This document describes a one-time password authentication system (OTP). The OTP system provides authentication for system access [login] and other applications requiring authentication that is secure against passive attacks based on replaying captured reusable passwords. OTP evolved from the S/KEY* One-Time Password System that was released by Bellcore and is described in references [3] and [5].

ABSTRACT

This document describes a one-time password authentication system (OTP). The OTP system provides authentication for system access [login] and other applications requiring authentication that is secure against passive attacks based on replaying captured reusable passwords. OTP evolved from the S/KEY* One-Time Password System that was released by Bellcore and is described in references [3] and [5].

OVERVIEW

One form of attack on networked computing systems is eavesdropping on network connections to obtain authentication information such as the login IDs and passwords of legitimate users. Once this information is captured, it can be used at a later time to gain access to the computing system. One-time password systems are designed to counter this type of attack, called a "replay attack" [4].

The authentication system described in this document uses a secret pass-phase known only by the user to generate a sequence

* S/KEY is a trademark of Bellcore
of single use (one-time) passwords. With this system, the
user’s secret pass-phrase never crosses the network at any time
such as during authentication or during pass-phrase changes.
Thus, it is not vulnerable to replay attacks. Added security is
provided by the property that no secret information need be
stored on any system, including the server being protected.

The OTP system protects against external passive attacks
against the authentication subsystem. It does not prevent a
network eavesdropper from gaining access to private
information, and does not provide protection against either
"social engineering" or against active attacks where the
potential intruder is able to intercept and modify the packet
stream.

INTRODUCTION

There are two entities in the operation of the OTP one-time
password system. The client must generate the appropriate one-
time password from the user’s secret pass-phrase and from
information provided in the challenge from the server. The
server must send a challenge that includes the appropriate
generation parameters to the client, must verify the one-time
password received, must store the last valid one-time password
it received, and must store the corresponding one-time password
sequence number. The server must also facilitate the changing
of the user’s secret pass-phrase in a secure manner.

The OTP system generator passes the user’s secret pass-phrase,
along with a "seed" received from the server as part of the
challenge, through multiple iterations of a secure hash
function to produce a one-time password. On each use, the
number of secure hash function iterations is reduced by one.
Thus, a unique sequence of passwords is generated. The server
verifies the one-time password received from the client by
computing the secure hash function once and comparing the
result with the previously accepted one-time password. This
technique was first suggested by Leslie Lamport [1].

SECURE HASH FUNCTION

The security of the OTP system is based on the non-
invertability of a secure hash function. Such a function must
be tractable to compute in the forward direction, but
computationally infeasible to invert.

The interfaces are currently defined for three such hash
by NIST. Clearly, the generator and server must use the same
algorithm in order to interoperate. Other hash algorithms may
be specified for use with this system by publishing the
appropriate interfaces.

The secure hash algorithms listed above have the property that
they accept an input that is arbitrarily long and produce a fixed size output. The OTP system folds this output to 64 bits
using the algorithms in the Appendix A. 64 bits is also the length of the one-time passwords. This is believed to be long enough to be secure and short enough to be entered manually (see below, Form of Output) when necessary.

GENERATION OF ONE-TIME PASSWORDS

This section describes the generation of the one-time passwords. This consists of an initial step in which all inputs are combined, a computation step where the secure hash function is applied a specified number of times, and an output function where the 64 bit one-time password is converted to a human readable form.

Initial Step

In principle, the user’s secret pass-phrase may be of any length. To reduce the risk from techniques such as exhaustive search or dictionary attacks, it MUST contain at least 10 characters (see Form of Inputs below). All implementations MUST support at least a 63 character pass-phrase. The secret pass-phrase is frequently, but is not required to be, textual information provided by a user.

In this step, the pass phrase is concatenated with a seed that is transmitted from the server in clear text. This non-secret seed allows a client to use the same secret pass-phrase on multiple machines (using different seeds) and to safely recycle secret passwords by changing the seed. The seed is a string of characters that must not contain any blanks and should consist of strictly alphanumeric characters from the ISO-646 Invariant Code set.

The result of the concatenation is passed through the secure hash function, and then is reduced to 64 bits using one of the function dependent algorithms shown in Appendix A.

Computation Step

A sequence of one-time passwords is produced by applying the secure hash function multiple times to the output of the initial step (called S). That is, the first one-time password to be used is produced by passing S through the secure hash function a number of times (N) specified by the user. The next one-time password to be used is generated by passing S though the secure hash function N-1 times. An eavesdropper who has monitored the transmission of a one-time password would not be able to generate the next required password because doing so would mean inverting the hash function.

Form of Inputs

The secret pass-phrase is seen only by the OTP generator. To allow interchangeability of generators, all generators must
support a secret pass-phrase of 10 to 60 characters. Implementations may support a longer pass-phrase, but such
implementations risk the loss of interchangeability with implementations supporting only the minimum.

To achieve a uniform minimum pass-phrase complexity, all generators must enforce the following rules: Pass-phrases MUST be at least 10 characters in length, MUST contain at least one upper case alphabetic, MUST contain at least one lower case alphabetic, and MUST contain at least one non-blank non-alphabetic.

