THE S/KEY™ ONE-TIME PASSWORD SYSTEM

Neil M. Haller

Bellcore
Morristown, New Jersey

ABSTRACT

Computing systems have been under increasingly sophisticated attack over the Internet and by using dial-up access ports. One form of attack is eavesdropping on network connections to obtain login id's and passwords of legitimate users. This information is used at a later time to attack the system. We have developed a prototype software system, the S/KEY™ one-time password system, to counter this type of attack and have been using it experimentally for external access to a research computer complex at Bellcore.

The S/KEY system has several advantages compared with other one-time or multi-use authentication systems. The user's secret password never crosses the network during login or when executing other commands requiring authentication such as the UNIX *passwd* (change password) or *su* (change privilege) commands. No secret information is stored anywhere, including on the host being protected, and the underlying algorithm may be made public. The remote end (client) of this system can run on any locally available computer and the host end (server) can be integrated into any application requiring authentication.

The S/KEY authentication system has been in experimental use at Bellcore for two years. It is available by anonymous ftp on the Internet.

INTRODUCTION

There are a variety of threats to be considered when operating a computer system. One can distinguish between *inside jobs* and external attacks; in this paper, we are concerned with attempts to penetrate a system of computers from *outside* the physical facility. We are not concerned with the additional security issues where legitimate users may attempt to increase their privilege (become super-users) or where insiders with physical access to the computers attempt to gain improper access.

We have built an experimental prototype S/KEY authentication system for a UNIX® environment, but there is nothing UNIX-specific about the design.

External Threats

There are several ways an external intruder might break into a UNIX system. These include guessing poorly chosen passwords, potentially with dictionary attacks; taking advantage of bugs in privileged UNIX system software (an example is the "Morris Worm" of November 1988 that exploited a bug in the Internet *finger* server[1].), and taking advantage of system configuration errors or poorly chosen system defaults. Properly configured and administered systems are not generally vulnerable to these attacks.

Other attacks take advantage of the information that crosses communications networks. One can obtain passwords for later use by passive eavesdropping, and the form of current passwords can be used to guess future ones. A potential intruder can actively interfere with legitimate network traffic by *spoofing* or disrupting the communications protocols.

Defenses Against Passive and Active Attacks

Data, including authentication information such as passwords, are carried on a variety of networks including LANS and private or public data and voice networks. If a potential intruder can gain access, either directly or using tools designed for other functions (such as network management), this access can be used to monitor traffic from legitimate users and collect passwords and other data for later use. This eavesdropping is classified as a *passive* network attack. Alternatively, an attacker might choose to disrupt or divert the communications of legitimate users, e.g., by spoofing traffic; this is classified as an *active* attack.

S/KEY is a trademark of Bellcore

UNIX is a registered trademark of X/Open.

In recent years, protocols capable of thwarting both passive and active attacks have been devised and implemented. Many of these, including ours, involve an exchange of data between the host being protected and the client attempting to gain authorized access. The defense against passive attacks is to make the data crossing the network useless to an eavesdropper; thus a potential intruder cannot gain improper access by *replaying* a saved valid authentication sequence.

The simplest form of such a defense is for the host to generate a random string and send it to the client. The client then uses some computing device to compute a key-based cryptographic function (see below, *Secure Hash Functions*) of this string and then return the output of this function to the host. The host executes the same function and compares the results.[2] A weakness of this system is that the secret keys must be available to the host, and protecting this host becomes a critical link in the security chain.

This system can be implemented in software or as a special purpose device to be carried by the client. The latter offers the advantages of portability and interface independence, but at an increased system cost.

Another defense is the use of hand-held device containing a clock that is synchronized to the host (or security processor). Both generate a sequence based on a secret seed that is stored in the host and is therefore a weak link.

The S/KEY system as described here is implemented in software. It is straightforward to convert this system to utilize a portable device, but we have not done so.

