E. Rescorla, A. Schiffman
INTERNET-DRAFT Enterprise Integration Technologies
<draft-ietf-wts-shttp-01.txt> Feb 1996 (Expires August-96)

The Secure HyperText Transfer Protocol

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 ftp.is.co.za (Africa), nic.nordu.net (Europe), munnari.oz.au (Pacific Rim), ds.internic.net (US East Coast), or ftp.isi.edu (US West Coast).

This document describes S-HTTP version 1.2. The original draft of this specification, defining S-HTTP version 1.0, was distributed by the CommerceNet Consortium in June 1994; in December 1994 a revised specification describing S-HTTP version 1.1 was published as an Internet Draft (draft-rescorla-shttp-00.txt). In July 1995, an updated version of that draft was published as an Internet Draft. That document deprecated some unimplemented facilities, provides additional clarifying material, and made minor corrections to the 12/94 version.

This document implements a decision reached at the December 1995 IETF WTS meeting to break up the single S-HTTP document into two documents, one describing the S-HTTP messaging protocol and negotiation syntax and one describing extensions to HTML to facilitate the use of S-HTTP. The companion document is draft-ietf-wts-shtml-00.txt [23].

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) is an extension of HTTP, providing 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.

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) [18]. The ease of use of the Web has prompted widespread interest in its 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 supports 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, PKCS-7 and PEM. 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), supporting symmetric session key 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 an HTML anchor); 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 (should the request be signed? encrypted? both? what about the reply?); cryptographic algorithms (RSA vs. DSA for signing, DES vs. RC2 for encrypting, etc.); and certificate selection (please sign with your "Mastercard 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.

1.2. Changes

This document describes S-HTTP/1.2. The prior draft described S-HTTP/1.1. This version adds a number of minor changes, including a new hash construction and a new way of binding cryptographic parameters to HTML anchors. S-HTTP/1.2 messages will be readable by S-HTTP/1.1 agents and vice versa, provided that compatible algorithms are used.

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 data object.
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 merges 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 7 for more on this topic.) Using this data, the sender applies the enhancements to the message cleartext 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 4 and 5.

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 and discover what sorts of cryptographic transformations were performed on the message, then remove them using some combination of the sender's and receiver's keying material, in the process while taking note of what 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 7 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 provided, including password-style manually shared secrets, public-key key exchange and Kerberos [19] ticket distribution. In particular, provision has been made for prearranged (in an earlier transaction) symmetric session keys in order to send confidential messages to those who have no 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. Keys may also be extracted from Kerberos tickets.

1.4.3. Message Integrity and Sender Authentication

Secure HTTP provides a means to verify message integrity and sender authenticity for a HTTP 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 Kerberos. 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 cryptographic facilities for the World-Wide Web, Secure HTTP deliberately caters to a variety of implementation options despite the fact that the resulting variability makes interoperation potentially problematic.

We anticipate that some implementors will choose to integrate an outboard PEM program with a WWW client or server; such implementations will not be able to use all operation modes or features of S-HTTP, but will be able to interoperate with most other implementations. Other implementors will choose to create a full-fledged PKCS-7 implementation (allowing for all the features of S-HTTP); in which case PEM support will be only a modest additional effort. Without completely prescribing a minimum implementation profile (although see section 8) then, we recommend that all S-HTTP implementations support the PEM message format.

2. HTTP Encapsulation

A Secure HTTP message consists of a request or status line (as in HTTP) followed by a series of RFC-822 style headers followed by an encapsulated content. Once the content has been decoded, it should either be another Secure HTTP message, an HTTP message, or simple data.

For the purposes of compatibility with existing HTTP implementations, we distinguish S-HTTP transaction requests and replies with a distinct protocol designator ('Secure-HTTP/1.2'). However, if a future version of HTTP (i.e., 'HTTP/2.0') subsumes this document use of a new protocol HTTP designator would provide the same backwards compatibility function and a distinction between such a future version of HTTP and Secure-HTTP would be unnecessary.

2.1. The Request Line

For HTTP requests, we define a new HTTP protocol method, 'Secure'. All secure requests (using this version of the protocol) should read:

```
Secure * Secure-HTTP/1.2
```

All case variations should be accepted. The asterisk shown here is a placeholder and should be ignored by servers; proxy-aware clients should substitute the URL (and must provide at least the host+port portion) of the request when communicating via proxy, as is the current HTTP convention; (e.g. http://www.terisa.com/*) proxies should remove the appropriate amount of this information to minimize the threat of traffic analysis. See Section 8.2.2.1 for a situation where providing more information is appropriate.

2.2. The Status Line

For server responses, the first line should be:

```
Secure-HTTP/1.2 200 OK
```

whether the request succeeded or failed. This prevents analysis of success or failure for any request. All case variations should be accepted.

2.3. Secure HTTP Header Lines

We define a series of new header lines to go in the header of the 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 [6] should be used only as a token separator unless otherwise quoted. Long header lines may be line folded in the style of RFC822 [6].

