Network Working Group
Internet Draft: PASS-DSS-SHA-3DES-1 SASL Mechanism
Document: draft-newman-sasl-passdss-00.txt

C. Newman
Innosoft

January 1998
Expires in six months

DSS Secured Password Authentication Mechanism

Status of this memo

Abstract

Some system administrators are faced with a choice between deploying a new authentication infrastructure or sending unencrypted passwords in the clear over the Internet. Deploying a new authentication infrastructure often involves modifying operating system services or keeping parallel authentication databases up to date and is thus unacceptable to many administrators.

Solutions which encrypt the entire session are often crippled with weak keys (due to government restrictions) which are unsuitable for passwords. In addition, such solutions often reduce performance of the entire session to an unacceptable level. This specification defines a SASL [SASL] mechanism which is compatible with existing password-based authentication databases and does not require a security layer for the remainder of the session.

[NOTE: Public discussion of this mechanism may take place on the

ietf-sasl@imc.org mailing list with a subscription address of ietf-sasl-request@imc.org. Private comments may be sent to the author].

1. How to Read This Document

This document is intended primarily for a programmer. If successful, it should be possible for a competent programmer to write a client implementation using this specification, the SASL [SASL] specification, an understanding of how to generate random numbers [RANDOM], a description or implementation of the DES and SHA1 [SHA1] algorithms and a multiprecision integer math library. A cryptographic library or a copy of "Applied Cryptography" [SCHNEIER] or similar reference is helpful for any implementation and necessary for server DSS key generation.

The key words "MUST", "MUST NOT", "SHOULD", "SHOULD NOT", and "MAY" in this document are to be interpreted as defined in "Key words for use in RFCs to Indicate Requirement Levels" [KEYWORDS].

1.1. Data Types Used in this Document

A list of data types used in this document follows. Note that the majority of this section is copied from the secure shell specification [SSH-ARCH].

octet A basic 8-bit unit of data.

uint32 A 32-bit unsigned integer. Stored as four octets in network byte order (also known as big endian or most significant byte [MSB] first). For example, the decimal value 699921578 (hexidecimal 29b7f4aa) is represented with the hexidecimal octet sequence 29 b7 f4 aa.

string A string is a length-prefixed octet string. The length is represented as a uint32 with the data immediately following. A length of 0 indicates an empty string. The string MAY contain NUL or 8-bit octets. When used to represent textual strings, the characters are interpreted in UTF-8 [UTF-8]. Other character encoding schemes MUST NOT be used.

mpint Represents multiple precision integers in two's complement format, stored as a string, most signficant octet first. Negative numbers have one in the most significant bit of the first octet of the string data. If the most significant bit would be set for a positive number, the number MUST be preceded by a zero byte. Unnecessary leading zero or 255

bytes MUST NOT be included. The value zero MUST be stored as a string with zero bytes of data.

By convention, a number that is used in modular computations in the field of integers mod n SHOULD be represented in the range 0 <= x < n.

Examples:

value (hex)	representation (hex)
0	00 00 00 00
9a378f9b2e332a7	00 00 00 08 09 a3 78 f9 b2 e3 32 a7
80	00 00 00 02 00 80
-1234	00 00 00 02 ed cc
-deadbeef	00 00 00 05 ff 21 52 41 11

1.2. Glossary

This section includes some acronyms used in this document.

DES The U.S. Government Data Encryption Standard is a symmetric encryption algorithm introduced in 1975 which uses a 56 bit key. The algorithm is documented in [SCHNEIER].

DSA The U.S. Government Digital Signature Algorithm standard. A public key signature algorithm available for unrestricted use without a license.

DSS The U.S. Government Digital Signature Standard [DSS] which employs the DSA algorithm.

HMAC A hash-based message authentication code [HMAC] summarized in Appendix A.4. Test cases are available in [HMAC-TEST].

SHA The Secure Hash Algorithm (version 1) which is part of the DSS standard.

triple-DES
Use of the DES algorithm three times in an encrypt-decrypt-encrypt mode with three independent keys as described in appendix A.3.

2. Overview

This section includes a brief discussion of design goals, intended use and an overview for this SASL mechanism. An overview of some of the algorithms used is in Appendix A.

