Network K. Burdis
Internet-Draft Rhodes University
Expires: November 28, 2003 R. Naffah
Forge Research
May 30, 2003

Secure Remote Password Authentication Mechanism
draft-burdis-cat-srp-sasl-08

Status of this Memo

Copyright Notice

Abstract

This document describes an authentication mechanism based on the Secure Remote Password protocol (SRP-6) and how to use it with the authentication frameworks Secure Authentication and Security Layer (SASL), Generic Security Services Application Programming Interface (GSS-API) and Extensible Authentication Protocol (EAP). This mechanism performs mutual authentication and can provide a security layer with replay detection, integrity protection and/or confidentiality protection.

Table of Contents

1. Introduction

The Secure Remote Password (SRP) is a password-based, zero-knowledge, authentication and key-exchange protocol developed by Thomas Wu. It has good performance, is not plaintext-equivalent and maintains perfect forward secrecy. It provides authentication (optionally mutual authentication) and the negotiation of a shared context key [SRP].

The mechanism described herein is based on the SRP-6 protocol, described in [SRP-6] and [SRP-6i]. SRP-6 is an improved version of the original SRP protocol (also called SRP-3) described in [RFC-2945]. Due to the design of the mechanism, mutual authentication is MANDATORY.

2. Conventions Used in this Document

- A hex digit is an element of the set:

  {0, 1, 2, 3, 4, 5, 6, 7, 8 , 9, A, B, C, D, E, F}

  A hex digit is the representation of a 4-bit string. Examples:

  7 = 0111

  A = 1010

- An octet is an 8-bit string. In this document an octet may be written as a pair of hex digits. Examples:

  7A = 01111010

  02 = 00000010

- All data is encoded and sent in network byte order (big-endian).

- The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC-2119].

3. Data Element Formats

This section describes the encoding of the data elements used by the mechanism described in this document.

3.1 Scalar Numbers

Scalar numbers are unsigned quantities. Using b[k] to refer to the k-th octet being processed, the value of a two-octet scalar is:

```
((b[0] << 8) + b[1]),
```

where << is the bit left-shift operator. The value of a four-octet scalar is:

```
((b[0] << 24) + (b[1] << 16) + (b[2] << 8) + b[3]).
```

3.2 Multi-Precision Integers

Multi-Precision Integers, or MPIs, are positive integers used to hold large integers used in cryptographic computations.

MPIs are encoded using a scheme inspired by that used by OpenPGP - [RFC-2440] (section 3.2) - for encoding such entities:

> The encoded form of an MPI SHALL consist of two pieces: a two-octet scalar that represents the length of the entity, in octets, followed by a sequence of octets that contain the actual integer.
>
> These octets form a big-endian number; A big-endian number can be encoded by prefixing it with the appropriate length.
>
> Examples: (all numbers are in hexadecimal)
>
> > The sequence of octets [00 01 01] encodes an MPI with the value 1, while the sequence [00 02 01 FF] encodes an MPI with the value of 511.
>
> Additional rule:
>
> * The length field of an encoded MPI describes the octet count starting from the MPI's first non-zero octet, containing the most significant non-zero bit. Thus, the encoding [00 02 01] is not formed correctly; It should be [00 01 01].

We shall use the syntax mpi(A) to denote the encoded form of the

multi-precision integer A. Furthermore, we shall use the syntax bytes(A) to denote the big-endian sequence of octets forming the multi-precision integer with the most significant octet being the first non-zero octet containing the most significant bit of A.

3.3 Octet Sequences

This mechanism generates, uses and exchanges sequences of octets; e.g. output values of message digest algorithm functions. When such entities travel on the wire, they shall be preceded by a one-octet scalar quantity representing the count of following octets.

Note that a zero-length octet sequence is encoded as a single 00 octet.

We shall use the syntax os(s) to denote the encoded form of the octet sequence. Furthermore, we shall use the syntax bytes(s) to denote the sequence of octets s, in big-endian order.

3.4 Extended Octet Sequences

Extended sequences of octets are exchanged when using the security layer. When these sequences travel on the wire, they shall be preceded by a four-octet scalar quantity representing the count of following octets.

We shall use the syntax eos(s) to denote the encoded form of the extended octet sequence. Furthermore, we shall use the syntax bytes(s) to denote the sequence of octets s, in big-endian order.

3.5 Text

The only character set for text is the UTF-8 encoding [RFC-2279] of Unicode characters [ISO-10646]. All text MUST be in Unicode Normalization Form KC [UNICODE-KC] without NUL characters.

In addition, to avoid non-interoperability due to incompatible normalisation techniques, the client MUST prepare strings using the [SASLprep] profile of [RFC-3454]

We shall use the syntax utf8(L) to denote the string L in UTF-8 encoding, preceded by a two-octet scalar quantity representing the count of following octets. Furthermore, we shall use the syntax bytes(L) to denote the sequence of octets representing the UTF-8 encoding of L, in big-endian order.

Not that the empty string is encoded as the two octet sequence 00 00.

3.6 Buffers

In this mechanism data is exchanged between the client and server using buffers. A buffer acts as an envelope for the sequence of data elements sent by one end-point of the exchange, and expected by the other.

A buffer MAY NOT contain other buffers. It may only contain zero, one or more data elements.

A buffer shall be encoded as two fields: a four-octet scalar quantity representing the count of following octets, and the concatenation of the octets of the data element(s) contained in the buffer.

We shall use the syntax {A|B|C} to denote a buffer containing A, B and C in that order. For example:

```
{ mpi(N) | mpi(g) | utf8(L) }
```

is a buffer containing, in the designated order, the encoded forms of an MPI N, an MPI g and a Text L.

3.7 Data Element Size Limits

The following table details the size limit, in number of octets, for each of the data element encodings described earlier.

Data element type	Header (octets)	Size limit in octets (excluding header)
Octet Sequence	1	255
MPI	2	65,535
Text	2	65,535
Extended Octet Sequence	4	2,147,483,383
Buffer	4	2,147,483,643

An implementation MUST signal an exception if any size constraint is violated.

3.8 Unsigned Integers

This mechanism uses unsigned integer values ranging from zero to 4,294,967,296.

When such entities travel on the wire, they shall be encoded as 4-octet Scalar Numbers. We shall use the syntax uint(n) to denote the encoding of an Unsigned Integer n.

4. Protocol Description

The following sections describe the sequence of data transmitted between the client and server for SRP authentication, as well as the extra control information exchanged to enable a client to request whether or not replay detection, integrity protection and/or confidentiality protection should be provided by a security layer. There are two possible mechanism data exchanges during the authentication phase:

The following exchange occurs when a new session is negotiated between the client and the server. It will also occur when the client requests re-use of the parameters of a previous session and either the server does not support such re-use or no longer considers the previous session to be valid:

```
 Client                                                        Server

 ---  { utf8(U) | utf8(I) | utf8(sid) | os(cn) }  ------------->

 <------ { 00 | mpi(N) | mpi(g) | os(s) | mpi(B) | utf8(L) } ---

 ---  { mpi(A) | os(M1) | utf8(o) | os(cIV) }  ---------------->

 <------ { os(M2) | os(sIV) | utf8(sid) | uint(ttl) }  ---------
```

where:

U   is the authentication identity (username),

I   is the authorisation identity (userid),

sid is the identifier of a previous session whose parameters the client wishes to re-use,

cn is the client's nonce used in deriving a new shared context key from the shared context key of the previous session,

00 is an octet indicating that the previous session parameters will NOT be re-used,

N   is the safe prime modulus,

g   is the generator,

s   is the user's password salt,

B   is the server's ephemeral public key,

L   is the options list indicating available security services,

A   is the client's ephemeral public key,

M1 is the client's evidence that the shared key K is known,

o   is the options list indicating chosen security services,

cIV is the client's initial vector for the chosen encryption algorithm,

M2 is the server's evidence that the shared key K is known.

sIV is the server's initial vector for the chosen encryption algorithm,

sid is the identifier the server gives to this session for possible later re-use of the negotiated parameters,

ttl is the time period for which this session's parameters may be re-usable,

The following exchange occurs when the client requests that the parameters negotiated in a previous session be re-used in this session, but with a newly derived shared context key, and the server agrees:

```
 Client                                                  Server

 ---  { utf8(U) | utf8(I) | utf8(sid) | os(cn) }  -------------->

 <-----------------------------------  { FF | os(sn) }  ----------
```

where:

U   is the authentication identity (username),

I   is the authorisation identity (userid),

sid is the identifier of a previous session whose parameters the client wishes to re-use,

cn is the client's nonce used in deriving a new shared context key from the shared context key of the previous session,

FF is an octet indicating that the previous session parameters WILL be re-used,

sn is the server's nonce used in deriving a new shared context key from the shared context key of the previous session,

4.1 Client Sends its Identity

The client determines its authentication identity U and authorisation identity I, encodes them and sends them to the server.