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

2.3.1. Content-Privacy-Domain

This header line exists to provide compatibility with PEM-based Secure HTTP systems. The two values defined by this document are 'PEM' and 'PKCS-7'. PKCS-7 [2] refers to the privacy enhancement specified in section 3. PEM refers to standard PEM message format as

defined in RFC1421 [1]. Note that MOSS[25] could be accomodated simply by adding a Content-Privacy-Domain: MOSS.

2.3.2. Content-Transfer-Encoding

The PKCS-7 message format is designed for an 8-bit clear channel, but may be passed over other channels using base-64 encoding (see RFC1421 [1] for a description of base-64).

For 'Content-Privacy-Domain: PKCS-7', acceptable acceptable values for this field are 'BASE64','8BIT', or 'BINARY'. Unless such a line is included, the rest of the message is assumed to be 'BINARY'. (Note that the difference between 'BINARY' and '8BIT' has to do with line length.)

For 'Content-Privacy-Domain: PEM', the only acceptable value for this field is '7BIT', since PEM messages are already encoded for RFC-822 (and hence 7-bit) transport.

2.3.3. Content-Type

Under normal conditions, the terminal encapsulated content (after all privacy enhancements have been removed) shall be considered to be an HTTP/1.0 message. In this case, there shall be a Content-Type line reading:

```
Content-Type: application/http
```

It is intended that this type be registered with IANA as a MIME content type. For backwards compatibility, 'application/x-http' is also acceptable.

However, the terminal content may be of some other type provided that that type is properly indicated by the use of an appropriate Content-Type header line. In this case, the header fields for the last (most deeply encapsulated) HTTP or S-HTTP message should be applied to the terminal content. It should be noted that unless the (S-)HTTP message from which the headers are taken is itself enveloped, then some possibly sensitive information has been passed in the clear.

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

2.3.4. 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 three methods for exchanging named keys, Inband, Kerberos and Outband. Inband and Kerberos 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: <Hdr-Cipher>','<CoveredDEK>','<CoverKey-ID>
<CoverKey-ID> := <method>':'<key-name>
<CoveredDEK> := <hex-digits>
<method> := 'inband' | 'krb-'<kv> | 'outband'
<kv> := '4' | '5'
```

While chaining ciphers require an Initialization Vector (IV) [16] 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 4.4.7).

<CoveredDEK> is the protected Data Exchange 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.

2.3.5. 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 [21] has demonstrated some insecurities in this approach, and this draft introduces a new construction. 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 2.3.5 for the names) that obey a different (hopefully stronger) construction.

```
MAC = 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. 48 times for MD5, 44 for
        SHA)
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 security cannot be increased past the length of the hash function itself, but can be reduced by using a shorter key.) [22] 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 security of short keys.

The format of the MAC-Info line is:

```
MAC-Info: [hex(<time>)],<hash-alg>, hex(<hash-data>),<key-spec>
<time> := "unsigned seconds since Unix epoch"
<hash-alg> := "hash algorithms from section 4.4.5"
<hash-data> := "computation as described above"
<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 this 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 version of the original HTTP Basic authentication mode in that the user can be made 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).

2.4. Content

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

For a PKCS-7 message, with '8BIT' Content-Transfer-Encoding, the content should simply be the PKCS-7 message itself.

If the Content-Transfer-Encoding is 'BASE64', the content should be preceded by a line that reads:

```
-----BEGIN PRIVACY-ENHANCED MESSAGE-----
```

and followed by a line that reads

```
-----END PRIVACY-ENHANCED MESSAGE-----
```

(see RFC1421) with the content simply being the base-64 representation of original content. If the inner (protected) content is itself a PKCS-7 message, than the ContentType of the outer content should be set appropriately. Else, the ContentType should be represented as 'Data'.

If the Content-Privacy-Domain is PEM, the content should consist of a normal encapsulated message, beginning with:

```
-----BEGIN PRIVACY-ENHANCED MESSAGE-----
```

and ending with

-----END PRIVACY-ENHANCED MESSAGE-----

as defined in RFC1421.

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, as in, for instance, 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 6.2.4.

3. Message Format Options

3.1. Content-Privacy-Domain: PKCS-7

Content-Privacy-Domain 'PKCS-7' follows the form of the PKCS-7 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 PKCS-7 coding via the MAC-Info header of section 2.3.5 since PKCS-7 does not support a 'KeyDigestedData' type, although it does support a 'DigestedData' type.

3.1.1. Signature

This enhancement uses the 'SignedData' (or 'SignedAndEnvelopedData') type of PKCS-7. When digital signatures are used, an appropriate certificate may either be attached to the message (possibly along with a certificate chain) as specified in PKCS-7 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 PKCS-7 compliant.

3.1.2. Encryption

3.1.2.1. Encryption -- normal, public key

This enhancement is performed precisely as enveloping (using either 'EnvelopedData' or 'SignedAndEnvelopedData' types) under PKCS-7. A message encrypted in this fashion, signed or otherwise, is PKCS-7 compliant.

3.1.2.2. Encryption -- prearranged key

This uses the 'EncryptedData' type of PKCS-7. 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 generate signed, encrypted data, it is necessary to generate the 'SignedData' production and then encrypt it (since PKCS-7 does not support a 'SignedAndEncryptedData' type).

3.2. Content-Privacy-Domain: PEM

This Content-Privacy-Domain simply refers to using straight PEM messages as per section 2.3.1. Note that clients and servers which implement the original HTTP access authorization protocols (as proposed by Tony Sanders and originally implemented by Rob McCool) can be converted to use S-HTTP (using this Content-Privacy-Domain) simply by changing the request/results lines to match S-HTTP and by adding the following three lines to the header:

```
Content-Privacy-Domain: PEM
Content-Type: application/http
Content-Transfer-Encoding: 7BIT
```

It would be helpful (but not necessary) to remove the 'authorization' line. No cryptographic transformations are necessary.

3.2.1. Correspondence of PEM and S-HTTP Modes

S-HTTP message protection modes for the PEM Content-Privacy-Domain necessarily follow the enhancement modes of PEM, that is: S-HTTP messages which are to be signed use PEM's MIC-ONLY (or MIC-CLEAR) mode; S-HTTP messages which are to be both signed and encrypted (using RSA key exchange) use PEM's misnamed ENCRYPTED enhancement mode.

3.3. HTTP/1.1 Header Interaction

3.3.1. Overview

HTTP/1.1 [23], while as yet in draft form, describes a number of header lines which have potential interactions with S-HTTP.

3.3.2. Content-Encoding

The Content-Encoding line is described as intended for compression or encryption. Since S-HTTP has it's own syntax for describing encryption, that use is inapplicable here. Compression is also in general inapplicable to encrypted data and, if desired, should be applied to the inner content, rather than to the S-HTTP message.

3.3.3. Transfer-Encoding

Transfer-Encoding (and in particular the 'chunk' mode) is, as stated in [23] intended to uniquely delimit the boundaries of the message. Since the message formats used by S-HTTP unambiquously define the end of the message, chunk transfer encodings are unnecessary, and it is an error to use one in the outer content of an S-HTTP message. (And redundant to use one in the inner content.)

3.3.4. Connection

The Connection header line is permitted in the header lines of S-HTTP requests and should be treated exactly as if the requests were HTTP requests. If the recipient of a message sees different values for the Connection header in an S-HTTP message and the inner HTTP content, the S-HTTP value should be ignored. However, if the Connection header appears only in the S-HTTP message but not in the inner HTTP content, it should be treated as if it appeared in the inner content.

If a server sees a Connection header in the S-HTTP header it should acknowledge it in the S-HTTP header of it's response. If it sees it in the HTTP header, it should acknowledge it in the HTTP header of it's response.

3.3.5. Keep-Alive

The Keep-Alive header line is permitted in the header lines of S-HTTP requests and should be treated exactly as if the requests were HTTP requests. If the recipient of a message sees different values for the Keep-Alive header in an S-HTTP message and the inner HTTP content, the S-HTTP value should be ignored. However, if the Keep-Alive header appears only in the S-HTTP message but not in the inner HTTP content, it should be treated as if it appeared in the inner content.

If a server sees a Keep-Alive header in the S-HTTP header it should acknowledge it in the S-HTTP header of it's response. If it sees it in the HTTP header, it should acknowledge it in the HTTP header of it's response.

3.3.6. If-Modified-Since

This may be used by the proxy to indicate that the document may be in it's cache and that it is prepared to serve the document to the current requestor. Servers receiving this header and deciding not to resend the document should respond using the 320 response code as described in Section 6.2.5.

This header should only be placed in S-HTTP headers by proxies. Clients wanting to use If-Modified-Since should place it in the HTTP headers of the inner content.

3.3.7. Content-MD5

Servers may generate a Content-MD5 header to enable proxies to detect when valid cache hits have occurred. Note that the Content-MD5 header provides the possibility of traffic analysis and servers using this should bear that risk in mind.

3.3.8. Other headers

No other HTTP/1.1 header lines should be placed in S-HTTP headers. If they are found, it is an error. Servers should respond with the 421 BogusHeader error.

4. Negotiation

4.1. Negotiation Overview

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

A negotiation block 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 (with respect to the negotiator).

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

As an example, the negotiation header:

```
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.''

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

4.2. Negotiation Header Format

The general format for negotiation header lines is:

```
<Line> := <Field> ':' <Key-val>(';'<Key-val>)*
<Key-val> := <Key> '=' <Value>(','<Value>)*
<Key> := <Mode>'-'<Action>
<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.

MANGROVE 1004

'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.

4.3. 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

    <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.

4.4. Negotiation Blocks

As described in Section 1.X.Y and in the previous section, 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.

In the case of a server, the scope and meaning of options is clear; they precondition the server's response to the request in which the options appear. However, since an HTTP response which contains an HTML document (as opposed to error returns as discussed in Section 6.2.X) may contain multiple references, some mechanism is needed to bind options to the various references.

Binding negotiation options to anchors using HTML extensions HTML is the topic of the companion document draft-ietf-wts-shtml-00.txt and will not be treated here.

Here, we provide a syntax to bind a group of negotiation options to a specific reference using standard HTML.

4.4.1. SHTTP-Cryptopts-Scope

This header line provides a list of named anchors in an HTML document (assigned using the NAME tag) to which the following set of negotiation headers (until the end of the headers or the next SHTTP-Cryptopts-Scope header, whichever comes first). The names are provided as a comma separated list. For instance

