource being requested. This header should be used to disambiguate among multiple potential security contexts within which this message could be interpreted. Note that the unwrapped HTTP message will have it's own Host field (assuming it's an HTTP/1.1 message). If these fields do not match, the server should respond with a 400 status code.

2.6.3.3. Connection

The Connection field has precisely the same semantics in S-HTTP headers as it does in HTTP headers. This permits persistent connections to be used with S-HTTP.

3. Cryptographic Parameters

3.1. Options Headers

As described in Section 1.3.2, every S-HTTP request is (at least conceptually) preconditioned by the negotiation options provided by the potential receiver. The two primary locations for these options are

1. In the headers of an HTTP Request/Response.
2. In the HTML which contains the anchor being dereferenced.

There are two kinds of cryptographic options which may be provided: Negotiation options, as discussed in Section 3.2 convey a potential message recipient's cryptographic preferences. Keying options, as discussed in Section 3.3 provide keying material (or pointers to keying material) which may be of use to the sender when enhancing a message.

Binding cryptographic options to anchors using HTML extensions is the topic of the companion document [SHTML] and will not be treated here.

3.2. Negotiation Options

3.2.1. Negotiation Overview

Both parties are able to express their requirements and preferences regarding what cryptographic enhancements they will permit/require the other party to provide. The appropriate option choices depend on implementation capabilities and the requirements of particular applications.

A negotiation header is a sequence of specifications each conforming to a four-part schema detailing:

Property -- the option being negotiated, such as bulk encryption algorithm.

Value -- the value being discussed for the property, such as DES-CBC

Direction -- the direction which is to be affected, namely: during reception or origination (from the perspective of the originator).

Strength -- strength of preference, namely: required, optional, refused

As an example, the header line:

```
SHTTP-Symmetric-Content-Algorithms: recv-optional=DES-CBC,RC2
```

could be thought to say: "You are free to use DES-CBC or RC2 for bulk encryption for encrypting messages to me."

We define new headers (to be used in the encapsulated HTTP header, not in the S-HTTP header) to permit negotiation of these matters.

3.2.2. Negotiation Option Format

The general format for negotiation options is:

```
Option = Field ":" Key-val ";" *(Key-val)
Key-val = Key "=" Value *("," Value)
Key = Mode"-"Action             ; This is represented as one
                                ; token without whitespace
Mode = "orig" | "recv"
Action = "optional" | "required" | "refused"
```

The <Mode> value indicates whether this <Key-val> refers to what the agent's actions are upon sending privacy enhanced messages as opposed to upon receiving them. For any given mode-action pair, the interpretation to be placed on the enhancements (<Value>s) listed is:

'recv-optional:' The agent will process the enhancement if the other party uses it, but will also gladly process messages without the enhancement.

'recv-required:' The agent will not process messages without this enhancement.

'recv-refused:' The agent will not process messages with this enhancement.

'orig-optional:' When encountering an agent which refuses this enhancement, the agent will not provide it, and when encountering an agent which requires it, this agent will provide it.

'orig-required:' The agent will always generate the enhancement.

'orig-refused:' The agent will never generate the enhancement.

The behavior of agents which discover that they are communicating with an incompatible agent is at the discretion of the agents. It is inappropriate to blindly persist in a behavior that is known to be unacceptable to the other party. Plausible responses include simply terminating the connection, or, in the case of a server response, returning 'Not implemented 501'.

Optional values are considered to be listed in decreasing order of preference. Agents are free to choose any member of the intersection of the optional lists (or none) however.

If any <Key-Val> is left undefined, it should be assumed to be set to the default. Any key which is specified by an agent shall override any appearance of that key in any <Key-Val> in the default for that field.

3.2.3. Parametrization for Variable-length Key Ciphers

For ciphers with variable key lengths, values may be parametrized using the syntax <cipher>'['<length>']'

For example, 'RSA[1024]' represents a 1024 bit key for RSA. Ranges may be represented as

<cipher>'['<bound1>'-'<bound2>']'

For purposes of preferences, this notation should be treated as if it read (assuming x and y are integers)

<cipher>[x], <cipher>[x+1],...<cipher>[y] (if x<y)

and

<cipher>[x], <cipher>[x-1],...<cipher>[y] (if x>y)

The special value 'inf' may be used to denote infinite length.

Using simply <cipher> for such a cipher shall be read as the maximum range possible with the given cipher.

3.2.4. Negotiation Syntax

3.2.4.1. SHTTP-Privacy-Domains

This header refers to the Content-Privacy-Domain type of section 2.3.1. Acceptable values are as listed there. For instance,

```
SHTTP-Privacy-Domains: orig-required=cms;
                       recv-optional=cms,MOSS
```

would indicate that the agent always generates CMS compliant messages, but can read CMS or MOSS (or, unenhanced messages).

3.2.4.2. SHTTP-Certificate-Types

This indicates what sort of Public Key certificates the agent will accept. Currently defined values are 'X.509' and 'X.509v3'.

3.2.4.3. SHTTP-Key-Exchange-Algorithms

This header indicates which algorithms may be used for key exchange. Defined values are 'DH', 'RSA', 'Outband' and 'Inband'. DH refers to Diffie-Hellman X9.42 style enveloping. [DH] RSA refers to RSA enveloping. Outband refers to some sort of external key agreement.

Inband refers to section 3.3.3.1.

The expected common configuration of clients having no certificates and servers having certificates would look like this (in a message sent by the server):

```
SHTTP-Key-Exchange-Algorithms: orig-optional=Inband, DH;
                               recv-required=DH
```

3.2.4.4. SHTTP-Signature-Algorithms

This header indicates what Digital Signature algorithms may be used. Defined values are 'RSA' [PKCS-1] and 'NIST-DSS' [FIPS-186] Since NIST-DSS and RSA use variable length moduli the parametrization syntax of section 3.2.3 should be used. Note that a key length specification may interact with the acceptability of a given certificate, since keys (and their lengths) are specified in public-key certificates.

3.2.4.5. SHTTP-Message-Digest-Algorithms

This indicates what message digest algorithms may be used. Previously defined values are 'RSA-MD2' [RFC-1319], 'RSA-MD5' [RFC-1321], 'NIST-SHS' [FIPS-180].