A notable example of a security protocol is the *Kerberos*[1] [3] authentication system from MIT's project Athena.[4] Kerberos solves the problem of passive eavesdropping within a single computing environment called a realm (multiple realms may be joined). This system does not, however, address the problem of access across a network using client software that is unable to fully participate in the Kerberos protocol. A client using a workstation within a realm (or connected realms) is completely and transparently protected by Kerberos against passive attacks, but a client using a simple (non-programmable) terminal or a non-participating[2] computer is vulnerable to an eavesdropping/replay attack.

A major strength of Kerberos is that it is capable of protecting against active attacks using encryption. This option gets limited use because of the overhead it imposes, but the increasing power of low cost computers makes this a short term problem. Given that it is much easier to conduct a passive attack than an active one without risking detection, we feel that there is still value in a system that protects only against passive attacks.

The S/KEY Authentication System

The S/KEY authentication system is a scheme that protects user passwords against passive attacks. It can be easily and quickly added to almost any UNIX system, without requiring any additional hardware and without requiring the system to store information (such as plain text passwords) that would be more sensitive than the encrypted passwords already stored. The S/KEY system can be used with "dumb terminals", personal computers with conventional communications programs, or workstations. It is conceptually compatible with a potential implementation based on smart cards or pocket calculators.

GOALS

Eavesdropping Protection

The primary goal of the S/KEY authentication system is to provide complete protection of the login-time authentication mechanism against passive eavesdropping. This protection implies that no information may cross the network that could potentially be used for authentication at a later time. An eavesdropper with complete transcripts of many user sessions, including password changes, should have no information that would be useful in attempting to login to the system.

Ease of Use

A security system must be easy to use. Not all users are willing to cope with a complex security system and it is virtually impossible to block all insider built back-doors. The more user-friendly the system is, the less likely it will be bypassed. Ideally the system should be as easy to use as a system protected by a conventional multi-use password system.

Automated Operation

A common form of remote access to a computing system is from another computer acting as a terminal using a communications program. This computer may

1. Athena and Kerberos are trademarks of MIT.
2. Most communication programs on non-UNIX systems (both using dial-up and Internet connectivity) are unable to participate in the Kerberos protocol. This is not to say that they could not be modified to do so.

be completely under the control of a single operator, and thus may be *trusted*. We want our system to be nearly as simple to use in this configuration as remote login using a multi-use password.

In some situations, it is useful to have one machine access another without human intervention. Assuming a remote client machine is in a secure environment so that it can be trusted with the underlying secret password, full automatic operation can be achieved.

The goal of providing automated operation implies that our system is primarily a secret based authentication system (something you know). As it requires computation to produce the one-time passwords, it is easily convertible to a token based (something you have) system.

No Secret Algorithms

The security of the authentication system must be based entirely on the secret (or secret containing token) and not on secret algorithms. A public algorithm can be evaluated by the industry, thus developing confidence in its cryptographic strength. If a system's security depends on a hidden algorithm, there is always a danger of exposure when someone who knows the secret changes jobs or loyalties.

No Stored Secrets

Storing secret keys or passwords on a host increases its attractiveness as a target, and causes a breach of security to be more wide-spread. When a common password file is used for many machines, this risk becomes even greater. On UNIX systems, the password file contains passwords already processed through a secure hash function and thus the information in this file is not directly usable to an intruder. We want our system to be no weaker[3] than this UNIX scheme, implying that no usable passwords may be stored on any host.

DESCRIPTION OF THE S/KEY SYSTEM

There are two sides to the operation of our one-time password system. On the remote client side, the appropriate one-time password must be generated. On the host side, the server must verify the one-time password. This section describes both sides, and the secure hash function on which the S/KEY authentication system is based.

3. Both UNIX password security and S/KEY authentication are vulnerable to dictionary attacks unless the passwords are well chosen.

Secure Hash Functions

A secure hash function is a function that is easy to compute in the forward direction, but computationally infeasible to invert. Consider:

$$y = f(x)$$

If f is the secure hash function with input x and output y, then computing y given x is fast and easy, but finding an x' such that

$$y = f(x')$$

for a given y is extremely difficult. Ideally, there should be no way to determine such an x' other than by trying an infeasible number of values to see which one works. If the number of possible values of x that must be tried is made large enough, then for all practical purposes the function cannot be inverted. We have chosen a hash function with 2^{64} (about 10^{19}) values.