```
SHTTP-Cryptopts-Scope: foo,bar,baz
```

As a special case, any headers which appear before the first SHTTP-Cryptopts-Scope header are considered to to apply to all references in the HTML document unless those references are otherwise bound. Note that this is an all-or-nothing proposition. That is, if a SHTTP-Cryptopts-Scope header binds headers to a reference, then none of these default headers apply, even if some of the default headers do not appear in the bound headers. Rather, the S-HTTP defaults found in Section 4.5.11 apply.

4.5. Negotiation Syntax

4.5.1. SHTTP-Privacy-Domains

This header line 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=pkcs-7;
                       recv-optional=pkcs-7,pem
```

would indicate that the agent always generates PKCS-7 compliant messages, but can read PKCS-7 or PEM (or, unenhanced messages).

All the negotiation headers described below can be considered to apply to all privacy domains (message formats) or to a particular one. To specify negotiation parameters which apply to all privacy domains, those header line(s) should be provided before any privacy-domain specifier. Negotiation headers which follow a privacy-domain header are considered to apply only to that domain. Multiple privacy-domain headers specifying the same privacy domain are permitted, in order to support multiple parameter combinations.

4.5.2. SHTTP-Certificate-Types

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

4.5.3. SHTTP-Key-Exchange-Algorithms

This line indicates which algorithms may be used for key exchange. Defined values are 'RSA', 'Outband', 'Inband', and 'Krb-'<kv>. RSA refers to RSA enveloping. Outband refers to some sort of external key agreement. Inband and Kerberos refer to the protocols of sections 5.4.1 and 5.4.2 respectively.

So, 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, RSA;
                               recv-required=RSA
```

4.5.4. SHTTP-Signature-Algorithms

This indicates what Digital Signature algorithms may be used. Defined values are 'RSA' and 'NIST-DSS' [17]. Since NIST-DSS and RSA use variable length moduli the parametrization syntax of section 4.3 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.

4.5.5. SHTTP-Message-Digest-Algorithms

This indicates what message digest algorithms may be used. Previously defined values are 'RSA-MD2' [7], 'RSA-MD5' [8], 'NIST-SHS' [9]. New digest algorithms 'RSA-MD2-HMAC', 'RSA-MD5-HMAC', and 'NIST-SHS-HMAC' are defined as the construction of 2.3.5 using the algorithms MD2, MD5, and SHA-1 respectively.

4.5.6. 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 [11])
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 [12]
RC2-CBC -- RSA's RC2 in CBC mode
CDMF-CBC -- IBM's CDMF (weakened key DES) [20] in CBC mode
```

Since RC2 keys are variable length, the syntax of section 4.3 should be used.

4.5.7. 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 [11])
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 4.3 should be used.

4.5.8. 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.

4.5.9. Your-Key-Pattern

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

```
Your-Key-Pattern : <key-use>,<pattern-info>
<key-use> := 'cover-key' | 'auth-key' | 'signing-key' | 'krbID-'<kv>
```

4.5.9.1. Cover Key Patterns

This parameter specifies desired values for key names used for encryption of transaction keys using the Prearranged-Key-Info syntax of section 2.3.4. 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.

4.5.9.2. 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.

4.5.9.3. 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-1485'. This parameter specifies desired values for fields of Distinguished Names. DNs are considered to be represented as specified in RFC1485, the order of fields and whitespace between fields is not significant.

Pattern-data is a modified RFC1485 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.

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 RFC1485 quoting to protect the comma (an RFC1485 field separator) and the POSIX 1003.2 quoting to protect the dot (a regular expression metacharacter).

```
Your-Key-Pattern: DN-1485,
            /OU=Persona Certificate, O="RSA Data Security, Inc\."/
```

4.5.9.4. Kerberos ID Pattern

This specifies acceptable Kerberos realms for the sender of the message being referred to by the negotiation headers. in the form of the name of a Kerberos principal; i.e.:

```
<user>@<realm>
```

(This specification only supports the common 'domain style' of Kerberos realm names.) 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.