3.2.4.6. SHTTP-Symmetric-Content-Algorithms

This header specifies the symmetric-key bulk cipher used to encrypt message content. Defined values are:

```
DES-CBC -- DES in Cipher Block Chaining (CBC) mode [FIPS-81]
DES-EDE-CBC -- 2 Key 3DES using Encrypt-Decrypt-Encrypt in outer
               CBC mode
DES-EDE3-CBC -- 3 Key 3DES using Encrypt-Decrypt-Encrypt in outer
                CBC mode
DESX-CBC -- RSA's DESX in CBC mode
IDEA-CBC -- IDEA in CBC mode
RC2-CBC -- RSA's RC2 in CBC mode
CDMF-CBC -- IBM's CDMF (weakened key DES) [JOHN93] in CBC mode
```

Since RC2 keys are variable length, the syntax of section 3.2.3 should be used.

3.2.4.7. SHTTP-Symmetric-Header-Algorithms

This header specifies the symmetric-key cipher used to encrypt message headers.

DES-ECB -- DES in Electronic Codebook (ECB) mode [FIPS-81]
DES-EDE-ECB -- 2 Key 3DES using Encrypt-Decrypt-Encrypt in ECB mode
DES-EDE3-ECB -- 3 Key 3DES using Encrypt-Decrypt-Encrypt in ECB mode
DESX-ECB -- RSA's DESX in ECB mode
IDEA-ECB -- IDEA
RC2-ECB -- RSA's RC2 in ECB mode
CDMF-ECB -- IBM's CDMF in ECB mode

Since RC2 is variable length, the syntax of section 3.2.3 should be used.

3.2.4.8. SHTTP-MAC-Algorithms

This header indicates what algorithms are acceptable for use in providing a symmetric key MAC. 'RSA-MD2', 'RSA-MD5' and 'NIST-SHS' persist from S-HTTP/1.1 using the old MAC construction. The tokens 'RSA-MD2-HMAC', 'RSA-MD5-HMAC' and 'NIST-SHS-HMAC' indicate the new HMAC construction of 2.3.6 with the MD2, MD5, and SHA-1 algorithms respectively.

3.2.4.9. SHTTP-Privacy-Enhancements

This header indicates security enhancements to apply. Possible values are 'sign', 'encrypt' and 'auth' indicating whether messages are signed, encrypted, or authenticated (i.e., provided with a MAC), respectively.

3.2.4.10. Your-Key-Pattern

This is a generalized pattern match syntax to describe identifiers for a large number of types of keying material. The general syntax is:

```
Your-Key-Pattern =
        "Your-Key-Pattern" ":" key-use "," pattern-info
key-use = "cover-key" | "auth-key" | "signing-key"
```

3.2.4.10.1. Cover Key Patterns

This header specifies desired values for key names used for encryption of transaction keys using the Prearranged-Key-Info syntax of section 2.3.5. The pattern-info syntax consists of a series of comma separated regular expressions. Commas should be escaped with backslashes if they appear in the regexps. The first pattern should be assumed to be the most preferred.

3.2.4.10.2. Auth key patterns

Auth-key patterns specify name forms desired for use for MAC authenticators. The pattern-info syntax consists of a series of comma separated regular expressions. Commas should be escaped with backslashes if they appear in the regexps. The first pattern should be assumed to be the most preferred.

3.2.4.10.3. Signing Key Pattern

This parameter describes a pattern or patterns for what keys are acceptable for signing for the digital signature enhancement. The pattern-info syntax for signing-key is:

```
pattern-info = name-domain "," pattern-data
```

The only currently defined name-domain is 'DN-1779'. This parameter specifies desired values for fields of Distinguished Names. DNs are considered to be represented as specified in RFC1779, the order of fields and whitespace between fields is not significant.

All RFC1779 values should use ',' as a separator rather than ';', since ';' is used as a statement separator in S-HTTP.

Pattern-data is a modified RFC1779 string, with regular expressions permitted as field values. Pattern match is performed field-wise, unspecified fields match any value (and therefore leaving the DN-Pattern entirely unspecified allows for any DN). Certificate chains may be matched as well (to allow for certificates without name subordination). DN chains are considered to be ordered left-to-right with the issuer of a given certificate on its immediate right, although issuers need not be specified. A trailing '.' indicates that the sequence of DNs is absolute. I.e. that the one furthest to the right is a root.

The syntax for the pattern values is,

```
Value = DN-spec *("," Dn-spec)["."]
Dn-spec = "/" *(Field-spec) "/"
Field-spec := Attr = "Pattern"
Attr = "CN" | "L" | "ST" | "O" |
           "OU" | "C" | <or as appropriate>
Pattern = <POSIX 1003.2 regular expressions>
```

For example, to request that the other agent sign with a key certified by the RSA Persona CA (which uses name subordination) one could use the expression below. Note the use of RFC1779 quoting to protect the comma (an RFC1779 field separator) and the POSIX 1003.2 quoting to protect the dot (a regular expression metacharacter).

```
   Your-Key-Pattern: signing-key, DN-1779,
               /OU=Persona Certificate, O="RSA Data Security,
Inc\."/
```

3.2.4.11. Example

A representative header block for a server follows.

```
SHTTP-Privacy-Domains: recv-optional=MOSS, CMS;
      orig-required=CMS
SHTTP-Certificate-Types: recv-optional=X.509;
      orig-required=X.509
SHTTP-Key-Exchange-Algorithms: recv-required=DH;
      orig-optional=Inband,DH
SHTTP-Signature-Algorithms: orig-required=NIST-DSS;
      recv-required=NIST-DSS
SHTTP-Privacy-Enhancements: orig-required=sign;
      orig-optional=encrypt
```

3.2.4.12. Defaults

Explicit negotiation parameters take precedence over default values. For a given negotiation option type, defaults for a given mode-action pair (such as 'orig-required') are implicitly merged unless explicitly overridden.

The default values (these may be negotiated downward or upward) are:

```
SHTTP-Privacy-Domains: orig-optional=CMS;
                       recv-optional=CMS
SHTTP-Certificate-Types: orig-optional=X.509;
                         recv-optional=X.509
SHTTP-Key-Exchange-Algorithms: orig-optional=DH,Inband,Outband;
```

```
                                    recv-optional=DH,Inband,Outband
        SHTTP-Signature-Algorithms: orig-optional=NIST-DSS;
                                   recv-optional=NIST-DSS
        SHTTP-Message-Digest-Algorithms: orig-optional=RSA-MD5;
                                     recv-optional=RSA-MD5
        SHTTP-Symmetric-Content-Algorithms: orig-optional=DES-CBC;
                                      recv-optional=DES-CBC
        SHTTP-Symmetric-Header-Algorithms: orig-optional=DES-ECB;
                                     recv-optional=DES-ECB
        SHTTP-Privacy-Enhancements: orig-optional=sign,encrypt, auth;
                                      recv-required=encrypt;
                                      recv-optional=sign, auth
```

3.3. Non-Negotiation Headers

There are a number of options that are used to communicate or identify the potential recipient's keying material.