The semantics of both U and I are intended to be the same as described in [SASL]. Specifically, the authentication identity U is derived from the client's authentication credentials, and the authorisation identity I is used by the server as the primary identity for making access policy decisions.

As a client might not have the same information as the server, clients SHOULD NOT themselves try to derive authorisation identities from authentication identities. When an authorisation identity is not specified by the user the client SHOULD send an empty string instead.

If the client does not wish to re-use parameters negotiated in a previous session then it sets sid to the empty string and cn to a zero-length octet sequence.

However, if the client does wish to attempt to re-use the parameters negotiated in a previous session then it sets sid to the session identifier for that session, and sets cn as follows:

```
cn = prng()
```

where:

prng() is a random number generation function that outputs at least 16 octets of data.

See Section 6.4 for more information on re-using negotiated parameters of a previous session.

The client sends:

```
{ utf8(U) | utf8(I) | utf8(sid) | os(cn) }
```

4.2 Server Agrees to Re-use Parameters of a Previous Session

If the server supports re-using the parameters negotiated in a previous session and it considers the previous session, identified by

the session identifier (sid) received from the client, to be valid, it responds as follows:

The server sends the octet FF as the first element of the message to indicate to the client that parameters of the previous session are being re-used. It also generates a nonce (sn), which is later used in deriving a new shared context key for this session:

```
sn = prng()
```

where:

prng() is a random number generation function that outputs at least 16 octets of data.

Note that the server nonce (sn) MUST NOT be the same as the client nonce (cn).

The server sends:

```
{ FF | os(sn) }
```

See Section 6.4 for more information on re-using negotiated parameters of a previous session and deriving the new shared context key.

4.3 Server Sends Protocol Elements

Otherwise, the server receives U and looks up the safe prime modulus N, the generator g, the salt s, and the verifier v, to be used for that identity. It uses the this information to generate its ephemeral public key B as follows:

```
b = prng();

B = ((3 * v) + (g ** b)) % N;
```

where:

prng() is a random number generation function,

b   is the MPI that will act as the server's private key,

v   is the stored password verifier value,

g   is the generator,

N   is the safe prime modulus,

*   is the multiplication operator,

+   is the addition operator,

**   is the exponentiation operator,

%   is the modulus operator,

The server also creates an options list L, which consists of a comma-separated list of option strings that specify the options the server supports. This options list MUST NOT contain any whitespace characters and all alphabetic characters MUST be in lowercase. When used in digest calculations by the client the options list MUST be used as received.

The following option strings are defined:

o "mda=<MDA-name>" indicates that the server supports the designated hash function as the underlying Message Digest Algorithm for the designated user to be used for all SRP calculations - to compute both client-side and server-side digests. The specified algorithm MUST meet the requirements specified in section 3.2 of [RFC-2945]:

  "Any hash function used with SRP should produce an output of at least 16 bytes and have the property that small changes in the input cause significant nonlinear changes in the output."

  Note that in the interests of interoperability between client and server implementations and with other SRP-based tools, both the client and the server MUST support SHA-160 as an underlying Message Digest Algorithm. While the server is not required to list SHA-160 as an available underlying Message Digest Algorithm, it must be able to do so.

o "integrity=hmac-<MDA-name>" indicates that the server supports integrity protection using the HMAC algorithm [RFC-2104] with <MDA-name> as the underlying Message Digest Algorithm. Acceptable MDA names are chosen from [SCAN] under the MessageDigest section. A server SHOULD send such an option string for each HMAC algorithm it supports. The server MUST advertise at least one integrity protection algorithm and in the interest of interoperability the server SHOULD advertise support for the HMAC-SHA-160 algorithm.

o "replay_detection" indicates that the server supports replay detection using sequence numbers. Replay detection SHALL NOT be activated without also activating integrity protection. If the replay detection option is offered (by the server) and/or chosen (by the client) without explicitly specifying an integrity

protection option, then the default integrity protection option "integrity=hmac-sha-160" is implied and SHALL be activated.

- "confidentiality=<cipher-name>" indicates that the server supports confidentiality protection using the symmetric key block cipher algorithm <cipher-name>. The server SHOULD send such an option string for each confidentiality protection algorithm it supports. Note that in the interest of interoperability, if the server offers confidentiality protection, it MUST send the option string "confidentiality=aes" since it is then MANDATORY for it to provide support for the [AES] algorithm.

- "mandatory=[integrity|replay_detection|confidentiality]" is an option only available to the server that indicates that the specified security layer option is MANDATORY and MUST be chosen by the client for use in the resulting security layer. If a server specifies an option as mandatory in this way, it MUST abort the connection if the specified option is not chosen by the client. It doesn't make sense for the client to send this option since it is only able to choose options that the server advertises. The client SHOULD abort the connection if the server does not offer an option that it requires. If this option is not specified then this implies that no options are mandatory. The server SHOULD always send the "mandatory=integrity" option indicating that integrity protection is required.

- "maxbuffersize=<number-of-bytes>" indicates to the peer the maximum number of raw bytes (excluding the buffer header) to be processed by the security layer at a time, if one is negotiated. The value of <number-of-bytes> MUST NOT exceed the Buffer size limit defined in section 3.7. If this option is not detected by a client or server mechanism, then it shall operate its security layer on the assumption that the maximum number of bytes that may be sent, to the peer server or client mechanism respectively, is the Buffer data size limit indicated in section 3.7. On the other hand, if a recipient detects this option, it shall break any octet-sequence longer than the designated limit into two or more fragments, before sending them separately, in sequence, to the peer.

For example, if the server supports integrity protection using the HMAC-SHA-160 and HMAC-MD5 algorithms, replay detection and no confidentiality protection, the options list would be:

```
mda=sha-1,integrity=hmac-sha-160,integrity=hmac-md5,replay_detection
```

The server sends the octet 00 as the first element of the message to indicate to the client that parameters from a previous session are

NOT being used.

The server sends:

```
{ 00 | mpi(N) | mpi(g) | os(s) | mpi(B) | utf8(L) }
```

4.4 Client Sends its Ephemeral Public Key and Evidence

The client receives the options list L from the server that specifies the Message Digest Algorithm(s) available to be used for all SRP calculations, the security service options the server supports, including the maximum buffer size the server can handle, and the server's ephemeral public key B. The client selects options from this list and creates a new options list o that specifies the selected Message Digest Algorithm to be used for SRP calculations and the security services that will be used in the security layer. At most one available Message Digest Algorithm name, one available integrity protection algorithm and one available confidentiality protection algorithm may be selected. In addition the client may specify the maximum buffer size it can handle. The client MUST include any option specified by the mandatory option.

The client SHOULD always select an integrity protection algorithm even if the server does not make it mandatory to do so. If the client selects a confidentiality protection algorithm it SHOULD then also select an integrity protection algorithm.

The options list o MUST NOT contain any whitespace characters and all alphabetic characters MUST be in lowercase. When used in digest calculations by the server the options list MUST be used as received.

The client generates its ephemeral public key A as follows:

```
a = prng();

A = (g ** a) % N;
```

where:

```
a      is the MPI that will act as the client's private key,
```

The client also calculates the shared context key K, and calculates the evidence M1 that proves to the server that it knows the shared context key K, as well as the server's ephemeral public key B, the user's authorisation identity I and the server's options list L.