4.5.10. Example

A representative header block for a server follows.

```
SHTTP-Privacy-Domains: recv-optional=PEM, PKCS-7;
                       orig-required=PKCS-7
SHTTP-Certificate-Types: recv-optional=X.509;
                         orig-required=X.509
SHTTP-Key-Exchange-Algorithms: recv-required=RSA;
                               orig-optional=Inband,RSA
SHTTP-Signature-Algorithms: orig-required=RSA; recv-required=RSA
SHTTP-Privacy-Enhancements: orig-required=sign;
                            orig-optional=encrypt
```

4.5.11. Defaults

Explicit negotiation parameters take precedence over default values. For a given negotiation header line 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=PKCS-7, PEM;
                       recv-optional=PKCS-7, PEM
SHTTP-Certificate-Types: orig-optional=X.509;
                         recv-optional=X.509
SHTTP-Key-Exchange-Algorithms: orig-optional=RSA,Inband;
                               recv-optional=RSA,Inband
SHTTP-Signature-Algorithms: orig-optional=RSA; recv-optional=RSA;
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
```

5. New HTTP Header Lines

We define a series of new header lines which go in the HTTP header block (i.e., in the encapsulated content) so that they may be cryptographically protected.

5.1. Security-Scheme

This mandatory header line specifies the version of the protocol (although it may be used by other security protocols). This header, with a value of 'S-HTTP/1.2' must be generated by every agent to be compatible with this specification.

Note that this is a mandatory HTTP header, meaning that an agent compliant with this specification must generate this line for every HTTP message, NOT just S-HTTP messages.

5.2. Encryption-Identity

This header line identifies a potential principal for whom the message described by these options could be encrypted; this permits

return encryption under (say) public key without the other agent signing first (or under a different key than that of the signature). Or, in the Kerberos case, provides information as the agent's Kerberos identity. The syntax of the Encryption-Identity line is:

```
Encryption-Identity: <name-class>,<key-sel>,<name-arg>
<name-class> := 'DN-1485' | 'krbID-'<kv>
```

The name-class is an ASCII string representing the domain within which the name is to be interpreted, in the spirit of the new MOSS drafts. There are two currently defined name classes, "DN-1485" and "KRB-{4,5}". Key-sel is a selector for (possibly numerous) keys bound to the same name-form. For name-forms where there is only one possible key, this field should be ignored. It is the intent here to absorb the newly flexible MOSS name forms once they are firm. Name-arg is an appropriate argument for the name-class, described in sections 5.2.1 and 5.2.2 below.

5.2.1. DN-1485 Name Class

The argument is an RFC-1485 encoded DN.

5.2.2. KRB-* Name Class

The argument is the name of a Kerberos principal, i.e.:

```
<user>@<realm>
```

This specification only supports the common 'domain style' of Kerberos realm names.

5.3. 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 header line. The format of this header line is:

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

<Cert-Fmt> should be the type of <Cert-Group> being presented. Defined values are 'PEM' and 'PKCS-7'. PKCS-7 certificate groups are provided as a base-64 encoded PKCS-7 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.

5.4. Key-Assign

This header line 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: <Method>,<Key-Name>,<Lifetime>,<Ciphers>;<Method-args>

<Key-name> := <string>
<Lifetime> := 'this' | 'reply'
<Method> :='inband' | 'krb-'<kv>
<Ciphers> := 'null' | <Cipher>+
<Cipher> := "Header cipher from section 4.4.7"
<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 4.4.7). 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 (or the duration of the connection, for future versions of HTTP that support retained connections). If this appears in a CRYPTOPTS block, it indicates that it is good for at least one (but perhaps only one) dereference of this anchor; the validity period for such a key is a local matter.

Method should be one of a number of key exchange methods. The currently defined values are 'inband', 'krb-4' and 'krb-5', referring respectively to Inband keys (i.e., direct assignment) and Kerberos versions 4 and 5 respectively. Method-args will depend on methods.

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>
```

5.4.1. 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. E.g.:

```
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.

5.4.2. Kerberos Key Assignment

This permits the binding of the shared secret derived from a Kerberos ticket/authenticator pair to a symbolic keyname. In this case, method-args should be the ticket/authenticator pair (each base64-encoded), comma separated. E.g.:

```
Key-Assign: krb-4,akerbkey,reply,DES-ECB;<krb-ticket>,<krb-auth>
```

5.5. 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.

5.5.1. 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 nonce header lines may be used, each to be echoed independently.

5.5.2. Nonce-Echo

The header is used to return the value provided in a previously received Nonce: field.

6. (Retriable) Server Status Error Reports

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

6.1. 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 SHTTP, 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 here 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.

6.2. Specific Retry Behavior

6.2.1. 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.]

6.2.2. 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.

6.2.2.1. 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.

6.2.2.2. SecurityRetries for HTTP Requests

If this header is made 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.

6.2.3. 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
```

6.2.4. 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.

6.2.5. 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 PKCS7 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 8.1.X except that the message body is elided.)

6.2.6. 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.

6.3. 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.

6.3.1. 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.

6.3.2. 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.

6.3.3. Automatic MAC Authentication Retry

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

7. Other Issues

7.1. 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.

7.2. 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 be enveloped (e.g., the request is to be encrypted). Use of these secure URLs permit the additional anchor attributes described in the following section.

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.

7.3. Server Conventions

7.3.1. Certificate Requests

We define the convention that issuing a normal HTTP request:

```
GET /SERVER-CERTIFICATE[-<DN>] <http-version>
```

shall cause the server to return the corresponding certificate. <DN>

is the base-64 encoding (to protect whitespace) of the fully-specified canonical ASCII form for the DN of the requested certificate (as in RFC 1485). If no DN is specified, then the server shall choose whatever certificate it deems most appropriate. The server should sign the response with the key corresponding to the DN supplied, if the DN is unspecified by the request.