3.3.1. Encryption-Identity

This header identifies a potential principal for whom the message described by these options could be encrypted; Note that this explicitly permits return encryption under (say) public key without the other agent signing first (or under a different key than that of the signature). The syntax of the Encryption-Identity line is:

```
Encryption-Identity =
        "Encryption Identity" ":" name-class,key-sel,name-arg
name-class = "DN-1779" | MOSS name forms
```

The name-class is an ASCII string representing the domain within which the name is to be interpreted, in the spirit of MOSS. In addition to the MOSS name forms of RFC1848, we add the DN-1779 name form to represent a more convenient form of distinguished name.

3.3.1.1. DN-1779 Name Class

The argument is an RFC-1779 encoded DN.

3.3.2. Certificate-Info

In order to permit public key operations on DNs specified by Encryption-Identity headers without explicit certificate fetches by the receiver, the sender may include certification information in the Certificate-Info option. The format of this option is:

```
Certificate-Info: <Cert-Fmt>',<Cert-Group>
```

<Cert-Fmt> should be the type of <Cert-Group> being presented.

Defined values are 'PEM' and 'CMS'. CMS certificate groups are provided as a base-64 encoded CMS SignedData message containing sequences of certificates with or without the SignerInfo field. A PEM format certificate group is a list of comma-separated base64-encoded PEM certificates.

Multiple Certificate-Info lines may be defined.

3.3.3. Key-Assign

This option serves to indicate that the agent wishes to bind a key to a symbolic name for (presumably) later reference.

The general syntax of the key-assign header is:

```
Key-Assign =
        "Key-Assign" ":" Method "," Key-Name ","
        Lifetime "," Ciphers ";" Method-args

Key-name = string
Lifetime = "this" | "reply" | ""
Method ="inband"
Ciphers = "null" | Cipher+
Cipher" = <Header cipher from section 3.2.4.7>
kv = "4" | "5"
```

Key-Name is the symbolic name to which this key is to be bound. Ciphers is a list of ciphers for which this key is potentially applicable (see the list of header ciphers in section 3.2.4.7). The keyword 'null' should be used to indicate that it is inappropriate for use with ANY cipher. This is potentially useful for exchanging keys for MAC computation.

Lifetime is a representation of the longest period of time during which the recipient of this message can expect the sender to accept that key. 'this' indicates that it is likely to be valid only for reading this transmission. 'reply' indicates that it is useful for a reply to this message. If a Key-Assign with the reply lifetime appears in a CRYPTOPTS block, it indicates that it is good for at least one (but perhaps only one) dereference of this anchor. An unspecified lifetime implies that this key may be reused for an indefinite number of transactions.

Method should be one of a number of key exchange methods. The only currently defined value is 'inband' referring to Inband keys (i.e., direct assignment).

This header line may appear either in an unencapsulated header or in an encapsulated message, though when an uncovered key is being directly assigned, it may only appear in an encrypted encapsulated content. Assigning to a key that already exists causes that key to be overwritten.

Keys defined by this header are referred to elsewhere in this specification as Key-IDs, which have the syntax:

```
Key-ID = method ":" key-name
```

3.3.3.1. Inband Key Assignment

This refers to the direct assignment of an uncovered key to a symbolic name. Method-args should be just the desired session key encoded in hexidecimal as in:

```
Key-Assign: inband,akey,reply,DES-ECB;0123456789abcdef
```

Short keys should be derived from long keys by reading bits from left to right.

Note that inband key assignment is especially important in order to permit confidential spontaneous communication between agents where one (but not both) of the agents have key pairs. However, this mechanism is also useful to permit key changes without public key computations. The key information is carried in this header line must be in the inner secured HTTP request, therefore use in unencrypted messages is not permitted.

3.3.4. Nonces

Nonces are opaque, transient, session-oriented identifiers which may be used to provide demonstrations of freshness. Nonce values are a local matter, although they are might well be simply random numbers generated by the originator. The value is supplied simply to be returned by the recipient.

3.3.4.1. Nonce

This header is used by an originator to specify what value is to be returned in the reply. The field may be any value. Multiple nonces may be supplied, each to be echoed independently.

The Nonce should be returned in a Nonce-Echo header line. See section 4.1.1.

3.4. Grouping Headers With SHTTP-Cryptopts

In order for servers to bind a group of headers to an HTML anchor, it is possible to combine a number of headers on a single S-HTTP Cryptopts header line. The names of the anchors to which these headers apply is indicated with a 'scope' parameter.

3.4.1. SHTTP-Cryptopts

This option provides a set of cryptopts and a list of references to which it applies. (For HTML, these references would be named using the NAME tag). The names are provided in the scope attribute as a comma separated list and separated from the next header line by a semicolon. The format for the SHTTP-Cryptopts line is:

```
SHTTP-Cryptopts =
                  "SHTTP-Cryptopts" ":" scope ";" cryptopt-list
scope = "scope="<tag-spec>    ; This is all one token without whitespace
tag-spec = tag *("," tag) | ""
cryptopt-list = cryptopt *(";" cryptopt)
cryptopt = <S-HTTP cryptopt lines described below>
tag = <value used in HTML anchor NAME attribute>
```

For example:

```
SHTTP-Cryptopts: scope=tag1,tag2;
                  SHTTP-Privacy-Domains:
                  orig-required=cms; recv-optional=cms,MOSS
```

If a message contains both S-HTTP negotiation headers and headers grouped on SHTTP-Cryptopts line(s), the other headers shall be taken to apply to all anchors not bound on the SHTTP-Cryptopts line(s). Note that this is an all-or-nothing proposition. That is, if a SHTTP-Cryptopts header binds options to a reference, then none of these global options apply, even if some of the options headers do not appear in the bound options. Rather, the S-HTTP defaults found in Section 3.2.4.11 apply.

4. New Header Lines for HTTP

Two non-negotiation header lines for HTTP are defined here.