2.1. Design Goals

The ideal authentication mechanism would be simple, fast, fully secure, freely distributable without restrictions and backwards compatible with deployed back-end authentication databases. Unfortunately, it is not possible to achieve all these goals so priorities and tradeoffs are necessary. This mechanism has compatibility with deployed back-end authentication databases and protection from passive and active attacks on the underlying connection as primary design goals. Simplicity and unrestricted binary distribution are secondary design goals.

Backwards compatibility is achived by using plaintext passphrases. Protection from passive and active attacks is achieved by using public and symmetric key technology to encrypt the passphrase and optionally protect the remainder of the ession. Some simplicity is achieved by avoiding complicated public key certification issues and formats as well as making the SASL session security layer and certification by the client optional. Unrestricted binary distribution is achieved by using unencumbered algorithms and making the SASL privacy layer optional.

2.2. Intended Use

This is intended as a plug-and-play mechanism for services layered on top of deployed passphrase-based back-end authentication databases. When no security layer is implemented it can be added to a SASL-based protocol without modifying or substituting network read and write APIs. When the optional session privacy protection is omitted, the author speculates that it may be possible to make a binary implementation which could be exportable from the United States.

For cases where simplicity, speed or unrestricted source code distribution is more desirable than backwards compatibility or security, a mechanism such as CRAM-MD5 [CRAM-MD5] or SCRAM [SCRAM] is preferred.

The optional SASL integrity and privacy protection is provided as a simple alternative to full service security layers such as TLS [TLS] or Secure Shell [SSH-ARCH]. However, there are many advantages to full service security layers such as compression, faster symmetric cipher options, and the ability to leverage other public key infrastructures. An implementation which supports TLS may have no incentive to support SASL security layers at all. The complexity verses functionality tradeoff is significant enough that these mechanisms do not compete.

2.3. Mechanism Overview

The PASS-DSS-SHA-3DES-1 mechanism uses three components to perform a secure authentication against a legacy passphrase database.

In order to protect against active attacks, a DSS public key in a format compatible with Secure Shell [SSH-TRANS] is used to authenticate the server to the client. The client is presumed to have the server's public key or a SHA-1 hash thereof stored locally in a secure database. If the client is willing to risk exposure to active attacks, it may skip the public key certification step altogether or do a one-time initialization of its local database, perhaps with user interaction.

In addition to the DSS public key, a Diffie-Hellman key exchange is used to generate a key for encrypting the passphrase. The "PASS-DSS-SHA-3DES-1" variant of this mechanism uses the same fixed Diffie-Hellman group used by Secure Shell's diffie-hellman-group1-sha1 key exchange [SSH-TRANS]. If more groups are necessary, they will be assigned to mechanism variants "PASS-DSS-SHA-3DES-2" and so forth.

Finally, the triple-DES algorithm is used to encrypt the client's passphrase and send it to the server.

2.4. Message Format Overview

This section provides a quick overview of the format of the messages. The formal definition of the syntax for these messages is in section 6. A detailed discussion of their implementation on clients and servers is in sections 3 and 4 respectively.

First message from client to server:

```
  string azname       ; the user name to login as, may be empty if
                        same as authentication name
  string authname     ; the authentication name
  mpint  X            ; Diffie-Hellman parameter X
```

The challenge from server to client is as follows:

```
  uint32   pklength   ; length of SSH-style DSA server public key
    string "ssh-dss"  ; constant string "ssh-dss" (lower case)
    mpint  p          ; DSA public key parameters
    mpint  q
    mpint  g
    mpint  y
  mpint    Y          ; Diffie-Hellman parameter Y
  OCTET    ssecmask   ; SASL security layers offered
  3 OCTET  sbuflen    ; maximum server security layer block size
```

```
  uint32   siglength  ; length of SSH-style dss signature
    string "ssh-dss"  ; constant string "ssh-dss" (lower case)
    mpint  r          ; DSA signature parameters
    mpint  s
```

The client then sends the following message encrypted with triple-DES:

```
  OCTET    csecmask   ; SASL security layer selection
  3 OCTET  cbuflen    ; maximum client block size
  string   passphrase ; the user's passphrase
  20 OCTET cli-hmac   ; a client HMAC-SHA-1 signature
```

3. Client Implementation of PASS-DSS-SHA-3DES-1

This section includes a step-by-step guide for client implementors. Although section 6 contains the formal definition of the syntax and is the authoritative reference in case of errors here, this section should be sufficient to build a correct implementation.