7.3.2. Policy Requests

Servers should (but not must) store the policies of the Policy Certification Authorities, if available, corresponding to their various certificates. The convention for retrieving such policies via HTTP is the request:

```
GET /POLICY-<DN> <http-version>
```

Again, <DN> is the DN (encoded as per section 7.3.1) of the certificate corresponding to the requested policy. It is recommended that this document be (pre-) signed by the PCA.

7.3.3. CRL Requests

Servers should (but not must) store the CRLs of the PCAs corresponding to their various certificates. The convention for retrieving such CRLs is:

```
GET /CRL-<DN> <http-version>
```

Again, <DN> is the DN (encoded as per section 7.3.1) of the certificate corresponding to the requested CRL.

7.4. Browser Presentation

7.4.1. 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.

7.4.2. 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.

7.4.3. 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 not recommended).

7.4.4. 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.

8. Implementation Notes

8.1. 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.

8.1.1. 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 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 8.2 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.

8.1.2. 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.

8.1.3. 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 simply take the signed document (a PKCS-7 or PEM 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) and send the message out. For example:

```
SECURE * Secure-HTTP/1.2
Content-Type: text/html
Content-Privacy-Domain: PKCS-7
Content-Transfer-Encoding: base64

-----BEGIN PRIVACY-ENHANCED MESSAGE-----
Random signed message here...
-----END PRIVACY-ENHANCED MESSAGE-----
```

8.1.4. Recursive Encapsulation

Consider a slight variation of the previous situation, where confidentiality is also required. This can be dealt with via a recursive encapsulation. That is, the S-HTTP message shown above can be used as the inner content of a new S-HTTP message, like so:

```
SECURE * Secure-HTTP/1.2
Content-Type: application/http
Content-Privacy-Domain: PKCS-7
Content-Transfer-Encoding: base64

-----BEGIN PRIVACY-ENHANCED MESSAGE-----
Encrypted version of the message above...
-----END PRIVACY-ENHANCED MESSAGE-----
```

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 sort of 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.

8.1.5. 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 PEM and PKCS-7 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.[26]

8.2. Proxy Interaction

The use of S-HTTP presents considerable challenges 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.

8.2.1. 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 enhanced, it should return the 422 response code (X.Y.Z).

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 X.Y.Z 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.

8.2.2. Proxy Caching of S-HTTP Messages

Although it is often considered that security in general and confidentiality in specific obviate caching, this is only true under certain circumstances. For example, when confidentiality is being used to restrict access to some class of documents to a broad class of users, and those users are behind a single proxy, it is obviously advantageous if that proxy can cache such documents. S-HTTP's message orientation makes this a fairly straightforward proposition, provided that the parties cooperate.

8.2.2.1. Client Behavior

All the client needs to do is to provide enough URL information to the proxy to enable the proxy to detect when potentially cached data is being requested. In order to do this, the client simply provides the whole URL HTTP style instead of the URI-less URL described in Section 2.1. Note that this provides the proxy with the URI. Consequently, clients which don't trust their proxy to receive that information or are worried about traffic analysis by the proxy should not enable caching in this way. (An insecure channel to the proxy can be defended against using a recursive encapsulation.)

8.2.2.2. Proxy Behavior

When forwarding requests, the proxy merely needs to recognize URLs that are in it's cache and add the If-Modified-Since header as it does for HTTP.

When forwarding responses, the proxy needs to detect the 320 response and reassemble a valid S-HTTP response from the cached data and the new keying material provided by the server. The proxy should check the Content-MD5 header if supplied to ensure that a valid cache hit

has occurred and retry the request minus the If-Modified-Since header if the Content-MD5s do not match.

8.2.2.3. Server Behavior

The server needs to detect the If-Modified-Since header provided by the proxy and generate the content-less message described in X.Y.Z. The logic for this decision should be the same logic that is applied in HTTP.

9. Implementation Recommendations and Requirements

All S-HTTP agents must support the MD5 message digest and MAC authentication. As of S-HTTP/1.2 All agents must also support the RSA-MD5-HMAC construction.

All S-HTTP agents must support Outband key exchange.

Support for encryption is recommended; agents which implement encryption must support the in-band key exchange method and one of the following three cryptosystems (in ECB and CBC modes): DES, RC2[40] and CDMF.

Agents are recommended to support signature verification; server support of signature generation is additionally recommended.

Note that conformant implementations of the protocol (although not recommended ones) can avoid the use of public key cryptography entirely.

10. Protocol Syntax Summary

We present below a summary of the main syntactic features of S-HTTP/1.2, excluding message encapsulation proper.