K, on the client's side is computed as follows:

```
      x = H(s | H(U | ":" | p));

      u = H(A | B);

      S = ((B - (3 * (g ** x))) ** (a + (u * x))) % N;

      K = H(S);
```

where:

```
      s    is the user's password salt,

      U    is the authentication identity (username),

      p    is the password value.

      A    is the client's ephemeral public key,

      B    is the server's ephemeral public key,

      g    is the generator,

      N    is the safe prime modulus,

      H()  is the result of digesting the designated input/data with the
      chosen underlying Message Digest Algorithm function.

      -    is the subtraction operator,

      *    is the multiplication operator,

      +    is the addition operator,

      **   is the exponentiation operator,

      %    is the modulus operator,
```

M1 is computed as:

```
            H(   bytes(H( bytes(N) )) ^ bytes( H( bytes(g) ))
               | bytes(H( bytes(U) ))
               | bytes(s)
               | bytes(A)
               | bytes(B)
               | bytes(K)
               | bytes(H( bytes(I) ))
               | bytes(H( bytes(L) ))
             )
```

where:

^   is the bitwise XOR operator.

All parameters received from the server that are used as input to a digest operation MUST be used as received.

If the client chooses to activate the Confidentiality Protection service in the Security Layer, it MUST send the Initial Vector cIV that the server will use to set up its encryption context. (See Section 5.2 for details on the Confidentiality Protection service and how cIV is generated.) However, this element MAY be a zero-length octet stream if the server does not advertise the Confidentiality Protection service or the client decides not to activate it.

The client sends:

```
{ mpi(A) | os(M1) | utf8(o) | os(cIV) }
```

4.5 Server Verifies Client's Evidence and Sends its Evidence

The server calculates the shared context key K, and verifies the client's evidence M1.

K, on the server's side is computed as follows:

```
u = H(A | B);

S = ((A * (v ** u)) ** b) % N;

K = H(S);
```

where:

A   is the client's ephemeral public key,

B   is the server's ephemeral public key,

v   is the stored password verifier value,

b   is the server's ephemeral private key,

N   is the safe prime modulus,

H() is the result of digesting the designated input/data with the chosen underlying Message Digest Algorithm function.

```
*    is the multiplication operator,

**   is the exponentiation operator,

%    is the modulus operator,
```

If the client chose to activate the Confidentiality Protection service in the Security Layer then the server MUST send the Initial Vector sIV that the client will use to set up its encryption context. (See Section 5.2 for details on the Confidentiality Protection service and how sIV is generated.) However, this element MAY be a zero-length octet sequence if the client did not choose to activate the Confidentiality Protection service.

If the server's policy allows re-using the parameters of this session then it sets sid to a unique identifier for this session and sets ttl to the number of seconds for which the session MAY be valid. If the server does not support re-using the parameters of this session then it sets sid to the empty string and ttl to any value. See Section 6.4 for more information on re-using negotiated parameters of a previous session.

The server computes its evidence M2, which proves to the client that it knows the shared context key K, as well as U, I and o, as follows:

```
H(   bytes(A)
   | bytes(M1)
   | bytes(K)
   | bytes(H( bytes(I) ))
   | bytes(H( bytes(o) ))
   | bytes(sid)
   | ttl
)
```

All parameters received from the client that are used as input to a digest operation MUST be used as received.

The server sends:

```
{ os(M2) | os(sIV) | sid | ttl }
```

5. Security Layer

Depending on the options offered by the server and chosen by the client, the security layer may provide integrity protection, replay detection, and/or confidentiality protection.

The security layer can be thought of as a three-stage filter through which the data flows from the output of one stage to the input of the following one. The first input is the original data, while the last output is the data after being subject to the transformations of this filter.

The data always passes through this three-stage filter, though any of the stages may be inactive. Only when a stage is active would the output be different from the input. In other words, if a stage is inactive, the octet sequence at the output side is an exact duplicate of the same sequence at the input side.

Schematically, the three-stage filter security layer appears as follows:

```
              +----------------------------+
              |                            |     I/ p1
    p1  --->| Confidentiality protection |---+
              |                            |   | A/ c
              +----------------------------+   |
                                               |
          +------------------------------------+
          |
          |   +----------------------------+
          |   |                            |     I/ p2
    p2  +-->|      Replay detection      |---+
              |                            |   | A/ p2 | q
              +----------------------------+   |
                                               |
          +------------------------------------+
          |
          |   +----------------------------+
          |   |                            |     I/ p3
    p3  +-->|    Integrity protection    |--->
              |                            |     A/ p3 | C
              +----------------------------+
```

where:

p1, p2 and p3 are the input octet sequences at each stage,

I/ denotes the output at the end of one stage if/when the stage is

inactive or disabled,

A/ denotes the output at the end of one stage if/when the stage is active or enabled,

c is the encrypted (sender-side) or decrypted (receiver-side) octet sequence. c1 shall denote the value computed by the sender, while c2 shall denote the value computed by the receiver.

q is a four-octet scalar quantity representing a sequence number,

C is the Message Authentication Code. C1 shall denote the value of the MAC as computed by the sender, while C2 shall denote the value computed by the receiver.

It is worth noting here that both client and server have their own distinct security contexts, including distinct encryption and decryption sub-contexts. In principal, nothing in this specification should prevent an implementation from supporting asynchronous connections.

5.1 Cryptographic Primitives

5.1.1 Pseudo Random Number Generator

This mechanism requires random data to be generated for use in:

1. The CALG key material for both the client and server when the Confidentiality Protection service is enabled.

2. The IALG key material for both the client and server when the Integrity Protection service is enabled.

The PRNG used in this specification is based on the pseudo-random function described in section 5 of [UMAC]. It uses the [AES] algorithm, in its 128-bit key size variant, as the underlying symmetric key block cipher for its operations.

A formal description of this PRNG follows:

- o Initialisation

    * SK: a 16-octet sequence (seeding key to AES)

- o Input

    * n: a positive integer

- o Output

  * Y: an n-octet sequence

- o Algorithm

  * (initialisation)

    1. Initialise an AES instance for encryption with the first 16 octets of SK as its user-supplied key material. Let "aes" be that instance; i.e. aes = AES(SK, ENCRYPTION);

    2. Initialise T to be an all-zero 16-octet long sequence;

  * (for every input)

    1. Initialise "remaining" to n;

    2. Initialise Y to be a 0-length octet sequence;

    3. while (remaining > 0) do

       1. T = aes(T);

       2. Append m octets from T to Y, where m is the minimum of 16 and remaining;

       3. Subtract 16 from remaining;

    4. return Y;

In this document, "PRNG(key,n)" will refer to this algorithm, with the initialisation parameter SK set to be the octets of the specified key, returning n bits of pseudo-random data. For example, "PRNG(K,n)" will refer to this algorithm, with the initialisation parameters SK set to the shared context key K computed by the SRP calculations (see Section 4.4 and Section 4.5), returning n bits of pseudo-random data.

This algorithm MAY also be used as part of the SRP calculations to generate the required "a" and "b" parameters used in creating the client and server ephemeral private keys ("A" and "B"), or to generate the cn and sn parameters used in session re-use, or to generate the initial vectors sIV and cIV used to set up the encryption contexts. In this case the initialisation parameter SK can be any 16-octet sequence (e.g. multiple representations of the time-of-day).

If the same PRNG instance is used for both these calculations and the calculations in this specification, it MUST be re-initialised with the shared context key K before any of the latter calculations are performed.

5.1.2 Key Derivation Function

During the authentication phase, both parties compute the shared context key K (see Section 4.4 for the client, and Section 4.5 for the server sides respectively). The length of K is s bits, where "s" is the output length of the chosen underlying Message Digest Algorithm used in the SRP calculations (see "mda" option in Section 4.3).

When Confidentiality Protection is required, and the length of K is not equal to the length of the user-supplied key material needed to initialise the chosen Confidentiality Algorithm (CALG), the peers MUST apply the Key Derivation Function (KDF) in order to obtain enough data for this purpose.

Similarly, when Integrity Protection is required, and the length of K is not equal to the required length of the key material needed to initialise the chosen Integrity Algorithm (IALG), the peers MUST apply the Key Derivation Function (KDF) in order to obtain enough data for this purpose too.