The SASL mechanism name is "PASS-DSS-SHA-3DES-1".

The value of n used for the Diffie-Hellman exchange is as follows (represented as an unsigned hexidecimal integer):

```
FFFFFFFF FFFFFFFF C90FDAA2 2168C234 C4C6628B 80DC1CD1
29024E08 8A67CC74 020BBEA6 3B139B22 514A0879 8E3404DD
EF9519B3 CD3A431B 302B0A6D F25F1437 4FE1356D 6D51C245
E485B576 625E7EC6 F44C42E9 A637ED6B 0BFF5CB6 F406B7ED
EE386BFB 5A899FA5 AE9F2411 7C4B1FE6 49286651 ECE65381
FFFFFFFF FFFFFFFF.
```

When represented as an "mpint", this would have a prefix of "0000008100." The value of g is 2. This group was taken from the ISAKMP/Oakley specification, and was originally generated by Richard Schroeppel at the University of Arizona. Properties of this prime are described in [Orm96].

The client begins by doing the following:

(A) Generate the Diffie-Hellman private value "x". This can be generated as a random integer between (n - 1)/4 and (n - 1)/2. It is important to generate a good random number [RANDOM].

(B) Compute the Diffie-Hellman public value "X" as follows. If X has a value of 0, repeat step (A).

$$X = 2^{x} \bmod n$$

The client then sends the following three pieces of information to the server:

(1) An authorization identity represented as a string. When the empty string is used, this defaults to the authentication identity. This is used by system administrators or proxy servers to login with a different user identity.

(2) An authentication identity represented as a string. This is the identity whose passphrase will be used.

(3) The "X" result from step (B) represented as an mpint.

The server responds by sending a message containing the following information:

(4) An "ssh-dss" public key compatible with Secure Shell, including the 32-bit length prefix in network byte order, the Secure Shell string "ssh-dss" and mpints for "p", "q", "g" and "y" (see Appendix A.1).

(5) The mpint "Y" as defined for the Diffie-Hellman key exchange (see Appendix A.2).

(6) A single octet bitmask representing the security layers available in the same format used by the KERBEROS_V4 mechanism [SASL]. Bit 0 (value 1) indicates it is permissible to have no security layer. Bit 1 (value 2) indicates integrity protection is permissible. Bit 2 (value 4) indicates privacy protection for the rest of the session is available. The remaining bits are reserved for future use.

(7) A three octet unsigned integer in network byte order representing the maximum cipher-text buffer size the server is able to receive. If this is less than 32, it indicates that a SASL security layer is not supported.

(8) A DSA signature, including a 32-bit length, the Secure Shell string "ssh-dss" and mpints for "r" and "s".

The client then does the following:

(C) Verify that "Y" is between 1 and n - 1 inclusive. If "Y" is outside this range, the client MUST cancel the authentication.

(D) Verify that the public key from step (4) belongs to the server. This can be done either with a database of SSH public keys or with a database of SHA1 hashes of such public keys. If the client does

not have a matching entry for the server or does not have a public key database, it MAY skip this step although it SHOULD alert the user that the connection is susceptible to active attacks if it does so. It MAY also record the public key (or SHA1 hash thereof) in its database with permission from the user.

(E) Compute the Diffie-Hellman key K as follows:

$$K = Y^{x} \bmod n$$

(F) Create a buffer containing data from steps (1) through (7) in order immediately followed by K represented as an mpint.

(G) Compute the SHA1 hash of the buffer from (F). This produces a 20 octet result.

(H) If the public key from step (4) was not certified, this step MAY be skipped. Otherwise, verify that the DSS signature is a signature of (G). This computation is done as defined in appendix A.1 where the output of step (G) represents the message "m" (note that this results in SHA1 being applied twice).