As the basis of our secure hash function, we chose the MD4 Message Digest algorithm[4] designed by Ronald Rivest[5] of RSA Data Security Inc. MD4 accepts an arbitrary number of bits as input and produces 16 bytes of output. MD4 is fast, and so far it is believed to be secure; i.e., there is no known way of finding the input that produced a given output that is better than by exhaustively trying possible inputs.

In order to be able to apply the hash function an arbitrary number of times, we have defined our function to take 8 bytes of input and to produce 8 bytes of output. This is done by running the 8 bytes of input through MD4 and then "folding" pairs of bytes in the 16-byte MD4 output down to 8 bytes with exclusive-OR operations.

Generation of One-Time Passwords

Our one-time passwords are 64 bits in length. We believe that this is long enough to be secure and short enough to be manually entered by users (see below, *Form of Password*, for the representation) when necessary.

Preparatory Step

The input to our hash function (described above) is 8 bytes. As the client's secret password may be (should

4. Although the security of MD4 has not been broken, the newer function MD5 has been released. MD5 is slightly slower and more complex; converting to MD5 is simple, but we have chosen to continue using MD4 because of the large number of client password computing programs that have been distributed.

be) longer, a preparatory step is needed. In this step, the password 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 password on multiple machines (using different seeds), and to safely *recycle* secret passwords by changing the seed. The result of the concatenation is passed through MD4, and then reduced to 8 bytes by exclusive-OR of the two 8-byte halves. This result, called *s* below, is passed on to the generation step.

Generation Step

The sequence of one-time passwords p_i is produced by applying the secure hash function multiple times. That is, the first one-way password is produced by running the client's processed secret password *s* through the hash function some specified number of times, *N*.

$$p_0 = f^N(s)$$

The next one-way password is generated by running the user's password through the hash function only $N-1$ times.

$$p_1 = f^{N-1}(s)$$

In general, the formula is:

$$p_i = f^{N-i}(s)$$

An eavesdropper who has monitored the use of the one-time password p_i will not be able to generate the next one in the sequence (p_{i+1}) because doing so would require inverting the hash function. Without knowing the secret key that was the starting point of the function iterations, this can not be done.

System Verification of Passwords

The host is initially given p_0. When a client attempts to be authenticated, the seed and current value of *i* are passed to the client. The client returns the next one-time password. The host computer first saves a copy of this one-time password, then it applies the hash function to it.

$$p_i \;=\; f(f^{N-i-1}(s)) \;=\; f(p_{i+1})$$

If the result does not match the copy stored in the system's password file, then the request fails. If they match, then the client's entry in the system password file is updated with the copy of the one-time password that was saved before the final execution (by the server) of the hash function. This updating advances the password sequence.

Because the number of hash function iterations executed by the user decreases by one each time, at some point the user must reinitialize the system or be unable to log in again. This is done by executing the *keyinit* command, that is essentially a special version of the the UNIX *passwd* command, to start a new sequence of one-time passwords. This operation is identical to a normal authentication, except that the one-time password received over the network is not checked against the entry already in the password file before it replaces it. In this way, the selection of a new password can be done safely even in the presence of an eavesdropper. This mechanism does not defend against an active attack.

OPERATION OF S/KEY SYSTEM

Overview

The S/KEY one-time password authentication system uses computation to generate a finite sequence of single-use passwords from a single secret. The security is entirely based on a single secret that is known only to the user[5]. The single-use passwords are related in a way that makes it computationally intractable to compute any password from the preceding sequence. (It is simple, however, to compute *previous* passwords from the current one.)

The single use, or one-time, passwords replace all authentication password requirements. They are used at login time and when using the UNIX *su* command. Even when the underlying secret password is changed, only a derived one-time password crosses the network. The host computer never sees, and has no way of learning, the real secret.

As no secret algorithms are used, and the code is freely available, it is straightforward to build the S/KEY one-time password security system into any command or product requiring authentication.