If the KDF is required for both the key used with the CALG and the key used with the IALG then it is first applied for the CALG key and thereafter for the IALG key.

We define this KDF as:

```
   Km = KDF(n)
```

where:

```
   Km  is the required key material,

   K   is the shared context key, and

   n   is the required length of Km.
```

The following steps describe the KDF algorithm:

```
   If length of K is greater than or equal to n, then

      Let Km be the first n bytes of K;
```

```
Else

   Let Km = PRNG(K, n);

return Km
```

5.2 Confidentiality Protection

The plaintext data octet sequence p1 is encrypted using the chosen confidentiality algorithm (CALG) with key size m, initialised for encryption with the key material Kc obtained as follows:

```
Kc = KDF(m)

c1 = CALG(Kc, ENCRYPTION)( bytes(p1) )
```

On the receiving side, the ciphertext data octet sequence p1 is decrypted using the chosen confidentiality algorithm (CALG) initialised for decryption, with the key Kc obtained by a similar process:

```
Kc = KDF(m)

c2 = CALG(Kc, DECRYPTION)( bytes(p1) )
```

The designated CALG symmetric-key block cipher MUST be used in OFB (Output Feedback Block) mode in the ISO variant, as described in [HAC], algorithm 7.20.

Let k be the block size of the chosen symmetric key block cipher algorithm; e.g. for AES this is 128 bits or 16 octets. The OFB mode used shall have a block size of k.

It is recommended that block ciphers operating in OFB mode be used with an Initial Vector (the mode's IV). In such a mode of operation - OFB with key re-use - the IV need not be secret. For the mechanism described in this document, the server MUST use cIV received from the client as the Initial Vector when initialising its encryption context, and the client MUST use sIV received from the server as the Initial Vector when initialising its encryption context. These Initial Vectors are generated as:

```
cIV = prng(k);

sIV = prng(k);
```

where:

prng() is a random number generation function that outputs k octets of data,

k is the block size of the chosen symmetric key block cipher algorithm

The input data to the confidentiality protection algorithm shall be a multiple of the symmetric key block cipher block size k. When the input length is not a multiple of k octets, the data shall be padded according to the following scheme (described in [PKCS7] which itself is based on [RFC-1423]):

Assuming the length of the input is l octets, (k - (l mod k)) octets, all having the value (k - (l mod k)), shall be appended to the original data. In other words, the input is padded at the trailing end with one of the following sequences:

```
          01 -- if l mod k = k-1
         02 02 -- if l mod k = k-2
                   ...
                   ...
                   ...
       k k ... k k -- if l mod k = 0
```

The padding can be removed unambiguously since all input is padded and no padding sequence is a suffix of another. This padding method is well-defined if and only if k < 256 octets, which is the case with symmetric block ciphers today, and in the forseeable future.

The output of this stage, when it is active, is:

at the sending side: CALG(Kc, ENCRYPT)(bytes(p1))

at the receiving side: CALG(Kc, DECRYPT)(bytes(p1))

5.3 Replay Detection

A sequence number q is incremented every time a message is sent to the peer.

The output of this stage, when it is active, is:

p2 | q

At the other end, the receiver increments its instance of the sequence number. This new value of the sequence number is then used in the integrity protection transformation, which must also be active as described in Section 4.3. See Section 6.3 for more details.

5.4 Integrity Protection

When the Integrity Protection stage is active, a message authentication code C is computed using the chosen integrity protection algorithm (IALG) as follows:

o the IALG is initialised (once) with the key material Ki of size n (the required key size of the chosen IALG); i.e. Ki = KDF(n),

o the IALG is updated with every exchange of the sequence p3, yielding the value C and a new IALG context for use in the following exchange.

At the other end, the receiver computes its version of C, using the same transformation, and checks that its value is equal to that received. If the two values do not agree, the receiver MUST signal an exception and abort.

The output of this stage, when it is active, is then:

```
IALG(Ki)( bytes(p3) )
```

5.5 Summary of Security Layer Output

The following table shows the data exchanged by the security layer peers, depending on the possible legal combinations of the three security services in operation:

CP	IP	RD	Peer sends/receives
I	I	I	{ eos(p) }
I	A	I	{ eos(p) | os(IALG(Ki)(bytes(p))) }
I	A	A	{ eos(p) | os(IALG(Ki)(bytes(p) | bytes(q))) }
A	I	I	{ eos(c) }
A	A	I	{ eos(c) | os(IALG(Ki)(bytes(c))) }
A	A	A	{ eos(c) | os(IALG(Ki)((bytes(c) | bytes(q)))}

where

CP   Confidentiality protection,

IP   Integrity protection,

```
      RD    Replay detection,

      I     Security service is Inactive/disabled,

      A     Security service is Active/enabled,

      p     The original plaintext,

      q     The sequence number.

      c     The enciphered input obtained by either:

         CALG(Kc, ENCRYPT)( bytes(p) ) at the sender's side, or

         CALG(Kc, DECRYPT)( bytes(p) ) at the receiver's side
```

6. Discussion

6.1 Mandatory Algorithms

The algorithms specified as mandatory were chosen for utility and availablity. We felt that a mandatory confidentiality and integrity protection algorithm for the security layer and a mandatory Message Digest Algorithm for SRP calculations should be specified to ensure interoperability between implementations of this mechanism:

- o The SHA-160 Message Digest Algorithm was chosen as an underlying algorithm for SRP calculations because this allows for easy interoperability with other SRP-based tools that use the SRP-SHA1 protocol described in section 3 of [RFC-2945] and create their password files using this algorithm.

- o The HMAC algorithm was chosen as an integrity algorithm because it is faster than MAC algorithms based on secret key encryption algorithms [RFC-2847].

- o AES was chosen as a symmetric-key block cipher because it has undergone thorough scrutiny by the best cryptographers in the world.

Since confidentiality protection is optional, this mechanism should be usable in countries that have strict controls on the use of cryptography.

6.2 Modulus and Generator Values

It is RECOMMENDED that the server use values for the modulus N and generator g chosen from those listed in Appendix A so that the client can avoid expensive constraint checks, since these predefined values already meet the constraints described in [RFC-2945]:

> "For maximum security, N should be a safe prime (i.e. a number of the form N = 2q + 1, where q is also prime). Also, g should be a generator modulo N (see [SRP] for details), which means that for any X where 0 < X < N, there exists a value x for which g**x == X (mod N).

If other values are used for N and g then these values SHOULD undergo the specified constraint checks.

6.3 Replay Detection Sequence Number Counters

The mechanism described in this document allows the use of a Replay Detection security service that works by including sequence number

counters in the message authentication code (MAC) created by the Integrity Protection service. As noted in Section 4.3 integrity protection is always activated when the Replay Detection service is activated.

Both the client and the server keep two sequence number counters. Each of these counters is a 32-bit unsigned integer initialised with a Starting Value and incremented by an Increment Value with every successful transmission of a data buffer through the security layer. The Sent counter is incremented for each buffer sent through the security layer. The Received counter is incremented for each buffer received through the security layer. If the value of a sequence number counter exceeds 2**32-1 it wraps around and starts from zero again.

When a sender sends a buffer it includes the value of its Sent counter in the computation of the MAC accompanying each integrity protected message. When a recipient receives a buffer it uses the value of it's Received counter in its computation of the integrity protection MAC for the received message. The recipient's Received counter must be the same as the sender's Sent counter in order for the received and computed MACs to match.

This specification assumes that for each sequence number counter the Starting Value is ZERO, and that the Increment Value is ONE. These values do not affect the security or the intended objective of the replay detection service, since they never travel on the wire.

6.4 Re-using the Parameters of a Previous Session

Re-using the parameters of a previous session enables the client and server to avoid the overhead of the full authentication exchange where the client and server communicate more than once during a server-specified time period.

Servers are not required to support re-using the parameters of the current session in future sessions. If they do not wish to support this then they send an empty string for the session identifier (sid). However, if the server's policy allows for the parameters of the current session to be re-used later, it generates a session identifier (sid) that will uniquely identify the session within the specified time period (ttl). The time period (ttl) is specified in seconds and only gives an indication to the client how long the session may be valid. The server is not required to ensure that the session is valid for this time period. Note that a ttl of 0 indicates an indeterminate time period.