(I) Compute the following 20-octet values. K represents the output of step (E) in mpint format. H represents the output of step (G). The || symbol represents string concatenation. "A" represents a single octet containing the US-ASCII value of capital letter A.

```
cs-encryption-iv    = SHA1( K || "A" || H )
sc-encryption-iv    = SHA1( K || "B" || H )
cs-encryption-key-1 = SHA1( K || "C" || H )
cs-encryption-key-2 = SHA1( K || cs-encryption-key-1 )
cs-encryption-key   = cs-encryption-key-1 || cs-encryption-key-2
sc-encryption-key-1 = SHA1( K || "D" || H )
sc-encryption-key-2 = SHA1( K || sc-encryption-key-1 )
sc-encryption-key   = sc-encryption-key-1 || sc-encryption-key-2
cs-integrity-key    = SHA1( K || "E" || H )
sc-integrity-key    = SHA1( K || "F" || H )
```

(J) Create a buffer beginning with a bitmask for the selected security layer (it MUST be one offered in 6) followed by three octets representing the maximum cipher-text buffer size (at least 32) the client can accept in network byte order. This is followed by a string containing the passphrase. Note that integrity protection is pointless unless the public key was certified in step (D) and the signature was verified in step (H).

(K) Create a buffer containing items (1) through (7) immediately followed by the first four octets of (J).

(L) Compute HMAC-SHA-1 with (K) as the data and the cs-integrity-key from step (I) as the key. This produces a 20 octet result. A summary of the HMAC-SHA-1 algorithm [HMAC] is in appendix A.4.

(M) Create a buffer containing (J) followed by (L) followed by an arbitrary number of zero octets as necessary to reach the block size of DES and conceal the passphrase length from an eavesdropper.

(N) Apply the triple-DES algorithm to (M) with the first 8 octets of cs-encryption-iv from step (I) as the initialization vector and the first 24 octets of cs-encryption-key as the key. If optional privacy protection is negotiated on, the triple-DES state is not reset.

The client then sends a message to the server containing the following:

(9) The output of step (N).

If a SASL security layer is negotiated on, the following steps are used when sending a message:

(O) Create a buffer containing a uint32 client packet number (starting from 0) immediately followed by the cs-integrity-key from step (I).

(P) Compute HMAC-SHA-1 with (O) as the key and the data to transmit as the data.

(Q) Create a buffer containing the data to transmit followed by the 20-octet output of (P). If privacy was negotiated on, this is followed by zero to seven padding octets followed by one more octet indicating the number of padding octets. The total size MUST be a multiple of the DES blocksize.

(R) The result of step (Q) is encrypted with triple-DES if privacy was negotiated and is sent prefixed by a uint32 length, as required by SASL.

If a SASL security layer was negotiated on, the following steps are taken when receiving a message:

(S) If privacy was negotiated on, the message is decrypted using triple-DES with the first 24 octets of sc-encryption-key as the key. The value of the last octet plus one indicates the number of octets to ignore at the end of the output. The sc-encryption-iv is used to initialize triple-DES state the first time this is done.

(T) Create a buffer containing a uint32 server packet number (starting from 0) immediately followed by the sc-integrity-key.

(U) Compute HMAC-SHA-1 with (T) as the key over the portion of the data excluding the 20 octet signature and any encryption padding. If this is the same as the 20 octet signature, then the data is not corrupted.

4. Server Implementation of PASS-DSS-SHA-3DES-1

The section includes a step-by-step guide for server implementors. It is intended to be read in conjunction with section 3.

The server MUST have a persistant DSS-SSH public key. Mechanisms for generating such keys are described in [SCHNEIER] and [DSS].

IMPORTANT NOTE: The server MUST be able to process any message from the client, including messages of any size, messages with invalid content and messages with NULs in the middle of strings. When input is illegal, the server MUST gracefully reject authentication or in extreme cases gracefully terminate the connection. Particular care to avoid buffer overruns is important if the user name or passphrase strings are copied.

The server performs the following computations prior to or during the connection by the client:

(a) Select a random number k from (p - 1)/4 to (p - 1)/2. It is important to generate a good random number [RANDOM].

(b) Compute signature component "r" as follows:

$$r = (g^{k} \bmod p) \bmod q$$

(c) Optionally pre-compute the group inverse of k, mod q and the value xr.

(d) Select a random number y from (n - 1)/4 to (n - 1)/2. It is important to generate a good random number [RANDOM].

(e) Compute the Diffie-Hellman public value Y as follows:

$$Y = 2^{y} \bmod n$$

(f) Verify that the value X from the client is between 1 and (n - 1). If it isn't, fail the authentication.

(g) Compute the Diffie-Hellman shared key K as follows:

$$K = X^{y} \bmod n$$

(h) Create a buffer containing items (1) through (7) above followed by K represented as an mpint.

(i) Compute the SHA-1 hash of the buffer from (h). This produces a 20 octet result.