The seed must consist of purely alphanumeric characters and must be of one to 16 characters in length.

The sequence number and seed together constitute a larger unit of data called the challenge. The challenge gives the generator the parameters it needs to calculate the correct one-time password from the secret pass-phrase. The challenge is in a standard syntax so that automated generators can recognize the challenge in context and extract these parameters. The syntax of the challenge is:

```
otp-<algorithm identifier> <sequence integer> <seed>
```

The three tokens are separated by a single space and the entire challenge string must be terminated with either a space or a new line. If additional algorithms are defined, appropriate identifiers (short, but not limited to three characters) must be defined. The currently defined algorithm identifiers are:

```
md4       MD4 Message Digest
md5       MD5 Message Digest
sha       NIST Secure Hash Algorithm
```

Form of Output

The one-time password generated by the above procedure is 64 bits in length. Entering a 64 bit number is a difficult and error prone process. Some generators insert this password into the input stream and some others make it available for system "cut and paste." Still other arrangements require the one-time password to be entered manually. The OTP system is designed to facilitate this manual entry without impeding automatic methods. The one-time password therefore may be converted to, and all servers MUST accept it as, a sequence of six short (1 to 4 letter) easily typed words that only use characters from ISO-646 IVCS. Each word is chosen from a dictionary of 2048 words; at 11 bits per word, all one-time passwords may be encoded.

Interoperability requires at all OTP servers and generators use the same dictionary. The standard dictionary is defined as an appendix to RFC 1760 [5] and is not included in this document.

To facilitate the implementation of smaller generators, hex
output is an acceptable alternative for the presentation of the one-time password. All implementations of the server software
MUST accept lower case hex as well as six-word format.

VERIFICATION OF ONE-TIME PASSWORDS

An application on the server system that requires OTP authentication is expected to issue an OTP challenge as described above. Given the parameters from this challenge and the secret pass-phrase, the generator can compute (or lookup) the one-time password which is passed to the server to be verified.

The server system has a database containing, for each user, the one-time password from the last successful authentication or the first OTP of a newly initialized sequence. To authenticate the user, the server decodes the one-time password received from the client into a 64-bit key and then runs this key through the secure hash function once. If the result of this operation matches the stored previous OTP, the authentication is successful and the accepted one-time password is stored for future use.

PASS-PHRASE CHANGES

Because the number of hash function applications executed by the client decreases by one each time, at some point the user must reinitialize the system or be unable to authenticate. The method of doing this is under discussion and will be provided in a subsequent document.

Implementations MUST provide clients with a means of reinitializing a sequence through explicit specification of the first one-time password of a sequence. This allows a client to initialize without making it necessary to send a secret pass-phrase over the network as only the one-time password is sent. When the sequence of one-time password is reinitialized, implementations MUST verify that it seed is changed. Installations are advised to discourage any operation that sends the secret pass-phrase over a network as such practice defeats the concept of a one-time password.

ACKNOWLEDGMENTS

The idea behind OTP authentication was first proposed by Leslie Lamport [1]. Bellcore’s S/KEY system, from which OTP is derived, was proposed by Phil Karn, who also wrote most of the Bellcore reference implementation.
REFERENCES


SECURITY CONSIDERATIONS

This entire document is about Security Considerations

AUTHOR’S ADDRESS

Neil Haller
Bellcore
MRE 2Q-280
445 South Street
Morristown, NJ, 07960-6438, USA

Phone: +1 201 829-4478
Fax: +1 201 829-2504
Email: nmh@bellcore.com

Craig Metz
Kaman Sciences Corporation
INTERNET DRAFT     A One Time Password System        June 12,1995

Washington, DC, 20375-5337

Phone: +1 202 404-7122
Fax: +1 202 404-7942
Email: cmetz@cs.nrl.navy.mil
Appendix A - Interfaces to Secure Hash Algorithms

MD4 Message Digest (see reference [2])

```c
strcpy(buf,seed);
strcat(buf,passwd);
MDbegin(&md)
MDupdate(&md,(unsigned char *)buf,8*buflen);

/* Fold result to 64 bits */
md.buffer[0] ^= md.buffer[2];
md.buffer[1] ^= md.buffer[3];
```

MD5 Message Digest (see reference [6])

```c
MD5_CTX mdCxt;
strcpy(buf,seed);
strcat(buf,passwd);

/* Crunch the key through MD5 */
MD5Init(&mdCxt);
MD5Update(&mdCxt,(unsigned char *)bits,strlen(bits));
MD5Update(&mdCxt,(unsigned char *)buf,buflen);
MD5Final(&mdCxt);

/* Fold result to 64 bits */
for( i = 0; i < 8; i++ )
    result[i] = mdCxt.digest[i] ^ mdCxt.digest[i+8];
```

SHA Secure Hash Algorithm (see reference [7])

```c
/* Fold 160 bit result to 64 bits */
md.buffer[0] ^= md.buffer[2];
md.buffer[1] ^= md.buffer[3];
md.buffer[0] ^= md.buffer[4];
```