Generation of S/KEY One-Time Passwords

As mentioned above, the one-time password sequence is derived from the secret password using a computer. The required computation can be executed on any PC or UNIX class machine. A supplier of credit card size devices estimated that such a device could be built for less than $30 in large quantities.

The program can also be stored on and executed from a standard floppy disk. This would allow operation on a remote computer that could not be entirely trusted not to contain a Trojan Horse that would attempt to capture the secret password[6]. It is also possible to

5. Alternatively, part of or the entire secret can be stored in a non-retrievable way, in the computing device.
6. For added security, one might prefer to boot off the floppy. The truly paranoid will worry about the integrity of the ROM.

pre-compute and print several one-time passwords that could be carried on a trip where no trusted local computation is available such as when using public workstations at a conference.

Description of Operation

The following narrative describes the procedure for logging into a UNIX system using the S/KEY one-time password system. In this example, a hand-held PC compatible computer is assumed. Note that the *sequence numbers* of successive one-time passwords decreases.

1. The user, call her Sue, identifies herself to the system by login name.
2. The system issues a challenge including the sequence number of the one-time password expected and a "seed". This "seed" allows Sue to securely use a single secret for several machines. In this example, the seed is "unix3" and the sequence number is 54.
3. Sue enters 54 and unix3 into her palm-top computer. She is prompted for her secret.
4. Sue enters her secret password that may be of any length. The palm-top computes the 54th one-time password and displays it.
5. Sue enters the one-time password and is authenticated.
6. Next time Sue wants access, she will be prompted for one-time password sequence number 53.

Semi-Automated Operation

We have built semi-automatic interfaces for clients using communications software on a MS-DOS[7] or Apple Macintosh[8] personal computer. The following example describes a client interface that runs under DOS as a Terminate and Stay Resident (TSR) program.

Consider Sue in the above example using a communications program on a MS-DOS machine. Before starting the communications program, Sue runs a program that ties itself to a *hot-key* such as function key F10. When the host issues its challenge, Sue presses the hot-key. The program then scans the screen for the challenge and extracts the sequence number and seed. It then prompts Sue for her secret password and generates the correct one-time password and stuffs it into the keyboard buffer simulating user entry of this password.

Fully automated operation is obviously possible, but it would require the client machine to *know* the secret password. This is only acceptable if the client machine is in a physically secure place.

Form of Password

Internally the one-time password is a 64 bit number providing 2^{64} possible unique one-time passwords. Entering a 64 bit number is not a pleasant task; the one-time password is therefore converted to a sequence of six short words (1 to 4 letters). Each word is chosen from a dictionary of 2048 English words thus providing a space of 2^{66} possible sequences. The contents and encoding of this dictionary are not kept secret.

ADMINISTRATION OF SYSTEM

Installation

The minimum that must be installed to use this one-time password system on a UNIX host is a replacement for the *login* command and an additional command similar to *passwd*. As with the original commands, these must run as root. In addition, it may be useful to install a one-time password version of the *su* command, a new version of *ftpd* for allowing ftp access via one-time passwords, and a command to compute one-time passwords.

Source Screening

It is frequently desirable for an installation to allow internal access with a multi-use password while requiring one-time passwords for external access. A screening table provides this function. When this table is present, login attempts that pass the screening test are permitted to use the normal password or a one-time password. Others are notified that the use of the one-time password is required.

Password echo

Normally systems disable printing during the typing of a password so that an onlooker cannot steal the password. With a one-time password, this is unnecessary. The S/KEY modification of the *login* command allows the user to turn echo on by pressing "return" at the password prompt. This makes it easier to enter the longer one-time password.

EXPERIENCE

The S/KEY authentication system has been in experimental use for off-premise access to Bellcore

7. MS-DOS is a registered trademark of Microsoft Corporation.
8. Macintosh is a trademark of Apple Corporation.

for about two years. It has been available as an *alternative* one-time password system to the users of computer resources of one research organization. This section is based on this experience as unscientificly observed by the biased eyes of the author.

Ease of Use

User reaction to the S/KEY system varied from delight to indifference. Our goal (see above, *GOALS*) of *ease of use* was not fully met because the reaction of potential users was mixed based on two factors.