(j) Generate a DSS signature of (i). The signature is made up of "r" from step (b) and the result following computation (partially completed in step c):

$$s = (k^{-1} (SHA1(h) + xr)) \bmod q$$

(k) Create a buffer containing items (4) through (8) and send it to the client.

(l) Perform the computations as described in step (I) where K is the result of step (g) in mpint format and H is the result of step (i).

(m) Decrypt message (9) from the client using triple-DES with cs-encryption-iv as the initialization vector and the first 24 octets of cs-encryption-key as the key.

(n) Verify the passphrase from the output of step (m) against the authentication database. Fail the authentication if verification fails.

(o) Verify that the selected security layer is permitted and the cipher text buffer size is at least 32. If not, fail the authentication.

(p) Create a buffer containing steps (1) through (7) followed by the first four octets of the result from (m).

(q) Compute the HMAC-SHA-1 of (p) with cs-integrity-key as the key. This produces a 20-octet result.

(r) Compare the output of (q) with the 20 octet signature after the passphrase in the output of (m). If they don't match, fail the authentication.

If a SASL security layer is negotiated on, sending and receiving procedures are similar to steps (O)-(U), with client and server roles exchanged (and thus sc-* values and cs-* value exchanged). Note that triple-DES state from step (m) is not reset.

5. Example

The following is an example of the PASS-DSS-SHA-3DES-1 mechanism using the IMAP [IMAP4] profile of SASL. Note that base64 encoding and the lack of an initial client reponse with the first command are characteristics of the IMAP profile of SASL and not characteristics of SASL or this mechanism.

In this example, "C:" represents lines sent from the client to the server and "S:" represents lines sent from the server to the client. The wrapped lines are for editorial clarity -- there are no actual newlines in the middle of the messages.

```
C: a001 AUTHENTICATE PASS-DSS-SHA-3DES-1
S: +
C: AAAAAAAAAAVjaHJpcwAAAIByNdmAm3TWjyEeKfuyLoGUaojr5moPVU6p92/r
   5QDUAgO9A81wvQH5RbljV7DBONObjlkjfDGbQp3MsmUbtOiNDZrNczUMlCvd
   oNsWtQQtU48WrTIL+UP6EQn2Xblw2TgPCFmSHS66p/VILI5BsQsrx31BHnnn
   YVwKz6S7KkYyDA==
S: AAAA8gAAAAdzc2gtZHNzAAAAQQDPVlO6nFefrq6fA/dQKIoNj75JjppkVv3D
   kyILABCox2dMql0bnO48rHFuo167y6oukT/ocKupIw6bgKmdofgdAAAAFQDR
   pB6FrxemUGRuLjY/oiH/Qef14QAAAEEAkVr9rOlB58k5XoqmPNYTrVGZKWbC
   PcYtaL92ANxgWyjyRo49+m0+fHPNhNibQoLddEZF8lHPKWgb7z7qz0QMdgAA
   AEARcIEiMz5jTZo8COf2njL3BTWRND5NGAgZY7s1YOm2BfjVyf1/MkOiQMiX
   eonrsfMc0sWQGgpRYRtJWpe56cc2AAAAgQDa/kF4dHlBpgYew6u/10sFJP8G
   QgSE4YdFUl2yJKW4S9azMmqSVsoiAHzeslZogV25yQE3vdsIjtqjVwhcCwu2
   nb0kt/Rfu5gOTCygUWsRD0yuiKeOpbiakCLQe1jtjIS2tgLRKQ66Z+q7HI9R
   YUqqxFUu53L/iGNf1cbVokepOgEAAAAAAAA7AAAAB3NzaC1kc3MAAAAUMkWf
   YSZlk0bJI4BKiA8Ju6Yh0DAAAAAUScKvxfqPCm2cdnNyzD7sgmUxC1E=
C: XTs/rtmZ6uYpK8A90hRN5NrdoEAXrzWISOb080+EOFEDd6Gg2Ci720Pw8kRe
   VY6I
S: a001 OK Authentication Completed
```

In this example, the client private "x" value, represented as an mpint in hexidecimal is:

```
00000080 4bb82066 70638c78 059368d3 8430006c 02e8a05b 708cb774
55b5263b 366f7825 4feaba0d 2e0191c0 411b8ca9 58421a61 26d5a1a1
246797de 7527d020 f0190df0 6939b4fd bf70ea68 4c7f7100 e930008e
4feaba0d 2e0191c0 411b8ca9 58421a61 4feaba0d 2e0191c0 411b8ca9
58421a61 4feaba0d 2e0191c0 411b8ca9 58421a61
```