To avoid session hijacking, servers SHOULD NOT indicate that a

session may be re-used unless a security layer with integrity protection and/or confidentiality protection has been negotiated.

Clients are not required to support re-using the parameters of previous sessions. If they do not wish to support it or they do not wish to re-use the parameters of a previous session then they send the empty string as the value for the session identifier (sid) and send a zero-length octet sequence for the nonce (cn). If they do support it and wish to use the parameters of a previous session then they send the session identifier for this session that they previously received from the server and calculate cn as described in Section 4.1.

If a client specifies a session id (sid) for a session that the server still considers valid then the server sends the octet FF, to indicate to the client that parameters of a previous session are being re-used, and the nonce (sn) calculated as described in Section 4.2. The client and server then calculate the new shared context key Kn for this session as follows:

    Kn = H(K | cn | sn)

where:

- K is the shared context key for the previous session identified by sid.

- H() is the result of digesting the designated input/data with the Message Digest Algorithm function negotiated in the previous session identified by sid.

Then, if the confidentiality and/or integrity protection services were negotiated for the previous session, new keys for these services are derived using the KDF for use in this session. (See Section 5.1.2.)

If the server does not support re-using parameters of previous sessions or no longer considers the specified previous session to be valid, it ignores the session id specified by the client and continues the full authentication exchange. However, the first element of the next buffer it sends is the octet 00, which indicates to the client that no parameters of a previous session will be re-used.

7. SASL

7.1 Overview

SASL is described as follows [RFC-2222]:

> The Simple Authentication and Security Layer (SASL) is a method for adding authentication support to connection-based protocols.

This document describes a mechanism that can be used within the SASL authentication framework.

7.2 Mechanism Name

The SASL mechanism name associated with this protocol is "SRP".

7.3 Security Layer

Section 3 of [RFC-2222] describes the operation of the security layer as follows:

> "The security layer takes effect immediately following the last response of the authentication exchange for data sent by the client and the completion indication for data sent by the server. Once the security layer is in effect, the protocol stream is processed by the security layer into buffers of cipher-text. Each buffer is transferred over the connection as a stream of octets prepended with a four octet field in network byte order that represents the length of the following buffer. The length of the cipher-text buffer must be no larger than the maximum size that was defined or negotiated by the other side."

7.4 Profile Considerations

As mentioned briefly in [RFC-2222], and detailed in [SASL] a SASL specification has three layers: (a) a protocol definition using SASL known as the "Profile", (b) a SASL mechanism definition, and (c) the SASL framework.

Point (3) in section 5 of [SASL] ("Protocol profile requirements") clearly states that it is the responsibility of the Profile to define "...how the challenges and responses are encoded, how the server indicates completion or failure of the exchange, how the client aborts an exchange, and how the exchange method interacts with any line length limits in the protocol."

The username entity, referenced as U throughout this document, and

used by the server to locate the password data, is assumed to travel "in the clear," meaning that no transformation is applied to its contents. This assumption was made to allow the same SRP password files to be used in this mechanism, as those used with other SRP applications and tools.

A Profile may decide, for privacy or other reason, to disallow such information to travel in the clear, and instead use a hashed version of U, or more generally a transformation function applied to U; i.e. f(U). Such a Profile would require additional tools to add the required entries to the SRP password files for the new value(s) of f(U). It is worth noting too that if this is the case, and the same user shall access the server through this mechanism as well as through other SRP tools, then at least two entries, one with U and the other with f(U) need to be present in the SRP password files if those same files are to be used for both types of access.

7.5 Example

The example below uses SMTP authentication [RFC-2554]. The base64 encoding of challenges and responses, as well as the reply codes preceding the responses are part of the SMTP authentication specification, not part of this SASL mechanism itself.

"C:" and "S:" indicate lines sent by the client and server respectively.

```
S: 220 smtp.example.com ESMTP server ready

C: EHLO zaau.example.com

S: 250-smtp.example.com
S: 250 AUTH SRP CRAM-MD5 DIGEST-MD5

C: AUTH SRP AAAADAAEdGVzdAAEdGVzdA==

 with:

   U = "test"

   I = "test"

S: 334 AAABygEArGvbQTJKmpvxZt5eE4lYL69ytmUZh+4H/DGSlD21YFCjcynLtKCZ
7YGT4HV3Z6E91SMSq0sDMQ3Nf0ip2gT9UOgIOWntt2ewz2CVF5oWOrNmGgX71fqq6Ck
YqZYvC5O4Vfl5k+yXXuqoDXQK2/T/dHNZ0EHVwz6nHSgeRGsUdzvKl7Q6I/uAFna9IH
pDbGSB8dK5B4cXRhpbnTLmiPh3SFRFI7UksNV9Xqd6J3XS7PoDLPvb9S+zeGFgJ5AE5
Xrmr4dOcwPOUymczAQce8MI2CpWmPOo0MOCca41+Onb+7aUtcgD2J965DXeI21SX1R1
m2XjcvzWjvIPpxEfnkr/cwABAgqsi3AvmIqdEbREALhtZGE9U0hBLTEsbWFuZGF0b3J
```

5PXJlcGxheSBkZXRlY3Rpb24scmVwbGF5IGRldGVjdGlvbixpbnRlZ3JpdHk9aG1hYy
1zaGExLGludGVncml0eT1obWFjLW1kNSxjb25maWRlbnRpYWxpdHk9YWVzLGNvbmZpZ
GVudGlhbGl0eT1jYXN0NSxjb25maWRlbnRpYWxpdHk9Ymxvd2Zpc2gsbWF4YnVmZmVy
c2l6ZT0yMTQ3NDgzNjQz

with:

N = "21766174458617435773191008891802753781907668374255538511144
643224689886235383840957210909013086056401571399717235807266581 6
496064721484102914133641521973644771808873956554837381150726774 0
223510176252190156982074029314952962041933326626207347105454836 8
736039519702486226506248861060256971802984953561121442680157668 0
007614299882224570904138739739701719270939921147517651680636147 6
111961547623342209644278311797123637164733387141433589577347466 7
308967050807005509320424799678417036867928316761272274230314067 5
482911335824795830614395775593471019617714061736843785227034834 9
53370376550067513284475105502992509244692888 19"

g = "2"

s = "814819216327401865851972"

L = "mda=sha-1,mandatory=replay_detection,replay_detection,integ
rity=hmac-sha1,integrity=hmac-md5,confidentiality=aes,confidenti
ality=cast5,confidentiality=blowfish,maxbuffersize=2147483643"

C: AAABYwEAAp5q/4zhXoTUzXBscozN97SWgfDcAImIk3lNHNvd0b+Dr7jEm6upXblZ
T5sL9mPgFsejlIh+B/eCu/HvzWCrXj6ylPZv8dy3LCH3LIORqQ45S7Lsbmrrg/dukDh
4tZCJMLD4r3evzaY8KVhtJeLMVbeXuh4JljKP42Ll59Lzwf8jfPh4+4Lae1rpWUCL9D
ueKcY+nN+xNHTit/ynLATxwL93P6+GoGY4TkUbUBfjiI1+rAMvyMDMw5XozGy07FOEc
++U0iPeXCQP4MT5FipOUoz8CYX7J1LbaXp2WJuFHlkyVXF7oCoyHbhld/5CfR3o6q/B
/x9+yZRqaHH+JfllOgBfbWRhPVNIQS0xLHJlcGxheSBkZXRlY3Rpb24saW50ZWdyaXR
5PWhtYWMtbWQ1LGNvbmZpZGVudGlhbGl0eT1ibG93ZmlzaCxtYXhidWZmZXJzaXplPT
IxNDc0ODM2NDM=

with:

A = "33059541846712102497463123211304342021934496372587869281515
969565823777988446277747885039497774455374693045189581561588840 5
056278070737087825375397936701907714288223702976616662327571822 7
655538983419084032208109159908908194732453790761392470705815003 7
780279077623179396214378641179251676003010243660362104654172939 6
689061339437990052741200706824255929942287289333211136584053649 5
185883474232883537338757318836995637988160638089067541196607366 5
110692200229403553347030154199927455720066670338953148179451662 5
475741844221598063493387653318996956261324149946529584983299909 1
40398081321840949606581251320320995783959866"

```
   o = mda=sha-1,replay_detection,integrity=hmac-md5,confidentialit
   y=blowfish,maxbuffersize=2147483643"
```

S: 334 AAABAgEAOUKbXpnzMhziivGgMwm+FS8sKGSvjh5M3D+80RF/5z9rm0oPoi4+
pF83fueWn4Hz9M+muF/22PHHZkHtlutDrtapj4OtirdxC21fS9bMtEh3F0whTX+3mPv
thw5sk11turandHiLvcUZOgcrAGIoDKcBPoGyBud+8bMgpkf/uGfyBM2nEX/hV+oGgg
X+LiHjmkxAJ3kewfQPH0eV9ffEuuyu8BUcBXkJsS6l7eWkuERSCttVOi/jS031c+CD/
nuecUXYiF8IYzW03rbcwYhZzifmTi3VK9C8zG2K1WmGU+cDKlZMkyCPMmtCsxlbgE8z
SHCuCiOgQ35XhcA0Qa0C3Q==

with:

```
   B: "722842847565031844205403087285424428589273458129750231766015
   4465607827529853239240118185263492617243523916106658696965596526
   8585300845435562962039149169549800169184521786717633959469278439
   8771344445002432579509292115598435685062882631760796416554562980
   8475896198325835507901319556929511421472132184990365213059654962
   7218189966140113906545856088040473723048909402258929560823932725
   2022154114087913895411927676707073040281136096806681758265221209
   8822374723416364340410020172215773934302794679034424699999611678
   9730443114919539575466941344964841591072763617954717789621871251
   71089179399349194452686682517183909017223901"
```

C: AAAAFRTkoju6xGP+zH89iaDWIFjfIKt5Kg==

S: 235 Authentication successful.

8. GSS-API

8.1 Overview

The GSS-API is described as follows:

> The Generic Security Service Application Program Interface (GSS-API), Version 2, as defined in [RFC-2078], provides security services to callers in a generic fashion, supportable with a range of underlying mechanisms and technologies and hence allowing source-level portability of applications to different environments.

According to [RFC-2078] there are certain specifications related to the GSS-API that are:

> "documents defining token formats, protocols, and procedures to be implemented in order to realize GSS-API services atop particular security mechanisms"

This specification is such a document - it defines a security mechanism that can be used with the GSS-API authentication framework.

8.2 Terminology

The tokens referred to in the GSS-API specification [RFC-2078] are the same as the buffers referred to in this document.

8.3 Initial Token

[RFC-2078] states that:

> The first context-level token obtained from GSS_Init_sec_context() is required to indicate at its very beginning a globally-interpretable mechanism identifier, i.e., an Object Identifier (OID) of the security mechanism. The remaining part of this token as well as the whole content of all other tokens are specific to the particular underlying mechanism used to support the GSS-API.

To satisfy this requirement and make use of the mechanism described in this document as a GSS-API mechanism, the following octets must be prefixed to the first buffer sent as part of the protocol described in Section 4:

```
[ 60 08 06 06 2B 06 01 05 05 08 ]
```

Each octet is written as a pair of hex digits - see Section 2.

These octets represent the encoding of the GSS-API mechanism identifier as per section 3.1 of [RFC-2078]. The OID for this mechanism is iso.org.dod.internet.security.mechanisms.srp (1.3.6.1.5.5.8).

Note that it is not possible to make this requirement part of the security protocol itself, because other authentication frameworks have different requirements for the initial octets in a mechanism buffer.

8.4 Security Layer

This mechanism does not provide distinct replay detection and sequencing services as part of the security layer. Both of these services are provided through the use of sequence numbers in integrity protected messages. If a GSS-API caller sets either the replay_det_req_flag or the sequence_req_flag (section 1.2.3 of [RFC-2078]) then this selects the "replay_detection" security service.

This mechanism does not make use of any channel binding data (section 1.1.6 of [RFC-2078]).

9. EAP

9.1 Overview

The Extensible Authentication Protocol (EAP) [RFC-2284] is an authentication framework that supports multiple authentication mechanisms. It is used with link layer protocols such as PPP and the IEEE-802 wired and wireless protocols.

9.2 Terminology

EAP uses the following terms to describe the entities involved in the authentication exchange [rfc2284bis]:

Authenticator: The entity that initiates EAP authentication in order to authenticate a Peer.

Peer: The entity that responds to requests from the Authenticator.

In this document, the Server corresponds to the Authenticator and the Client corresponds to the Peer.

9.3 Method Details

The EAP authentication method described in this document has the following properties:

Method Name: SRP

Method Type: 7

As described in section 2 of [rfc2284bis] the EAP authentication exchange is initiated by the Authenticator sending a Request packet to the peer with a Type field indicating the type of request. The Peer responds with a corresponding Reply packet, and the Authenticator and Peer exchange additional corresponding Request and Reply packets until the Authenticator deems that the authentication exchange is successful and complete, whereafter the Authenticator sends a Success packet. However, if at any time the Authenticator deems the authentication exchange to be unsuccessful it sends a Failure packet to indicate this.

When using this authentication method, the Type field in all Request and Reply packets is set to 7 and the Type Data is as described in Section 4 and the rest of this document. The diagrams below illustrate the EAP packet exchanges for this authentication method.

The following exchange occurs when a new session is negotiated

between the client and the server. It will also occur when the client requests re-use of the parameters of a previous session and either the server does not support such re-use or no longer considers the previous session to be valid:

```
 Peer (client)                            Authenticator (server)

 <------------  Request [ 7, { } ]  ----------------------------

 ----  Reply [ 7, { U, I, sid, cn } ] ------------------------->

 <------------  Request [ 7, { 00, N, g, s, B, L } ]  ----------

 ----  Reply [ 7, { A, M1, o, cIV } ]  ------------------------>

 <------------  Request [ 7, { M2, sIV, sid, ttl } ]  ----------

 ----  Reply [ 7, { } ]  -------------------------------------->
```

The following exchange occurs when the client requests that the parameters negotiated in a previous session be re-used in this session, but with a newly derived shared context key, and the server agrees:

```
 Peer (client)                            Authenticator (server)

 <-----------------------------  Request [ 7, { } ]  -----------

 ---------  Reply [ 7, { U, I, sid, cn } ]  ------------------->

 <-----------------------------  Request [ 7, { FF, sn } ]  ----

 ---------  Reply [ 7, { } ]  --------------------------------->
```

If a security layer is negotiated then the payloads of all subsequent lower layer packets sent over the link are protected using the negotiated security services.

9.4 Security Claims

As required by section 7.2 of [rfc2284bis], these are the security claims made by this authentication method indicating the level of security provided:

Intended Use: Wired networks, including PPP, PPPOE, and IEEE-802 wired media. Use over the Internet or with wireless media only when the recommended security layer has been negotiated.

Mechanism: Passphrase

Mutual authentication: Yes. This mechanism requires mutual authentication.

Integrity protection: Yes. The calculations of evidence that the shared context key is known - M1 sent by the client and M2 sent by the server - include the protocol elements received from the other party, so any modification by a third party will be detected. SRP itself is resistent to known active and passive attacks - see [SRP].

Replay protection: Yes. Both the client and the server randomly generate ephemeral private keys (a and b) that are used in the SRP calculations, but are not publicly revealed. New ephemeral private keys are generated for each session making replay attacks infeasible.

Confidentiality: No.

Key Derivation: No.

Dictionary attack protection: Yes. From [SRP]: "An attacker with neither the user's password nor the host's password file cannot mount a dictionary attack on the password".

Fast reconnect: Yes. An optional, optimised alternate authentication exchange is available where the parameters of a previously negotiated session are re-used, but with a newly derived shared context key - see Section 6.4.