Type of Terminal

Those who accessed our systems using terminal programs on personal computers were generally happy (some were enthusiastic). These people generally used one of the semi-automated client access programs. Those whose access was from non-programmable terminals (such as X-Terminals) were less satisfied. They generally had to print lists of one-time passwords and enter them as required. No one liked typing the six-word one-time passwords although some preferred it to alternative systems.

Ease of Learning

Once you know how this system works, it is very easy to use. And once you understand the underlying concept, it is easy to understand how it works. But users are generally not interested in that level of understanding. Several users stated that the system was easier to use than other systems, but took longer to learn. Several potential users never bothered to learn and chose the token authenticator in general use at Bellcore. We learned that documentation is important, and that good user instructions are hard to write.

Ease of Installation

Installing the S/KEY system requires replacing the *login* program. While the changes were straightforward, modifying the existing login is impossible for systems without access to source code. In our case, we replaced the SunOS login with a modified program from BSD UNIX.

AVAILABILITY

The server code for UNIX and the client interfaces are available **as-is** over the Internet by anonymous ftp. Some documentation in the form of *man* pages is also available. These files are available from *thumper.bellcore.com* in subdirectories of *pub/nmh*. The contents of these subdirectories are:

skey	UNIX files including source, makefile, and man pages
dos	DOS client interfaces and documentation in UNIX man-page format
mac	Macintosh client interface package

SUMMARY

Computing systems have been under increasingly sophisticated attack using dial-up and other external access ports. The one-time password technology described is a simple and effective way to keep plain text passwords out of the hands of an eavesdropper. It is more general than some other systems as it allows protected access to super-user privilege, and allows its underlying secret password to be securely changed. No authenticating secret is ever transmitted in a re-usable form.

The S/KEY one-time password system described has been in use for about two years. We have learned that ease of use and ease of learning are critical to user acceptance. S/KEY is easy to use from workstations and personal computers, but more difficult from non-programmable terminals. In the later case, a self contained token authentication system seemed preferable to some users.

One-time passwords protect only at the time of authentication. They do not protect against an eavesdropper learning the content of the monitored session. They also do not protect against false authentication using more sophisticated active attacks. For example, a legitimate user might log into a system only to have the network connection "stolen" by an intruder. This could happen immediately after login, or the intruder could wait until the legitimate user attempts to log out (to reduce the chances of detection). Attempts to defeat active attacks may require more computing power on the user end of the connection than is frequently available, certainly more than is available on a "dumb" terminal. Techniques under study include the encryption or protecting with cryptographic checksums of some or all of every packet of data exchanged.

ACKNOWLEDGMENTS

The idea behind our system was originally described by Leslie Lamport.[6] The specific system described was proposed by Phil Karn who wrote most of the UNIX software. Additional details of the design were contributed by the author and John S. Walden who wrote the initial version of the MS-DOSclient software. The Macintosh one-time password generator was written by Mark Segal, and the current MS-DOS client interfaces were written by the author.

REFERENCES

1. Eugene H. Spafford, The Internet worm program: An analysis. *Computer Communications Review* 19(1):17-57, January 1989.
2. R. M. Wong, T. A. Berson, R. J. Feiertag, "Polonius: An Identity Authentication System", *Proceedings of the 1985 Symposium on Security and Privacy*, pp. 101-107, Oakland, California, April 1985.
3. J. G. Steiner, C. Neuman, and J. I. Schiller. Kerberos: An authentication service for open network systems. *USENIX Conference Proceedings*, pp. 191-202, Dallas, Texas, February 1988.
4. Champine, G., Geer, D., and Ruh, W. "Project Athena as a Distributed Computer System", *IEEE Computer*, September 1990.
5. R. L. Rivest, The MD4 Message-Digest Algorithm, *Request For Comments (RFC) 1320*, MIT and RSA Data Security, Inc., April 1992.
6. Leslie Lamport, "Password Authentication with Insecure Communication", *Communications of the ACM* 24.11 (November 1981), 770-772.