The server private "y" value represented as an mpint in hexidecimal is:

```
00000080 4340eba3 631a0fe9 224bdd5c 8d660535 6c290c1d 446e8122
768c6298 2cfa2d6c 057ab971 80cd57d1 c51ff528 0884cbf8 057ab971
80cd57d1 c51ff528 0884cbf8 057ab971 80cd57d1 c51ff528 0884cbf8
```

```
  057ab971 80cd57d1 c51ff528 0884cbf8 057ab971 80cd57d1 c51ff528
  0884cbf8 057ab971 80cd57d1 c51ff528 0884cbf8
```

The Diffie-Hellman shared secret represented as an mpint in hexidecimal is:

```
  00000080 7a81b3c2 702168d8 cce24ccc ae0b938e 84cf27c9 2b192c20
  4ab821f5 fa7b0120 26c45066 c8af8839 4ea186f9 0972d80e 2ff2341a
  ba3e8f99 36c10574 c27989fe 719c42ff 2ec45d87 87bba1f5 e9bd2dbf
  fbfdf72b b1eee8f3 109e9a3c 40168785 ed4f3ba7 118a864c ebd6dea3
  96d93250 821197c4 40a3f13a d0f7ad5f cf22664b
```

The DSA private key value represented as an mpint in hexidecimal is:

```
  00000040 11708122 333e634d 9a3c08e7 f69e32f7 05359134 3e4d1808
  1963bb35 60e9b605 f8d5c9fd 7f3243a2 40c8977a 89ebb1f3 1cd2c590
  1a0a5161 1b495a97 b9e9c736 00000014 252bcbfa 5634d706 6ed43128
  972e181e 66bf9c30
```

And the SHA-1 hash value used to compute the keys is:

```
  a5567b02 0f5abe1a b8aa9040 08404432 71744e6a
```

6. Formal Syntax of PASS-DSS-SHA-3DES-1 Messages

This is the formal syntactic definition of the client and server messages. This uses ABNF [ABNF] notation including the core rules. The first three rules define the formal exchange. The later rules define the elements of the exchange.

```
client-msg-1     = [azname] authname diffie-hellman-X

server-msg-1     = dss-public-key diffie-hellman-Y
                   ssecmask sbuflen dss-signature

client-msg-2     = client-blob


authname         = string
                   ;; interpreted as UTF-8 [UTF-8]

azname           = string
                   ;; interpreted as UTF-8 [UTF-8]

cbuflen          = 3OCTET
                   ;; Big endian binary unsigned integer
                   ;; max length of client read buffer
```

```
cli-hmac         = 20OCTET

client-blob      = 8*OCTET
                   ;; encrypted version of client-encrypted

client-encrypted = csecmask cbuflen passphrase cli-hmac *NUL
                   ;; MUST be multiple of DES block size

csecmask         = OCTET
                   ;; client selected protection layer

diffie-hellman-X = mpint

diffie-hellman-Y = mpint

dss-g            = mpint

dss-p            = mpint

dss-public-key   = length NUL NUL NUL %x07 "ssh-dss"
                   dss-p dss-q dss-g dss-y
                   ;; length is total length of remainder
                   ;; as defined in [SSH-TRANS]

dss-q            = mpint

dss-r            = mpint

dss-signature    = length NUL NUL NUL %x07 "ssh-dss"
                   dss-r dss-s
                   ;; length is total length of remainder

dss-s            = mpint

dss-y            = mpint

length           = 4OCTET
                   ;; binary number, big endian format (MSB first)

mpint            = length *OCTET
                   ;; length specifies number of octets
                   ;; see section 1 for detailed mpint definition

passphrase       = string
                   ;; At least 64 octets MUST be supported
```

```
sbuflen          = 3OCTET
                   ;; Big endian binary unsigned integer
                   ;; max length of server read buffer

ssecmask         = OCTET
                   ;; server protection layer mask

string           = length *OCTET
                   ;; the length determines the number of octets
                   ;; OCTETs are interpreted as UTF-8

NUL              = %x00  ;; US-ASCII NUL character
```

7. Security Considerations

Security considerations are discussed throughout this memo.