4.1. Security-Scheme

All S-HTTP compliant agents must generate the Security-Scheme header in the headers of all HTTP messages they generate. This header permits other agents to detect that they are communicating with an S-HTTP compliant agent and generate the appropriate cryptographic

options headers.

For implementations compliant with this specification, the value must be 'S-HTTP/1.4'.

4.1.1. Nonce-Echo

The header is used to return the value provided in a previously received Nonce: field. This has to go in the encapsulated headers so that it an be cryptographically protected.

5. (Retriable) Server Status Error Reports

We describe here the special processing appropriate for client retries in the face of servers returning an error status.

5.1. Retry for Option (Re)Negotiation

A server may respond to a client request with an error code that indicates that the request has not completely failed but rather that the client may possibly achieve satisfaction through another request. HTTP already has this concept with the 3XX redirection codes.

In the case of S-HTTP, it is conceivable (and indeed likely) that the server expects the client to retry his request using another set of cryptographic options. E.g., the document which contains the anchor that the client is dereferencing is old and did not require digital signature for the request in question, but the server now has a policy requiring signature for dereferencing this URL. These options should be carried in the header of the encapsulated HTTP message, precisely as client options are carried.

The general idea is that the client will perform the retry in the manner indicated by the combination of the original request and the precise nature of the error and the cryptographic enhancements depending on the options carried in the server response.

The guiding principle in client response to these errors should be to provide the user with the same sort of informed choice with regard to dereference of these anchors as with normal anchor dereference. For instance, in the case above, it would be inappropriate for the client to sign the request without requesting permission for the action.

5.2. Specific Retry Behavior

5.2.1. Unauthorized 401, PaymentRequired 402

The HTTP errors 'Unauthorized 401', 'PaymentRequired 402' represent failures of HTTP style authentication and payment schemes. While S-HTTP has no explicit support for these mechanisms, they can be performed under S-HTTP while taking advantage of the privacy services offered by S-HTTP. (There are other errors for S-HTTP specific authentication errors.)

5.2.2. 420 SecurityRetry

This server status reply is provided so that the server may inform the client that although the current request is rejected, a retried request with different cryptographic enhancements is worth attempting. This header shall also be used in the case where an HTTP request has been made but an S-HTTP request should have been made. Obviously, this serves no useful purpose other than signalling an error if the original request should have been encrypted, but in other situations (e.g. access control) may be useful.

5.2.2.1. SecurityRetries for S-HTTP Requests

In the case of a request that was made as an SHTTP request, it indicates that for some reason the cryptographic enhancements applied to the request were unsatisfactory and that the request should be repeated with the options found in the response header. Note that this can be used as a way to force a new public key negotiation if the session key in use has expired or to supply a unique nonce for the purposes of ensuring request freshness.

5.2.2.2. SecurityRetries for HTTP Requests

If the 420 code is returned in response to an HTTP request, it indicates that the request should be retried using S-HTTP and the cryptographic options indicated in the response header.

5.2.3. 421 BogusHeader

This error code indicates that something about the S-HTTP request was bad. The error code is to be followed by an appropriate explanation, e.g.:

```
421 BogusHeader Content-Privacy-Domain must be specified
```

5.2.4. 422 SHTTP Proxy Authentication Required

This response is analagous to the 420 response except that the options in the message refer to enhancements that the client must perform in order to satisfy the proxy.

5.2.5. 320 SHTTP Not Modifed

This response code is specifically for use with proxy-server interaction where the proxy has placed the If-Modified-Since header in the S-HTTP headers of its request. This response indicates that the following S-HTTP message contains sufficient keying material for the proxy to forward the cached document for the new requestor.

In general, this takes the form of an S-HTTP message where the actual enhanced content is missing, but all the headers and keying material are retained. (I.e. the optional content section of the CMS message has been removed.) So, if the original response was encrypted, the response contains the original DEK re-covered for the new recipient. (Notice that the server performs the same processing as it would have in the server side caching case of 7.1 except that the message body is elided.)

5.2.6. Redirection 3XX

These headers are again internal to HTTP, but may contain S-HTTP negotiation options of significance to S-HTTP. The request should be redirected in the sense of HTTP, with appropriate cryptographic precautions being observed.

5.3. Limitations On Automatic Retries

Permitting automatic client retry in response to this sort of server response permits several forms of attack. Consider for the moment the simple credit card case:

> The user views a document which requires his credit card. The user verifies that the DN of the intended recipient is acceptable and that the request will be encrypted and dereferences the anchor. The attacker intercepts the server’s reply and responds with a message encrypted under the client’s public key containing the Moved 301 header. If the client were to automatically perform this redirect it would allow compromise of the user’s credit card.

5.3.1. Automatic Encryption Retry

This shows one possible danger of automatic retries -- potential compromise of encrypted information. While it is impossible to consider all possible cases, clients should never automatically reencrypt data unless the server requesting the retry proves that he already has the data. So, situations in which it would be acceptable to reencrypt would be if:

1. The retry response was returned encrypted under an inband key freshly generated for the original request.
2. The retry response was signed by the intended recipient of the original request.
3. The original request used an outband key and the response is encrypted under that key.

This is not an exhaustive list, however the browser author would be well advised to consider carefully before implementing automatic reencryption in other cases. Note that an appropriate behavior in cases where automatic reencryption is not appropriate is to query the user for permission.

5.3.2. Automatic Signature Retry

Since we discourage automatic (without user confirmation) signing in even the usual case, and given the dangers described above, it is prohibited to automatically retry signature enchancement.

5.3.3. Automatic MAC Authentication Retry

Assuming that all the other conditions are followed, it is permissible to automatically retry MAC authentication.

6. Other Issues

6.1. Compatibility of Servers with Old Clients

Servers which receive requests in the clear which should be secured should return ’SecurityRetry 420’ with header lines set to indicate the required privacy enhancements.

6.2. URL Protocol Type

We define a new URL protocol designator, ’shttp’. Use of this designator as part of an anchor URL implies that the target server is S-HTTP capable, and that a dereference of this URL should undergo S-HTTP processing.

Note that S-HTTP oblivious agents should not be willing to dereference a URL with an unknown protocol specifier, and hence sensitive data will not be accidentally sent in the clear by users of non-secure clients.

