The S/Key Password System

Chris Walsh
Computer Science and Engineering Laboratory
Electrical Engineering and Computer Science
Northwestern University

William LeFebvre[1]
Argonne National Laboratory
Decision and Information Sciences Division
Advanced Computer Applications Center

Introduction

On EECSNet, as in many computing environments, passwords provide the first line of defense against unauthorized use. Users who are able to respond with the correct password at the `Password:` prompt are presumed to be who they say they are. An obvious vulnerability springs to mind: anyone who can guess or steal a legitimate user's password is in. Guessing can be made much less probable by avoiding the selection of easily-guessed passwords. Theft can be minimized by not writing down passwords, not telling them to others, and not allowing anyone to see them when they are typed in. The `passwd` program currently installed on departmental Suns does not permit the selection of many types of bad passwords, such as words from a dictionary, and all users have been warned repeatedly about writing down their passwords or telling them to others. Presumably, users are savvy enough not to enter their password when someone is looking over their shoulder, so it would seem that barring a gradual creeping lassitude, EECSNet password security has been taken care of.

Unfortunately, this is not the case. Unlike the days of yore, when logins took place from hardwired terminals, and the only place to intercept a password was over the user's shoulder or off the note he had taped to his adm3a, today's ubiquitous interconnected networks make it possible for passwords to be grabbed as they traverse the Internet. Indeed, there have been well-publicized instances of password "sniffers" being used on major regional networks and the machines of Internet Service Providers, leading to thousands of passwords being compromised. One way to prevent such compromises in the future is for authentication to take place over an encrypted connection. United States legal restrictions on the export of cryptographic technologies, however, have retarded the development of universally available solutions on this front. As an alternative, however, one can use a scheme which makes passwords obtained through eavesdropping useless. This is the approach taken by S/KEY™.

What S/KEY Is

S/KEY is a software package developed at Bellcore. It is a *one-time password* system.

Each password used in the system is usable only for one authentication. Passwords cannot be re-used, and thus, intercepted passwords are of no utility. Moreover, knowledge of already-used passwords in a user's S/KEY password sequence provide no information about future passwords. Thus, even all of one's S/KEY passwords are "sniffed" as they transit an insecure network, they will not benefit their interceptor. CERT recommends that such a system be used in order to protect authentication data [CERT, 1994].

How It Works

A user initializes S/KEY by selecting a secret password and *n*, a number of passwords to generate. A secure hash function (currently MD4) is applied to the secret password *n* times. The result is stored on the server. When the user attempts to log in, the server issues a challenge, which is the number *n-1*. Software on the user's client machine prompts for her secret password, and applies *n-1* iterations of the hash function to it, and sends this response to the server. The server applies the hash function to this response. If the result it obtains is the same as the value it stored earlier, the authentication worked. The user is allowed in, and the server replaces the stored value with the response obtained from the client, and decrements the password counter. [Haller, 1994; Rubin, 1995]

S/KEY in the EECSNet environment

S/KEY is currently installed on an experimental basis on the geneEECSNetral-access departmental Suns: delta, arcadia, asgard, atlantis, canaan, eden, laputa, nirvana, and olympus. The CSEL staff *strongly recommends* that it be used to authenticate all logins which do not both begin and end on EECSNet. Beginning February 20th, 1996, S/Key use will be required for all non-local connections to the EECSNet general access machines. Non-local means originating outside off EECSNet.

Before you can begin to use S/KEY for authentication, however, you need to initialize the system. You also need a secure local computer equipped with the software used to generate responses from S/KEY challenges, or a printed list of one-time passwords and their corresponding challenges. The latter should be used *only* if a trusted machine is unavailable, such as while you are attending a conference. The CSEL can supply S/KEY software and documentation for Mac, PC, or UNIX® platforms, so only in fairly unusual circumstances should the use of pre-printed lists be necessary. The following paragraphs describe the steps you need to take in order to begin using S/KEY. For simplicity, we assume you will be logging into delta. The procedure is exactly the same regardless of which EECS Sun you use. Be advised, however, that (unlike our standard UNIX passwords) S/KEY passwords differ from machine to machine. The steps we describe below will need to be followed on each machine you wish to log into directly. You may prefer to always login to a single machine such as delta, and use `rlogin` from there to connect to other EECS machines. The examples presented here assume you'll be using delta, but this is not required.

S/KEY Initialization

1. Log in to delta from another EECSNet machine or terminal server.
2. Type the command `keyinit`.
3. Enter your ordinary EECSNet password when prompted.
4. Enter the S/KEY secret password of your choice when prompted. This password will not be stored anywhere, so you must remember it. The password can be of any length, and may include punctuation and spaces, as well as letters and numerals. We suggest you use a long sentence. Since this password is the key to the entire system, you *must* be performing this step from within EECSNet.
5. Enter the secret password a second time when prompted for it.
6. `keyinit` will determine the encrypted form of your password, and will store it on delta.