This mechanism supplies the server with the plaintext passphrase, so the server gains the ability to masquerade as the user to any other services which share the same passphrase.

If the public key certification step is skipped, then an active attacker can gain the client's passphrase and thus the ability to masquerade as the user to any other services which share the same passphrase. Negotiating a security layer will fail to provide protection from an active attacker in this case.

If no security layer is negotiated, the rest of the protocol session is subject to active and passive attacks.

If an integrity-only layer is negotiated, the rest of the protocol is subject to passive eavesdropping.

The quality of this mechanism depends on the quality of the random number generator used. See [RANDOM] for more information.

8. Multinational Considerations

As remote access is a crucial service, users are encouraged to restrict user names and passphrases to the US-ASCII character set. However, if characters outside the US-ASCII chracter set are used in user names and passphrases, then they are interpreted according to UTF-8 [UTF-8] and it is a protocol error to include any octet sequences not legal for UTF-8. Servers are encourged to enforce this restriction to discourage clients which use unlabelled character sets in this context.

9. Intellectual Property Issues and Acknowledgements

David Kravitz holds U.S. Patent #5,231,668 on the DSA algorithm. NIST [NIST] has made this patent available world-wide on a royalty-free basis.

Diffie-Hellman was the first public-key algorithm ever invented. It was first published in 1976 [DIFFIE-HELLMAN]. U.S. Patent #4,200,770 granted April 1980 has expired. Canada Patent #1,121,480 was granted April 6, 1982 and may still apply at this time.

DES is covered under U.S. Patent #3,962,539 granted June 1978, which has expired.

The majority of the constructions in this specification were copied from the Secure Shell specifications [SSH-ARCH, SSH-TRANS]. Additional information is paraphrased from "Applied Cryptography" [SCHNEIER].

10. References

[ABNF] Crocker, Overell, "Augmented BNF for Syntax Specifications: ABNF", RFC 2234, Internet Mail Consortium, Demon Internet Ltd, November 1997.

[CRAM-MD5] Klensin, Catoe, Krumviede, "IMAP/POP AUTHorize Extension for Simple Challenge/Response", RFC 2195, MCI, September 1997.

[DIFFIE-HELLMAN] Diffie, W., Hellman, M.E., "Privacy and Authentication: An introduction to Cryptography," Proceedings of the IEEE, v. 67, n. 3, March 1979, pp. 397-427.

[DSS] National Institute of Standards and Technology, "Digital Signature Standard," NIST FIPS PUB 186, U.S. Department of Commerce, May 1994.

[HMAC] Krawczyk, Bellare, Canetti, "HMAC: Keyed-Hashing for Message Authentication", RFC 2104, IBM, UCSD, February 1997.

[HMAC-TEST] Cheng, Glenn, "Test Cases for HMAC-MD5 and HMAC-SHA-1", RFC 2202, IBM, NIST, September 1997.

[IMAP4] Crispin, M., "Internet Message Access Protocol - Version 4rev1", RFC 2060, University of Washington, December 1996.

[KEYWORDS] Bradner, "Key words for use in RFCs to Indicate Requirement Levels", RFC 2119, Harvard University, March 1997.

[NIST] "Proposed Federal Information Processing Standard for Secure Hash Standard," Federal Register, v. 57, n. 21, January 1992, pp. 3747-3749.

[Orm96] Orman, H., "The Oakley Key Determination Protocol", version 1, TR97-92, Department of Computer Science Technical Report, University of Arizona.

[RANDOM] Eastlake, Crocker, Schiller, "Randomness Recommendations for Security", RFC 1750, DEC, Cybercash, MIT, December 1994.

[SASL] Myers, "Simple Authentication and Security Layer (SASL)", RFC 2222, Netscape Communications, October 1997.

[SCHNEIER] Schneier, B., "Applied Cryptography: Protocols, Algorithms and Source Code in C," John Wiley and Sons, Inc., 1996.

[SCRAM] Newman, "Salted Challenge Response Authentication Mechanism (SCRAM)", work in progress, October 1997.

[SHA1] See [DSS].

[SSH-ARCH] Ylonen, Kivinen, Saarinen, "SSH Protocol Architecture", Work in progress, SSH, October 1997.