10.1. S-HTTP (Unencapsulated) Headers

```
Content-Privacy-Domain: ('PKCS-7' | 'PEM')
Content-Transfer-Encoding: ('8BIT' | '7BIT' | 'BASE64')
Prearranged-Key-Info: <Hdr-Cipher>,<Key>,<Key-ID>
Content-Type: 'application/http'
MAC-Info: [hex(timeofday)',']<hash-alg>','hex(<hash-data>)','
        <key-spec>
```

10.2. HTTP (Encapsulated) Non-negotiation Headers

```
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>
```

10.3. Encapsulated Negotiation Headers

```
SHTTP-Cryptopts-Scope: <string>(,<string>)*
SHTTP-Privacy-Domains: ('PKCS-7' | 'PEM')
SHTTP-Certificate-Types: ('X.509')
SHTTP-Key-Exchange-Algorithms: ('RSA' | 'KRB-'<kv>)
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>
```

10.4. HTTP Methods

```
Secure * Secure-HTTP/1.2
```

10.5. Server Status Reports

```
Secure-HTTP/1.2 200 OK
SecurityRetry 420
BogusHeader 421 <reason>
```

10.6. Server Conventions

```
GET SERVER-CERTIFICATE-<B64-DN> <http-version>
GET POLICY-<B64-DN> <http-version>
GET CRL-<B64-DN> <http-version>
```

11. 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, since we use base-64 encoding here for expository purposes, the example messages have the otherwise unnecessary PEM-style "BEGIN/END PRIVACY-ENHANCED MESSAGE" delim- iters.

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

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

```
==============================================================
200 OK HTTP/1.0
Server-Name: Navaho-0.1.2.3alpha
SHTTP-Cryptopts-Scope: foobar
Certificate-Info: MIAGCSqGSIb3DQEHAqCAMIACAQExADCABgkqhkiG9w0BBwEAAKCAM
        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-1485, null, CN=Setec Astronomy, OU=Persona
        Certificate,O="RSA Data Security, Inc.", C=US
SHTTP-Privacy-Enhancements: recv-required=encrypt

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

(Note that this uses HTTP header syntax of section 4.4.1. An appropriate HTTP request to dereference this URL would be:

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

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

MANGROVE 1004

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/1.2
Content-Transfer-Encoding: base64
Content-Type: application/http
Content-Privacy-Domain: PKCS-7

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

The data between the delimiters is a PKCS-7 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/1.2
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-HTTP/1.2 200 OK
Content-Transfer-Encoding: base64
Content-Type: application/http
Prearranged-Key-Info: des-ecb,03e5d6f7997eaa5b,inband:1
Content-Privacy-Domain: PKCS-7

-----BEGIN PRIVACY-ENHANCED MESSAGE-----
MIAGCSqGSIb3DQEHBqCAMIACAQAwgAYJKoZIhvcNAQcBMBEGBSsOAwIHBAiDM8nY
HcK+IoCCARir/4frekvV8FJufQFzHJVn3rWXMYovumgzNXJQfPAr+oysnjmg5dtG
i96aMkhM4BF21rebPHwii+PZocEqiea1ibkRvzCnAiNie2EUzMgx1fh8Uro49I33
zTjqrkKngZeDCvU1y2x1l2FPrMpHm9/zafLKs9oznnkm0GGbz75mBomIrywuST7b
DYj52btqR24qdO573CPdBXQNkjEVI2lAuWqIINDZ49gKi5DZRTYW7zzM13SExN5U
ECajW+zEcnuW0WxYOu1Dh8gywWzBvmi59sKwLe69FvJiuhQFtdL2wngiQRlGtdjF
tSwlGKmHJsrSonewRPJ0SVBlmBRp+Pi6iwJns3K6Z00hqwRp8jNkmoAO2DP8WNi0
AAAAAAAAAAA=
-----END PRIVACY-ENHANCED MESSAGE-----
=============================================================
```

The data between the delimiters is a PKCS7 message encrypted under a randomly-chosen DEK which can be recovered by computing:

```
DES-DECRYPT(inband:1,03e5d6f7997eaa5b)
```

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

11.2. 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/1.2
User-Agent: Web-O-Vision 1.1beta
Accept: *.*

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

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

```
==============================================================
Secure * Secure-HTTP/1.2
Content-Transfer-Encoding: base64
Content-Type: application/http
MAC-Info:2ffc120b,rsa-md5,1425a951f1bbf3bd8d6dc7d07ab731bb,inband:alice1
Content-Privacy-Domain: PKCS-7

-----BEGIN PRIVACY-ENHANCED MESSAGE-----
MIAGCSqGSIb3DQEHAYBjR0VUIC9wcml6ZS5odG1sIEhUVFAvMS4wClNlY3VyaXR5
LVNjaGVtZTogUy1IVFRQLzEuMQpVc2VyLUFnZW50OiBXZWItTy1WaXNpb24gMS4x
YmV0YQpBY2NlcHQ6ICouKgoKAAA=
-----END PRIVACY-ENHANCED MESSAGE-----
==============================================================
```

The data between the delimiters is a PKCS-7 'Data' representation of the request.