6.3. Browser Presentation

6.3.1. Transaction Security Status

While preparing a secure message, the browser should provide a visual indication of the security of the transaction, as well as an indication of the party who will be able to read the message. While reading a signed and/or enveloped message, the browser should indicate this and (if applicable) the identity of the signer. Self-signed certificates should be clearly differentiated from those validated by a certification hierarchy.

6.3.2. Failure Reporting

Failure to authenticate or decrypt an S-HTTP message should be presented differently from a failure to retrieve the document. Compliant clients may at their option display unverifiable documents but must clearly indicate that they were unverifiable in a way clearly distinct from the manner in which they display documents which possessed no digital signatures or documents with verifiable signatures.

6.3.3. Certificate Management

Clients shall provide a method for determining that HTTP requests are to be signed and for determining which (assuming there are many) certificate is to be used for signature. It is suggested that users be presented with some sort of selection list from which they may choose a default. No signing should be performed without some sort of explicit user interface action, though such action may take the form of a persistent setting via a user preferences mechanism (although this is discouraged.)

6.3.4. Anchor Dereference

Clients shall provide a method to display the DN and certificate chain associated with a given anchor to be dereferenced so that users may determine for whom their data is being encrypted. This should be distinct from the method for displaying who has signed the document containing the anchor since these are orthogonal pieces of encryption information.

7. Implementation Notes

7.1. Preenhanced Data

While S-HTTP has always supported preenhanced documents, in previous versions it was never made clear how to actually implement them. This section describes two methods for doing so: preenhancing the HTTP request/response and preenhancing the underlying data.

7.1.1. Motivation

The two primary motivations for preenhanced documents are security and performance. These advantages primarily accrue to signing but may also under special circumstances apply to confidentiality or repudiable (MAC-based) authentication.

Consider the case of a server which repeatedly serves the same content to multiple clients. One such example would be a server which serves catalogs or price lists. Clearly, customers would like to be able to verify that these are actual prices. However, since the prices are typically the same to all comers, confidentiality is not an issue. (Note: see Section 7.1.5 below for how to deal with this case as well).

Consequently, the server might wish to sign the document once and simply send the cached signed document out when a client makes a new request, avoiding the overhead of a private key operation each time. Note that conceivably, the signed document might have been generated by a third party and placed in the server’s cache. The server might not even have the signing key! This illustrates the security benefit of presigning: Untrusted servers can serve authenticated data without risk even if the server is compromised.

7.1.2. Presigned Requests/Responses

The obvious implementation is simply to take a single request/response, cache it, and send it out in situations where a new message would otherwise be generated.

7.1.3. Presigned Documents

It is also possible using S-HTTP to sign the underlying data and send it as an S-HTTP messsage. In order to do this, one would take the signed document (a CMS or MOSS message) and attach both S-HTTP headers (e.g. the S-HTTP request/response line, the Content-Privacy-Domain) and the necessary HTTP headers (including a Content-Type that reflects the inner content).

```
SECURE * Secure-HTTP/1.4
Content-Type: text/html
Content-Privacy-Domain: CMS

Random signed message here...
```

This message itself cannot be sent, but needs to be recursively encapsulated, as described in the next section.

7.1.4. Recursive Encapsulation

As required by Section 7.3, the result above needs to be itself encapsulated to protect the HTTP headers. the obvious case [and the one illustrated here] is when confidentiality is required, but the auth enhancement or even the null transform might be applied instead. That is, the message shown above can be used as the inner content of a new S-HTTP message, like so:

```
SECURE * Secure-HTTP/1.4
Content-Type: application/s-http
Content-Privacy-Domain: CMS

Encrypted version of the message above...
```

To unfold this, the receiver would decode the outer S-HTTP message, reenter the (S-)HTTP parsing loop to process the new message, see that that too was S-HTTP, decode that, and recover the inner content.

Note that this approach can also be used to provide freshness of server activity (though not of the document itself) while still providing nonrepudiation of the document data if a NONCE is included in the request.

7.1.5. Preencrypted Messages

Although preenhancement works best with signature, it can also be used with encryption under certain conditions. Consider the situation where the same confidential document is to be sent out repeatedly. The time spent to encrypt can be saved by caching the ciphertext and simply generating a new key exchange block for each recipient. [Note that this is logically equivalent to a multi- recipient message as defined in both MOSS and CMS and so care must be taken to use proper PKCS-1 padding if RSA is being used since otherwise, one may be open to a low encryption exponent attack [HAST96].

7.2. Proxy Interaction

The use of S-HTTP presents implementation issues to the use of HTTP proxies. While simply having the proxy blindly forward responses is straightforward, it would be preferable if S-HTTP aware proxies were still able to cache responses in at least some circumstances. In addition, S-HTTP services should be usable to protect client-proxy authentication. This section describes how to achieve those goals using the mechanisms described above.

7.2.1. Client-Proxy Authentication

When an S-HTTP aware proxy receives a request (HTTP or S-HTTP) that (by whatever access control rules it uses) it requires to be S-HTTP authenticated (and if it isn't already so), it should return the 422 response code (5.7.4).

When the client receives the 422 response code, it should read the cryptographic options that the proxy sent and determine whether or not it is willing to apply that enhancement to the message. If the client is willing to meet these requirements, it should recursively encapsulate the request it previously sent using the appropriate options. (Note that since the enhancement is recursively applied, even clients which are unwilling to send requests to servers in the clear may be willing to send the already encrypted message to the proxy without further encryption.) (See Section 7.1 for another example of a recursively encapsulated message)

When the proxy receives such a message, it should strip the outer encapsulation to recover the message which should be sent to the server.

8. Implementation Recommendations and Requirements

All S-HTTP agents must support the MD5 message digest and MAC authentication. As of S-HTTP/1.4, all agents must also support the RSA-MD5-HMAC construction.

All S-HTTP agents must support Outband, Inband, and DH key exchange.

All agents must support encryption using DES-CBC.

Agents must support signature generation and verification using NIST-DSS.

9. Protocol Syntax Summary

We present below a summary of the main syntactic features of S-HTTP/1.4, excluding message encapsulation proper.