Man-in-the-Middle resistance: Yes. The calculations of evidence - M1 sent by the client and M2 sent by the server - include the protocol elements received from the other party, so any modification by a third party will be detected. SRP itself is resistent to known active attacks, including man-in-the-middle attacks - see [SRP].

Acknowledged result indications: Yes. When the client receives M2 from the server it knows that the server has verified that the evidence (M1) it presented to prove its knowledge of the shared context key is correct, so it knows that it is authenticated to the server. When the server receives the empty response from the client at the end of the authentication exchange, it knows that the client has verified that the evidence (M2) it presented to prove its knowledge of the shared context key is correct, so it knows that it is authenticated to the client. Similarly for session re-use where the client receives the server nonce (sn)

from the server, and the server receives the final empty response from the client.

Key hierarchy: N/A

Key strength: The shared context key (K) negotiated between the client and server has a length of s, where "s" is the output length of the chosen underlying Message Digest Algorithm used in the SRP calculations (see "mda" option in Section 4.3). For example, the recommended Message Digest Algorithm SHA-160 has an output length of 160 bits, so in this case the length of K would be 160 bits. Keys for the confidentiality and integrity protection services are derived from K - see Section 5.1.2 - and have sizes appropriate for the algorithms being used. Note that all Message Digest Algorithms used with this mechanism MUST have an output of at least 16 bytes (see "mda" option in Section 4.3), which means that the shared context key will always have a length of at least 128 bits.

10. Security Considerations

This mechanism relies on the security of SRP, which bases its security on the difficulty of solving the Diffie-Hellman problem in the multiplicative field modulo a large safe prime. See section 4 "Security Considerations" of [RFC-2945], section 4 "Security analysis" of [SRP], and [SRP-6i].

This mechanism also relies on the security of the HMAC algorithm and the underlying hash function when integrity protection is used. Section 6 "Security" of [RFC-2104] discusses these security issues in detail. Weaknesses found in MD5 do not impact HMAC-MD5 [DOBBERTIN].

U, I, A and o, sent from the client to the server, and N, g, s, B and L, sent from the server to the client, could be modified by an attacker before reaching the other party. For this reason, these values are included in the respective calculations of evidence (M1 and M2) to prove that each party knows the shared context key K. This allows each party to verify that these values were received unmodified.

The use of integrity protection is RECOMMENDED to detect message tampering and to avoid session hijacking after authentication has taken place.

Replay attacks may be avoided through the use of sequence numbers, because sequence numbers make each integrity protected message exchanged during a session different, and each session uses a different key.

Research [KRAWCZYK] shows that the order and way of combining message encryption (Confidentiality Protection) and message authentication (Integrity Protection) are important. This mechanism follows the EtA (encrypt-then-authenticate) method and is "generically secure".

This mechanism uses a Pseudo-Random Number Generator (PRNG) for generating some of its parameters. Section 5.1.1 describes a securely seeded, cryptographically strong PRNG implementation for this purpose.

11. Acknowledgements

The following people provided valuable feedback in the preparation of this document:

Stephen Farrell <stephen.farrell@baltimore.ie>

Sam Hartman <hartmans@mit.edu>

Timothy Martin <tmartin@andrew.cmu.edu>

Alexey Melnikov <mel@messagingdirect.com>

Ken Murchison <ken@oceana.com>

Magnus Nystrom <magnus@rsasecurity.com>

David Taylor <DavidTaylor@forge.com.au>

Thomas Wu <tom@arcot.com>

Normative References

[RFC-2078]
Linn, J., "Generic Security Service Application Program Interface, Version 2", RFC 2078, January 1997, <http://www.ietf.org/rfc/rfc2078.txt>.

[RFC-2104]
Krawczyk, H., "HMAC: Keyed-Hashing for Message Authentication", RFC 2104, February 1997, <http://www.ietf.org/rfc/rfc2104.txt>.

[RFC-2119]
Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 0014, RFC 2119, March 1997, <http://www.ietf.org/rfc/rfc2119.txt>.

[RFC-2222]
Myers, J., "Simple Authentication and Security Layer (SASL)", RFC 2222, October 1997, <http://www.ietf.org/rfc/rfc2222.txt>.

[RFC-2284]
Blunk, L. and J. Vollbrecht, "PPP Extensible Authentication Protocol (EAP)", RFC 2284, March 1998, <http://www.ietf.org/rfc/rfc2284.txt>.

[rfc2284bis]
Blunk, L., Vollbrecht, J., Aboba, B., Carlson, J. and H. Levkowetz, "Extensible Authentication Protocol (EAP), work in progress", May 2003, <http://www.ietf.org/internet-drafts/draft-ietf-eap-rfc2284bis-03.txt>.

[RFC-2945]
Wu, T., "The SRP Authentication and Key Exchange System", RFC 2945, September 2000, <http://www.ietf.org/rfc/rfc2945.txt>.

[RFC-3454]
Hoffman, P. and M. Blanchet, "Preparation of Internationalized Strings ("stringprep")", RFC 3454, December 2002, <http://www.ietf.org/rfc/rfc3454.txt>.

[SASL] Myers, J., "Simple Authentication and Security Layer (SASL)", April 2002, <http://www.ietf.org/internet-drafts/draft-myers-saslrev-02.txt>.

[SASLprep]

Zeilenga, K., "SASLprep: Stringprep profile for user names and passwords, work in progress", May 2003, <http://www.ietf.org/internet-drafts/draft-ietf-sasl-saslprep-01.txt>.

[SRP] Wu, T., "The Secure Remote Password Protocol, Proceedings of the 1998 Internet Society Network and Distributed System Security Symposium, San Diego, CA, Mar 1998, pp. 97-111", March 1998, <http://srp.stanford.edu/ndss.html>.

[SRP-6i] Wu, T., "SRP-6: Improvements and Refinements to the Secure Remote Password Protocol", October 2002, <http://srp.stanford.edu/srp6.ps>.

Informative References

[AES] National Institute of Standards and Technology, "Rijndael: NIST's Selection for the AES", December 2000, <http://csrc.nist.gov/encryption/aes/rijndael/Rijndael.pdf>.

[DOBBERTIN] Dobbertin, H., "The Status of MD5 After a Recent Attack", December 1996, <ftp://ftp.rsasecurity.com/pub/cryptobytes/crypto2n2.pdf>.

[HAC] Menezes, A., van Oorschot, P. and S. Vanstone, "Handbook of Applied Cryptography", CRC Press, Inc., ISBN 0-8493-8523-7, 1997, <http://www.cacr.math.uwaterloo.ca/hac/about/chap7.ps>.

[ISO-10646] International Standards Organization, "International Standard --Information technology-- Universal Multiple-Octet Coded Character Set (UCS) -- Part 1 Architecture and Basic Multilingual Plane. UTF-8 is described in Annex R, adopted but not yet published. UTF-16 is described in Annex Q, adopted but not yet published.", ISO/IEC 10646-1, 1993.

[KRAWCZYK] Krawczyk, H., "The order of encryption and authentication for protecting communications (Or: how secure is SSL?)", June 2001, <http://eprint.iacr.org/2001/045/>.

[PKCS7] RSA Data Security, Inc., "PKCS #7: Cryptographic Message Syntax Standard", Version 1.5, November 1993, <ftp://ftp.rsasecurity.com/pub/pkcs/ascii/pkcs-7.asc>.

[RFC-1423] Balenson, D., "Privacy Enhancement for Internet Electronic Mail: Part III: Algorithms, Modes, and Identifiers", RFC 1423, February 1993, <http://www.ietf.org/rfc/rfc1423.txt>.

[RFC-2279] Yergeau, F., "UTF-8, a transformation format of Unicode and ISO 10646", RFC 2279, January 1998, <http://www.ietf.org/rfc/rfc2279.txt>.

[RFC-2440] Callas, J., Donnerhacke, L., Finney, H. and R. Thayer, "OpenPGP Message Format", RFC 2440, November 1998, <http:/

/www.ietf.org/rfc/rfc2440.txt>.

[RFC-2554]
Myers, J., "SMTP Service Extension for Authentication", RFC 2554, March 1999.

[RFC-2629]
Rose, M., "Writing I-Ds and RFCs using XML", RFC 2629, June 1999, <http://www.ietf.org/rfc/rfc2629.txt>.