Here's an example. User Chris is initializing a sequence of 99 passwords on delta.

```
delta:/homes/chris[55]% keyinit
Adding chris:
Reminder - Only use this method if you are directly connected.
If you are using telnet or rlogin exit with no password and use
keyinit -s.
Enter secret password: (secret password not shown, and will not be echoed)
Again secret password: (likewise)

ID chris s/key is 99 pe61662
LOB PER RICK WINO HARK HAL
```

At this point, Chris is ready to have his delta logins authenticated via S/KEY. The procedures to be followed are described below.

Login Authentication with S/KEY

1. Connect to delta using `telnet`.

2. Enter your username at the `login:` prompt. Rather than the usual `password:` prompt, delta will respond with the S/KEY challenge, which to continue the previous example would look like this: `s/key 98 pe61662`, and then the `password:` prompt. The number 98 is the S/KEY sequence number, and "pe61662" is the encryption salt. Notice how the sequence number is one less than the initial sequence number of 99 which was used when Chris ran `keyinit`.

3. On your local computer, you need to run the S/KEY software which will compute the correct response to the challenge you were just presented. On a UNIX machine, the command is `key` followed by the second and third parts of the challenge line. Thus, the correct command to issue in order to compute the response to the challenge presented in step 2 would be: `key 98 pe61662.`

4. `key` will prompt for your secret password. Respond appropriately.

5. key will respond with six words. Return to the `telnet` session and enter them at the prompt (which is still there, awaiting your response).

Since you need to enter your secret password in order for the correct response to the S/KEY challenge to be calculated, it is imperative that you execute the `key` command on a machine whose keyboard you are at, or while operating within a trusted network. Running it over an insecure network could lead to your secret password being divulged.

Here is a full example. User Chris wants to log in to delta from his workstation waldo.

```
waldo:/homes/chris[56]% telnet delta
Trying 129.105.5.105 ...
Connected to delta.eecs.nwu.edu.
Escape character is '^]'.


SunOS UNIX (delta)

login: chris
s/key 98 pe61662
Password:
```

Now, on waldo:

```
waldo:/homes/mack23[51]% key 98 pe61662
Reminder - Do not use this program while logged in via telnet or
rlogin.
```

```
Enter secret password: (entry of secret password not shown)
GWEN RUSK SAYS WAG HUT LOGE
waldo:/homes/chris[52]% exit
```

Back on delta:

```
Password: gwen rusk says wag hut loge (S/KEY passwords are case-insensitive)

Last login: Wed Jul  5 11:47:08 from foobar.eecs.nwu.edu
SunOS Release 4.1.4 (GENERIC) #2: Tue Jun 27 00:03:32 CDT 1995
You have new mail.
delta:/homes/chris[51]%
```

The process of using `key` to calculate the proper response is extremely simple when one can cut and paste text from one window into another. In the example above, if the `telnet` to delta is running in an `xterm` window on waldo, it is extremely convenient to cut the "key 98 pe61662" substring from the challenge, and paste it into a shell window on waldo, thereby making the calculation of the response take only a few mouse clicks. Once `key` has calculated the response, it can be cut from the `xterm` and pasted in at the waiting delta `Password:` prompt just as easily. X11 users will find the `winkey` handy — it runs `key` in an `xterm` window for you.

The process is equally simple for DOS, Windows, and Mac users. Please see the **Software Availability** section later in this document for information concerning where you can get S/KEY software and documentation for your DOS, Windows, or Mac computer.

Establishing a New Password Sequence

In the example just discussed, Chris used `keyinit` to initialize a sequence of 99 passwords. Eventually, he'll run out (and thus be locked out!) unless he can generate a new sequence. This is done using `keyinit`. By default, a sequence of 99 passwords is generated. You may wish to increase this number in order to avoid having to run `keyinit` too frequently. As noted above, it is imperative that your secret password not traverse any potentially insecure networks, so you will want to run `keyinit` from within EECSNet. Occasionally, however, this may prove impractical. You may be away from Evanston for an extended period, yet still connecting to delta on a daily basis from a remote workstation. Happily, there is a way to use `keyinit` which will allow you to initialize a new password sequence, but which does *not* require that your secret password travel over an untrusted network. This technique uses *local* software to do the encryption of your secret password. You then supply the result to `keyinit`. Here's how it is done.

1. Type the command `keyinit -s`

2. It will tell you what the old salt is, then prompt for a new sequence count. Enter a desired number of passwords (eg., 1000).