9.1. S-HTTP (Unencapsulated) Headers

```
Content-Privacy-Domain: ('CMS' | 'MOSS')
Prearranged-Key-Info: <Hdr-Cipher>,<Key>,<Key-ID>
Content-Type: 'message/http'
MAC-Info: [hex(timeofday)',']<hash-alg>','hex(<hash-data>)','
        <key-spec>
```

9.2. HTTP (Encapsulated) Non-negotiation Options

```
Key-Assign: <Method>','<Key-Name>','<Lifetime>','
        <Ciphers>';'<Method-args>
Encryption-Identity: <name-class>','<key-sel>','<name-args>
Certificate-Info: <Cert-Fmt>','<Cert-Group>
Nonce: <string>
Nonce-Echo: <string>
```

9.3. Encapsulated Negotiation Options

```
SHTTP-Cryptopts: <scope>';'<string>(,<string>)*
SHTTP-Privacy-Domains: ('CMS' | 'MOSS')
SHTTP-Certificate-Types: ('X.509')
SHTTP-Key-Exchange-Algorithms: ('DH', 'RSA' | 'Inband' | 'Outband')
SHTTP-Signature-Algorithms: ('RSA' | 'NIST-DSS')
SHTTP-Message-Digest-Algorithms:  ('RSA-MD2' | 'RSA-MD5' | 'NIST-SHS'
        'RSA-MD2-HMAC', 'RSA-MD5-HMAC', 'NIST-SHS-HMAC')
SHTTP-Symmetric-Content-Algorithms: ('DES-CBC' | 'DES-EDE-CBC' |
        'DES-EDE3-CBC' | 'DESX-CBC' | 'CDMF-CBC' | 'IDEA-CBC' |
        'RC2-CBC' )
SHTTP-Symmetric-Header-Algorithms: ('DES-ECB' | 'DES-EDE-ECB' |
        'DES-EDE3-EBC' | 'DESX-ECB' | 'CDMF-ECB' | 'IDEA-ECB' |
        'RC2-ECB')
SHTTP-Privacy-Enhancements: ('sign' | 'encrypt' | 'auth')
Your-Key-Pattern: <key-use>','<pattern-info>
```

9.4. HTTP Methods

```
Secure * Secure-HTTP/1.4
```

9.5. Server Status Reports

```
Secure-HTTP/1.4 200 OK
SecurityRetry 420
BogusHeader 421 <reason>
```

10. An Extended Example

We provide here a contrived example of a series of S-HTTP requests and replies. Rows of equal signs are used to set off the narrative from sample message traces. Note that the actual encrypted or signed message bodies would normally be binary garbage. In an attempt to preserve readability while still using (mostly) genuine messages, the bodies of the requests have been base64 encoded. To regenerate actual S-HTTP messages, it is necessary to remove the base64 encoding from the message body.

10.1. A request using RSA key exchange with Inband key reply

Alice, using an S-HTTP-capable client, begins by making an HTTP request which yields the following response page:

```
==========================================================
200 OK HTTP/1.0
Server-Name: Navaho-0.1.3.3alpha
Certificate-Info: CMS,MIAGCSqGSIb3DQEHAqCAMIACAQExADCABgkqh
        kiG9w0BBwEAAKCAM
        IIBrTCCAUkCAgC2MA0GCSqGSIb3DQEBAgUAME0xCzAJBgNVBAYTAlVTMSAwH
        gYDVQQKExdSU0EgRGF0YSBTZWN1cml0eSwgSW5jLjEcMBoGA1UECxMTUGVyc
        29uYSBDZXJ0aWZpY2F0ZTAeFw05NDA0MDkwMDUwMzdaFw05NDA4MDIxODM4N
        TdaMGcxCzAJBgNVBAYTAlVTMSAwHgYDVQQKExdSU0EgRGF0YSBTZWN1cml0e
        SwgSW5jLjEcMBoGA1UECxMTUGVyc29uYSBDZXJ0aWZpY2F0ZTEYMBYGA1UEA
        xMPU2V0ZWMgQXN0cm9ub215MFwwDQYJKoZIhvcNAQEBBQADSwAwSAJBAMy8Q
        cW7RMrB4sTdQ8Nmb2DFmJmkWn+el+NdeamIDElX/qw9mIQu4xNj1FfepfJNx
        zPvA0OtMKhy6+bkrlyMEU8CAwEAATANBgkqhkiG9w0BAQIFAANPAAYn7jDgi
        rhiIL4wnP8nGzUisGSpsFsF4/7z2P2wqne6Qk8Cg/Dstu3RyaN78vAMGP8d8
        2H5+Ndfhi2mRp4YHiGHz0HlK6VbPfnyvS2wdjCCAccwggFRAgUCQAAAFDANB
        gkqhkiG9w0BAQIFADBfMQswCQYDVQQGEwJVUzEgMB4GA1UEChMXUlNBIERhd
        GEgU2VjdXJpdHksIEluYy4xLjAsBgNVBAsTJUxvdyBBc3N1cmFuY2UgQ2Vyd
        GlmaWNhdGlvbiBBdXRob3JpdHkwHhcNOTQwMTA3MDAwMDAwWhcNOTYwMTA3M
        jM1OTU5WjBNMQswCQYDVQQGEwJVUzEgMB4GA1UEChMXUlNBIERhdGEgU2Vjd
        XJpdHksIEluYy4xHDAaBgNVBAsTE1BlcnNvbmEgQ2VydGlmaWNhdGUwaTANB
        gkqhkiG9w0BAQEFAANYADBVAk4GqghQDa9Xi/2zAdYEqJVIcYhlLN1FpI9tX
        Q1m6zZ39PYXK8Uhoj0Es7kWRv8hC04vqkOKwndWbzVtvoHQOmP8nOkkuBi+A
        QvgFoRcgOUCAwEAATANBgkqhkiG9w0BAQIFAANhAD/5Uo7xDdp49oZm9GoNc
        PhZcW1e+nojLvHXWAU/CBkwfcR+FSf4hQ5eFu1AjYv6Wqf430Xe9Et5+jgnM
        Tiq4LnwgTdA8xQX4elJz9QzQobkE3XVOjVAtCFcmiin80RB8AAAMYAAAAAAA
        AAAAA==
```

```
Encryption-Identity: DN-1779, null, CN=Setec Astronomy, OU=Persona
        Certificate,O="RSA Data Security, Inc.", C=US;
SHTTP-Privacy-Enhancements: recv-required=encrypt

<A name=tag1 HREF="shttp://www.setec.com/secret">
Don't read this. </A>
============================================================
```

An appropriate HTTP request to dereference this URL would be:

```
============================================================
GET /secret HTTP/1.0
Security-Scheme: S-HTTP/1.4
User-Agent: Web-O-Vision 1.2beta
Accept: *.*
Key-Assign: Inband,1,reply,des-ecb;7878787878787878

============================================================
```