[RFC-2847]
Eisler, M., "LIPKEY - A Low Infrastructure Public Key Mechanism Using SPKM", RFC 2847, June 2000, <http://www.ietf.org/rfc/rfc2847.txt>.

[SCAN] Hopwood, D., "Standard Cryptographic Algorithm Naming", June 2000, <http://www.eskimo.com/~weidai/scan-mirror/>.

[SRP-6] Wu, T., "SRP Protocol Design", October 2002, <http://srp.stanford.edu/design.html>.

[SRPimpl] Wu, T., "SRP: The Open Source Password Authentication Standard", March 1998, <http://srp.stanford.edu/srp/>.

[UMAC] Black, J., Halevi, S., Krawczyk, H., Krovetz, T. and P. Rogaway, "UMAC: Fast and Secure Message Authentication, Advances in Cryptology - CRYPTO '99. Lecture Notes in Computer Science, vol. 1666, Springer-Verlag, 1999, pp. 216-233", October 2000, <http://www.cs.ucdavis.edu/~rogaway/umac/umac_proc.pdf>.

[UNICODE] The Unicode Consortium, "The Unicode Standard, Version 3.2.0, is defined by The Unicode Standard, Version 3.0, as amended by the Unicode Standard Annex #27: Unicode 3.1 and by the Unicode Standard Annex #28: Unicode 3.2.", March 2002, <http://www.unicode.org/reports/tr28/tr28-3.html>.

[UNICODE-KC]
Durst, D., "Unicode Standard Annex #15: Unicode Normalization Forms.", March 2001, <http://www.unicode.org/unicode/reports/tr15>.

Authors' Addresses

Keith Burdis
Rhodes University
Computer Science Department
Grahamstown 6139
ZA

EMail: keith@rucus.ru.ac.za

Raif S. Naffah
Forge Research Pty. Limited
Suite 116, Bay 9
Locomotive Workshop,
Australian Technology Park
Cornwallis Street
Eveleigh, NSW 1430
AU

EMail: raif@forge.com.au

Appendix A. Modulus and Generator Values

Modulus N and generator g values for various modulus lengths are given below. In each case the modulus is a large safe prime and the generator is a primitve root of GF(n) [RFC-2945]. These values are taken from software developed by Tom Wu and Eugene Jhong for the Stanford SRP distribution [SRPimpl].

```
[264 bits]
  Modulus (base 16) =
    115B8B692E0E045692CF280B436735C77A5A9E8A9E7ED56C965F87DB5B2A2
    ECE3
  Generator = 2

[384 bits]
  Modulus (base 16) =
    8025363296FB943FCE54BE717E0E2958A02A9672EF561953B2BAA3BAACC3E
    D5754EB764C7AB7184578C57D5949CCB41B
  Generator = 2

[512 bits]
  Modulus (base 16) =
    D4C7F8A2B32C11B8FBA9581EC4BA4F1B04215642EF7355E37C0FC0443EF75
    6EA2C6B8EEB755A1C723027663CAA265EF785B8FF6A9B35227A52D86633DB
    DFCA43
  Generator = 2

[640 bits]
  Modulus (base 16) =
    C94D67EB5B1A2346E8AB422FC6A0EDAEDA8C7F894C9EEEC42F9ED250FD7F0
    046E5AF2CF73D6B2FA26BB08033DA4DE322E144E7A8E9B12A0E4637F6371F
    34A2071C4B3836CBEEAB15034460FAA7ADF483
  Generator = 2

[768 bits]
  Modulus (base 16) =
    B344C7C4F8C495031BB4E04FF8F84EE95008163940B9558276744D91F7CC9
    F402653BE7147F00F576B93754BCDDF71B636F2099E6FFF90E79575F3D0DE
    694AFF737D9BE9713CEF8D837ADA6380B1093E94B6A529A8C6C2BE33E0867
    C60C3262B
  Generator = 2

[1024 bits]
  Modulus (base 16) =
    EEAF0AB9ADB38DD69C33F80AFA8FC5E86072618775FF3C0B9EA2314C9C256
    576D674DF7496EA81D3383B4813D692C6E0E0D5D8E250B98BE48E495C1D60
    89DAD15DC7D7B46154D6B6CE8EF4AD69B15D4982559B297BCF1885C529F56
    6660E57EC68EDBC3C05726CC02FD4CBF4976EAA9AFD5138FE8376435B9FC6
```

```
    1D2FC0EB06E3
  Generator = 2

[1280 bits]
  Modulus (base 16) =
    D77946826E811914B39401D56A0A7843A8E7575D738C672A090AB1187D690
    DC43872FC06A7B6A43F3B95BEAEC7DF04B9D242EBDC481111283216CE816E
    004B786C5FCE856780D41837D95AD787A50BBE90BD3A9C98AC0F5FC0DE744
    B1CDE1891690894BC1F65E00DE15B4B2AA6D87100C9ECC2527E45EB849DEB
    14BB2049B163EA04187FD27C1BD9C7958CD40CE7067A9C024F9B7C5A0B4F5
    003686161F0605B
  Generator = 2

[1536 bits]
  Modulus (base 16) =
    9DEF3CAFB939277AB1F12A8617A47BBBDBA51DF499AC4C80BEEEA9614B19C
    C4D5F4F5F556E27CBDE51C6A94BE4607A291558903BA0D0F84380B655BB9A
    22E8DCDF028A7CEC67F0D08134B1C8B97989149B609E0BE3BAB63D4754838
    1DBC5B1FC764E3F4B53DD9DA1158BFD3E2B9C8CF56EDF019539349627DB2F
    D53D24B7C48665772E437D6C7F8CE442734AF7CCB7AE837C264AE3A9BEB87
    F8A2FE9B8B5292E5A021FFF5E91479E8CE7A28C2442C6F315180F93499A23
    4DCF76E3FED135F9BB
  Generator = 2

[2048 bits]
  Modulus (base 16) =
    AC6BDB41324A9A9BF166DE5E1389582FAF72B6651987EE07FC3192943DB56
    050A37329CBB4A099ED8193E0757767A13DD52312AB4B03310DCD7F48A9DA
    04FD50E8083969EDB767B0CF6095179A163AB3661A05FBD5FAAAE82918A99
    62F0B93B855F97993EC975EEAA80D740ADBF4FF747359D041D5C33EA71D28
    1E446B14773BCA97B43A23FB801676BD207A436C6481F1D2B9078717461A5
    B9D32E688F87748544523B524B0D57D5EA77A2775D2ECFA032CFBDBF52FB3
    786160279004E57AE6AF874E7303CE53299CCC041C7BC308D82A5698F3A8D
    0C38271AE35F8E9DBFBB694B5C803D89F7AE435DE236D525F54759B65E372
    FCD68EF20FA7111F9E4AFF73
  Generator = 2
```

Appendix B. Changes since the previous draft

Removed specific references to SASL in the main document, instead isolating them to their own section.

Added sections describing how the mechanism can be used with the GSS-API and EAP authentication frameworks.

Adopted SRP-6 exchange for the base protocol.

Mandated the use of SASLprep profile for text based information.

Added an optional, optimised alternate authentication exchange where the parameters of a previously negotiated session are re-used, but with a newly derived shared context key.

TODO: Regenerate SASL example.

Intellectual Property Statement

The IETF takes no position regarding the validity or scope of any intellectual property or other rights that might be claimed to pertain to the implementation or use of the technology described in this document or the extent to which any license under such rights might or might not be available; neither does it represent that it has made any effort to identify any such rights. Information on the IETF's procedures with respect to rights in standards-track and standards-related documentation can be found in BCP-11. Copies of claims of rights made available for publication and any assurances of licenses to be made available, or the result of an attempt made to obtain a general license or permission for the use of such proprietary rights by implementors or users of this specification can be obtained from the IETF Secretariat.

The IETF invites any interested party to bring to its attention any copyrights, patents or patent applications, or other proprietary rights which may cover technology that may be required to practice this standard. Please address the information to the IETF Executive Director.

Full Copyright Statement

Acknowledgement

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