[SSH-TRANS] Ylonen, Kivinen, Saarinen, "SSH Transport Layer Protocol", Work in progress, SSH, October 1997.

[TLS] Dierks, Allen, "The TLS Protocol Version 1.0", Work in progress.

[UTF8] Yergeau, F. "UTF-8, a transformation format of Unicode and ISO 10646", RFC 2044, Alis Technologies, October 1996.

11. Author's Address

Chris Newman
Innosoft International, Inc.
1050 Lakes Drive
West Covina, CA 91790 USA

Email: chris.newman@innosoft.com

Appendix A. Algorithm Overview

This section provides a quick overview of the algorithms used. For

a full understanding, the reader is encouraged to read "Applied Cryptography" [SCHNEIER]. The follow descriptions are paraphrased from that source.

Note that an overview of the DES algorithm is not included as publicly available implementations and descriptions are very common.

Appendix A.1. DSA Algorithm

The DSA algorithm is a public key algorithm which can be used to sign messages such that the source can be verified using a public key. The algorithm has the following parameters:

p is a prime number L bits long. Implementations MUST support L between 512 and 1024 bits.

q is a 160-bit prime factor of (p - 1).

$$g = h^{(p-1)/q} \bmod p$$ where h is any number less than p - 1 such that $$h^{(p-1)/q}$$ is greater than 1.

x is a number less than q and represents the private key.

$$y = g^{x} \bmod p$$ and represents the public key.

To sign a message m, the client generates a random number k less than q and computes:

$$r = (g^{k} \bmod p) \bmod q$$

$$s = (k^{-1} (SHA1(m) + xr)) \bmod q$$

The signature is represented as r and s, and is verified as follows:

$$w = s^{-1} \bmod q$$

$$u1 = (SHA1(m) * w) \bmod q$$

$$u2 = (rw) \bmod q$$

$$v = ((g^{u1} * y^{u2}) \bmod p) \bmod q$$

If v = r then the signature is verified.

Appendix A.2. Diffie-Hellman Algorithm

The Diffie-Hellman algorithm is a key-exchange algorithm. It allows two ends of a communications channel to establish a shared secret which a passive eavesdropper can not easily determine. This key can then be used in a symmetric algorithm such as triple-DES. The two ends have a prior agreement on two numbers:

n a large prime number

g a primiative mod n.

The client chooses a random large integer x and computes:

$$X = g^{x} \bmod n$$

and sends X to the server. The server chooses a random large integer y and computes:

$$Y = g^{y} \bmod n$$

$$K = X^{y} \bmod n$$

The server sends Y to the client. The client computes:

$$K = Y^{x} \bmod n$$

At this point, the client and server share the same secret K.

Appendix A.3. Triple-DES Algorithm in EDE/outer-CBC Mode

The DES algorithm uses an 8 octet (64 bit) key of which 56 bits are significant. The triple-DES EDE algorithm uses a 24 octet (192 bit) key of which roughly 112 bits are significant see [SCHNEIER] for more details. The "EDE" refers to encrypt-decrypt-encrypt, and

the "CBC" refers to cipher-block-chaining where each cipher block affects future cipher blocks. If E() is the DES encryption function, D() is the DES decryption function, C is a cipher text block and P is a plaintext block, then triple-DES EDE in CBC mode with outer chaining is:

$$C_i = E_{K3}(D_{K2}(E_{K1}(P_i \text{ XOR } C_{i-1})))$$

NOTE: C_0 is the initialization vector

and the decryption function is:

$$P_i = C_{i-1} \text{ XOR } D_{K3}(E_{K2}(D_{K1}(C_i)))$$

K1 is the first 8 octets of the triple-DES key, K2 is the second 8 octets and K3 is the final 8 octets.

Appendix A.4. HMAC-SHA-1 Keyed hash function

HMAC-SHA-1 uses the SHA-1 hash function to create a keyed hash function suitable for use as an integrity protection function. A more complete description is in [HMAC]. A brief summary of the algorithm follows:

(A) If the key is longer than 64 octets, it is run through the SHA-1 function to produce a 20 octet key.

(B) The key is exclusive-ored with a 64 octet buffer filled with the octet value 0x36.

(C) SHA-1 is computed over (B) followed by the input text.

(D) The key is exclusive-ored with a 64 octet buffer filled with the octet value 0x5C.

(E) SHA-1 is computed over (D) followed by (C).