Appendix: A Review of PKCS-7

PKCS-7 ("Cryptographic Message Syntax Standard") is a cryptographic message encapsulation format, similar to PEM, which was defined by RSA Laboratories as part of a family of related standards. They state: "The PKCS standards are offered by RSA Laboratories to developers of computer systems employing public key cryptography. It is RSA Laboratories' intention to improve and refine the standards in conjunction with computer system developers, with the goal of producing standards that most if not all developers adopt."

PKCS-7 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 PKCS-7 here.

PKCS-7 is a superset of PEM, in that PEM messages can be converted to PKCS-7 messages without any cryptographic operations, and vice-versa (given PKCS-7 messages which are restricted to PEM facilities). Additionally, PEM key management materials such as certificates and certificate revocation lists are compatible with PKCS-7's.

PKCS-7 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 PKCS-7 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.

SignedAndEnvelopedData -- The obvious combination of SignedData and EnvelopedData for a single 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 PKCS-7 in informal BNF (with much gloss). In the actual encoding, most productions have explicit tag and length fields.

```
<Message> := (<Content>)+
<Content> := <Data> | <SignedData> | <EnvelopedData> |
                <SignedAndEnvelopedData> |
                <DigestedData> | <EncryptedData>
<Data> := <Bytes>
<SignedData> := <DigestAlg>* <Content> <Certificates>*
                 <CRLs>* <SignerInfo>*
<EnvelopedData> := <RecipientInfo>+ <BulkCryptAlg>
                Encrypted(<Content>)
<SignedAndEnvelopedData> := <RecipientInfo>* <DigestAlg>*
                <EncryptedData> <Certificates>*
                <CRLs>* <SignerInfos>*
<DigestedData> := <DigestAlg> <Content> <DigestBytes>
<EncryptedData> := <BulkCryptAlg> Encrypted(<Bytes>)
<SignerInfo> := <CertID> ... Encrypted(<DigestBytes>) ...
<RecipientInfo> := <CertID> <KeyCryptAlg> Encrypted(<DEK>)
```

References

[1] Linn J. "Privacy Enhancement for Internet Electronic Mail: Part I: Message Encryption and Authentication Procedures", RFC1421, Feb 1993.

[2] RSA Data Security, Inc. "Cryptographic Message Syntax Standard", PKCS-7, Nov 1, 1993.

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

[4] Kent, S. "Privacy Enhancement for Internet Electronic Mail: Part II: Certificate-Based Key Management", RFC1422, Feb 1993.

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

[6] Crocker, D. "Standard For The Format Of ARPA Internet Text Messages", RFC822, August 1982.

[7] Kaliski, B. "The MD2 Message-Digest Algorithm", RFC1319, April 1992

[8] Rivest, R. "The MD5 Message-Digest Algorithm", RFC1321, April 1992

[9] Federal Information Processing Standards Publication (FIPS PUB) 180, "Secure Hash Standard", 1993 May 11.

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

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

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

[13] Hardcastle-Kille, S. "A String Representation of Distinguished Names", RFC1485, July 1993.

[14] pgformat.doc (v 2.6). Obtainable from net-dist.mit.edu/pub/PGP.

[15] Berners-Lee, T., Connolly, D., "Hypertext Markup Language - 2.0", draft-ietf-html-spec-04, June 1995 (working draft)

[16] Berners-Lee, T. "Uniform Resource Locators (URLs)", RFC1738, Dec 1994

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

[18] Berners-Lee, T., Fielding, R. T., Nielsen, H., "Hypertext Transfer Protocol -- HTTP/1.0", draft-ietf-http-v10-spec-00, March 1995. (working draft)

[19] Kohl, J., and Neuman, C., "The Kerberos Authentication Service (V5)", RFC1510, September 1993.

[20] 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.

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

[22] Krawczyk, H. personal communication.

[23] Need reference for HTTP/1.1

[24] Rescorla, E., Schiffman, A., "Security Extensions For HTML", draft-ietf-wts-shtml-00.txt.

[25] MOSS

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

Security Considerations

This entire document is about security.

Acknowledgements

The authors wish to thank our colleagues at Enterprise Integration Technologies, RSA Data Security, TIS, BBN, HP Labs Bristol, NCSA, Spyglass, MIT, CERN, Open Market, Spry, Digital, W3C and elsewhere for their review of earlier drafts. We also wish to thank the many users who shared their experience with the S-HTTP reference implementation distributed by the CommerceNet Consortium.

This work was initiated at Enterprise Integration Technologies Corporation and funded in part by the ARPA MADE (Manufacturing Automation and Design Engineering) program, under contract management by

the USAF Wright Laboratory. In addition to the funding support, we appreciate the administrative and intellectual resources of the sponsors and the research community they maintain.

Authors' Address

Eric Rescorla <ekr@terisa.com>
Terisa Systems, Inc.
4984 El Camino Real
Los Altos, CA 94022
Phone: (415) 919-1753

Allan M. Schiffman <ams@terisa.com>
Terisa Systems, Inc.
4984 El Camino Real
Los Altos, CA 94022
Phone: (415) 919-1755

Table of Contents