The added Key-Assign line that would not have been in an ordinary HTTP request permits Bob (the server) to encrypt his reply to Alice, even though Alice does not have a public key, since they would share a key after the request is received by Bob. This request has the following S-HTTP encapsulation:

```
============================================================
Secure * Secure-HTTP/1.4
Content-Type: message/http
Content-Privacy-Domain: CMS

MIAGCSqGSIb3DQEHA6CAMIACAQAxgDCBqQIBADBTME0xCzAJBgNVBAYTAlVTMSAw
HgYDVQQKExdSU0EgRGF0YSBTZWN1cml0eSwgSW5jLjEcMBoGA1UECxMTUGVyc29u
YSBDZXJ0aWZpY2F0ZQICALYwDQYJKoZIhvcNAQEBBQAEQCU/R+YCJSUsV6XLilHG
cNVzwqKcWzmT/rZ+duOv8Ggb7oO/d8H3xUVGQ2LsX4kYGq2szwj8Q6eWhsmhf4oz
lvMAADCABgkqhkiG9w0BBwEwEQYFKw4DAgcECFif7BadXlw3oIAEgZBNcMexKe16
+mNxx8YQPukBCL0bWqS86lvws/AgRkKPELmysBi5lco8MBCsWK/fCyrnxIRHs1oK
BXBVlsAhKkkusk1kCf/GbXSAphdSgG+d6LxrNZwHbBFOX6A2hYS63Iczd5bOVDDW
Op2gcgUtMJq6k2LFrs4L7HHqRPPlqNJ6j5mFP4xkzOCNIQynpD1rV6EECMIk/T7k
1JLSAAAAAAAAAAAAAA==
============================================================
```

The data between the delimiters is a CMS message, RSA enveloped for Setec Astronomy.

Bob decrypts the request, finds the document in question, and is ready to serve it back to Alice.

An appropriate HTTP server response would be:

```
============================================================
HTTP/1.0 200 OK
Security-Scheme: S-HTTP/1.4
Content-Type: text/html

Congratulations, you've won.
<A href="/prize.html"
 CRYPTOPTS="Key-Assign: Inband,alice1,reply,des-ecb;020406080a0c0e0f;
 SHTTP-Privacy-Enhancements: recv-required=auth">Click here to
claim your prize</A>
============================================================
```

This HTTP response, encapsulated as an S-HTTP message becomes:

```
============================================================
Secure * Secure-HTTP/1.4
Content-Type: message/http
Prearranged-Key-Info: des-ecb,697fa820df8a6e53,inband:1
Content-Privacy-Domain: CMS

MIAGCSqGSIb3DQEHBqCAMIACAQAwgAYJKoZIhvcNAQcBMBEGBSsOAwIHBAifqtdy
x6uIMYCCARgvFzJtOZBn773DtmXlx037ck3giqnV0WC0QAx5f+fesAiGaxMqWcir
r9XvT0nT0LgSQ/8tiLCDBEKdyCNgdcJAduy3D0r2sb5sNTT0TyL9uydG3w55vTnW
aPbCPCWLudArI1UHDZbnoJICrVehxG/sYX069M8v6VO8PsJS7//hh1yM+0nekzQ5
l1p0j7uWKu4W0csrlGqhLvEJanj6dQAGSTNCOoH3jzEXGQXntgesk8poFPfHdtj0
5RH4MuJRajDmoEjlrNcnGl/BdHAd2JaCo6uZWGcnGAgVJ/TVfSVSwN5nlCK87tXl
nL7DJwaPRYwxb3mnPKNq7ATiJPf5u162MbwxrddmiE7e3sST7naSN+GS0ateY5X7
AAAAAAAAAAA=
============================================================
```

The data between the delimiters is a CMS message encrypted under a randomly-chosen DEK which can be recovered by computing:

```
        DES-DECRYPT(inband:1,697fa820df8a6e53)
```

where 'inband:1' is the key exchanged in the Key-Assign line in the original request.

10.2. A request using the auth enhancement

There is a link on the HTML page that was just returned, which Alice dereferences, creating the HTTP message:

```
============================================================
GET /prize.html HTTP/1.0
Security-Scheme: S-HTTP/1.4
User-Agent: Web-O-Vision 1.1beta
Accept: *.*

============================================================
```

Which, when encapsulated as an S-HTTP message, becomes:

```
============================================================
Secure * Secure-HTTP/1.4
Content-Type: message/http
MAC-Info:31ff8122,rsa-md5,b3ca4575b841b5fc7553e69b0896c416,inband:alice1
Content-Privacy-Domain: CMS

MIAGCSqGSIb3DQEHAaCABGNHRVQgL3ByaXplLmh0bWwgSFRUUC8xLjAKU2VjdXJp
dHktU2NoZW1lOiBTLUhUVFAvMS4xClVzZXItQWdlbnQ6IFdlYi1PLVZpc2lvbiAx
LjFiZXRhCkFjY2VwdDogKi4qCgoAAAAA
============================================================
```

The data between the delimiters is a CMS ’Data’ representation of the request.

Appendix: A Review of CMS

CMS ("Cryptographic Message Syntax Standard") is a cryptographic message encapsulation format, similar to PEM, based on RSA's PKCS-7 cryptographic messaging syntax.

CMS is only one of two encapsulation formats supported by S-HTTP, but it is to be preferred since it permits the least restricted set of negotiable options, and permits binary encoding. In the interest of making this specification more self-contained, we summarize CMS here.

CMS is defined in terms of OSI's Abstract Syntax Notation (ASN.1, defined in X.208), and is concretely represented using ASN.1's Basic Encoding Rules (BER, defined in X.209). A CMS message is a sequence of typed content parts. There are six content types, recursively composable:

Data -- Some bytes, with no enhancement.

SignedData -- A content part, with zero or more signature blocks, and associated keying materials. Keying materials can be transported via the degenerate case of no signature blocks and no data.

EnvelopedData -- One or more (per recipient) key exchange blocks and an encrypted content part.

DigestedData -- A content part with a single digest block.

EncryptedData -- An encrypted content part, with key materials externally provided.

Here we will dispense with convention for the sake of ASN.1-impaired readers, and present a syntax for CMS in informal BNF (with much gloss). In the actual encoding, most productions have explicit tag and length fields.

```
Message = *Content
Content = Data | SignedData | EnvelopedData |
                DigestedData | EncryptedData
Data = Bytes
SignedData = *DigestAlg Content *Certificates
                 *CRLs SignerInfo*
EnvelopedData = *RecipientInfo BulkCryptAlg
                Encrypted(Content)
```

```
DigestedData = DigestAlg Content DigestBytes
EncryptedData = BulkCryptAlg Encrypted(Bytes)
SignerInfo = CertID ... Encrypted(DigestBytes) ...
RecipientInfo = CertID KeyCryptAlg Encrypted(DEK)
```

Appendix: Internet Media Type message/s-http

In addition to defining the S-HTTP/1.4 protocol, this document serves as the specification for the Internet media type "message/s-http". The following is to be registered with IANA.

```
Media Type name:        message
Media subtype name:     s-http
Required parameters:    none
Optional parameters:    version, msgtype
```

version: The S-HTTP version number of the enclosed message (e.g. "1.4"). If not present, the version can be determined from the first line of the body.

msgtype: The message type -- "request" or "response". If not present, the type can be determined from the first line of the body.

Encoding considerations: only "7bit", "8bit", or "binary" are permitted.

Security considerations: this is a security protocol.

Bibliography and References

[BELL96] Bellare, M., Canetti, R., Krawczyk, H., "Keying Hash Functions for Message Authentication", Preprint.

[FIPS-46-1] Federal Information Processing Standards Publication (FIPS PUB) 46-1, Data Encryption Standard, Reaffirmed 1988 January 22 (supersedes FIPS PUB 46, 1977 January 15).

[FIPS-81] Federal Information Processing Standards Publication (FIPS PUB) 81, DES Modes of Operation, 1980 December 2.

[FIPS-180] Federal Information Processing Standards Publication (FIPS PUB) 180-1, "Secure Hash Standard", 1995 April 17.

[FIPS-186] Federal Information Processing Standards Publication (FIPS PUB) 186, Digital Signature Standard, 1994 May 19.

[HAST86] Hastad, J., "On Using RSA With Low Exponents in a Public Key Network," Advances in Cryptology-CRYPTO 95 Proceedings, Springer-Verlag, 1986.

[JOHN93] Johnson, D.B., Matyas, S.M., Le, A.V., Wilkins, J.D., "Design of the Commercial Data Masking Facility Data Privacy Algorithm," Proceedings 1st ACM Conference on Computer & Communications Security, November 1993, Fairfax, VA., pp. 93-96.

[KRAW96b] Krawczyk, H. personal communication.

[LAI92] Lai, X. "On the Design and Security of Block Ciphers," ETH Series in Information Processing, v. 1, Konstanz: Hartung-Gorre Verlag, 1992.

[PKCS-6] RSA Data Security, Inc. "Extended Certificate Syntax Standard", PKCS-6, Nov 1, 1993.

[CMS] Housley, R., "Cryptographic Message Syntax", RFC 2630, June 1999.

[RFC-822] Crocker, D., "Standard For The Format Of ARPA Internet Text Messages", STD 11, RFC 822, August 1982.

[RFC-1319] Kaliski, B., "The MD2 Message-Digest Algorithm", RFC 1319, April 1992.

[RFC-1321] Rivest, R., "The MD5 Message-Digest Algorithm", RFC 1321, April 1992.

[RFC-1421] Linn, J., "Privacy Enhancement for Internet Electronic Mail: Part I: Message Encryption and Authentication Procedures", RFC 1421, February 1993.

[RFC-1422] Kent, S., "Privacy Enhancement for Internet Electronic Mail: Part II: Certificate-Based Key Management", RFC 1422, February 1993.

[RFC-1779] Kille, S., "A String Representation of Distinguished Names", RFC 1779, March 1995.

[RFC-2045] Freed, N. and N. Borenstein, "Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies", RFC 2045, September 1993.

[RFC-1738] T. Berners-Lee, "Uniform Resource Locators (URLs)", RFC 1738, December 1994.

[RFC-1847] Galvin, J., Murphy, S., Crocker, S., and N. Freed, "Security Muliparts for MIME: Multipart/Signed and Multipart/Encrypted", RFC 1847, October 1995.

[RFC-1848] Crocker, S., Freed, N., Galvin, J., and S. Murphy, "MIME Object Security Services", RFC 1848, October 1995.

[RFC-1864] Myers, J. and M. Rose, "The Content-MD5 Header Field", RFC 1864, October 1995.

[RFC-2616] 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.

[RFC-2617] Franks, J., Hallam-Baker, P., Hostetler, J., Leach, P., Luotonen, A. and L. Stewart, "HTTP Authentication: Basic and Digest Access Authentication", RFC 2617, June 1999.

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

[SHTML] Rescorla, E. and A. Schiffman, "Security Extensions For HTML", RFC 2659, August 1999.

[VANO95] B. Prennel and P. van Oorschot, "On the security of two MAC algorithms", to appear Eurocrypt'96.

[X509] CCITT Recommendation X.509 (1988), "The Directory - Authentication Framework".

Security Considerations

This entire document is about security.

Authors' Addresses

Eric Rescorla
RTFM, Inc.
30 Newell Road, #16
East Palo Alto, CA 94303

Phone: (650) 328-8631
EMail: ekr@rtfm.com

Allan M. Schiffman
SPYRUS/Terisa
5303 Betsy Ross Drive
Santa Clara, CA 95054

Phone: (408) 327-1901
EMail: ams@terisa.com

15. Full Copyright Statement

Copyright (C) The Internet Society (1999). All Rights Reserved.

This document and translations of it may be copied and furnished to others, and derivative works that comment on or otherwise explain it or assist in its implementation may be prepared, copied, published and distributed, in whole or in part, without restriction of any kind, provided that the above copyright notice and this paragraph are included on all such copies and derivative works. However, this document itself may not be modified in any way, such as by removing the copyright notice or references to the Internet Society or other Internet organizations, except as needed for the purpose of developing Internet standards in which case the procedures for copyrights defined in the Internet Standards process must be followed, or as required to translate it into languages other than English.

The limited permissions granted above are perpetual and will not be revoked by the Internet Society or its successors or assigns.

This document and the information contained herein is provided on an "AS IS" basis and THE INTERNET SOCIETY AND THE INTERNET ENGINEERING TASK FORCE DISCLAIMS 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.

Acknowledgement

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