3. `keyinit` then prompts for a new key, and provides a default response. Accept this default.

4. `keyinit` then provides a challenge of the same type seen during the login sequence. Run your *local* S/KEY encryption program (eg., `key`) just as if you were logging in.

5. Your local invocation of `key` will prompt for your secret password. Enter it, and you will get an encrypted response.

6. Enter the encrypted response obtained in the previous step at the waiting *remote* invocation of `keyinit`. You will now have successfully generated a new sequence of passwords.

Here is an example. User Chris, separated from delta by an insecure network, needs to establish a new S/KEY password sequence from his workstation *cicero*.

```
delta:/homes/chris[52]% keyinit -s
Updating chris:
Old key: pe61662
Reminder you need the 6 english words from the skey command.
Enter sequence count from 1 to 9999: 1000
Enter new key [default pe61663]:
s/key  pe61663
s/key access password:
```

Then...

```
cicero.spudly.com:/usr/chris[57]% key 1000 pe61663
Reminder - Do not use this program while logged in via telnet or
rlogin.
Enter secret password:
PEA TUB YALE BOWL GULF JUTE
cicero.spudly.com:/usr/chris[58]%
```

Back on delta...

```
s/key access password: PEA TUB YALE BOWL GULF JUTE
```

This completes the process. The next time Chris tries to log in to delta, he will be challenged for the 999th password in the sequence.

Creating a List of Passwords

Occasionally, such as when you are travelling, you will have no trusted local host upon which to run the `key` command or its equivalent. Under such circumstances, you can run `key` prior to your departure, and have it generate a list of passwords which you can refer to during your trip. This list should be treated with the utmost care. No identifying information should appear on it, and it should be only as long as is absolutely necessary.

To generate the list, you need to know the current key and sequence number for your S/KEY password sequence. This is the information presented to you as a login challenge. It is maintained in the file `/etc/skeykeys`. You can extract your information from this file like using the `keyinfo` command. The first field is the sequence number, and the second is the key. These will be used in conjunction with your secret password to generate the list of one-time passwords for your trip. Here is an example. Chris is going to a conference, and needs to log in once a day. He therefore generates seven passwords on the machine into which he will be telnetting while away.

```
waldo:/homes/chris[57]% keyinfo
chris 1000 pe61664
waldo:/homes/chris[58]% key -n 7 1000 pe61664
Reminder - Do not use this program while logged in via telnet or
rlogin.
Enter secret password: (not shown)
994: REIN SAG WART NOVA GORE NE
995: OW OWE SARA OAT TUNA GREY
996: RAYS GALE DRY ROAD RISK VETO
997: RAY RUSS SIT HYDE LOGE LAP
998: WOK BLUM YOGA RUTH SING RUNG
999: BE RISK BOLT HEN COAL ROSS
1000: JOKE WINE HOVE AUNT TIER DRUG
```

These can be printed off, and used while travelling. When `login` presents its numbered S/KEY challenge, Chris can simply look up the password corresponding to it, and enter it. The care with which such lists of passwords should be guarded cannot be overemphasized. Immediately contact the CSEL staff if you have lost such a list.

Software Availability

S/KEY software for your PC, Mac, or UNIX machine is either already installed (in the case of Departmental Suns), or can be provided by e-mail requests directed to `root@eecs.nwu.edu`. We have binaries for PCs and Macs, and can supply source code for various flavors of UNIX, including Linux. Specifically, the CSEL recommends the use of the following packages:

`KEYAPP.EXE` is a Microsoft Windows application which computes S/KEY™ one-time passwords, given an S/KEY challenge and a user's secret password. Its installation and use are described in the document *Using `KEYAPP.EXE`*, available in the CSEL office.

Macintosh users should refer to the document *Using S/KEY for the Mac*, which describes S/KEY software for the Macintosh platform.

`TERMKEY.EXE` is a TSR which DOS users can "pop-up" on demand in order to compute S/KEY one-time passwords. Refer to the document *Using TERMKEY.EXE*, available in the CSEL office, or to the on-line man page for `termkey` for details (although it is a DOS program, there is a UNIX-style man page for termkey).

The software mentioned above is available on diskettes in the CSEL office, as well as in the directories `/usr/local/lib/skey/mac`, `/usr/local/lib/skey/dos`, or `/usr/local/lib/skey/windows`.

The source code for S/KEY is located in `/vol/src/logdaemon-4.9/skey`.

Additional information is available on-line at:
`http://web.eecs.nwu.edu/~mack23/skey.html`

Future Developments

Since passwords traverse networks for much more than logins, the S/KEY approach needs to be applied to much more than `/bin/login`. We have installed S/KEY capable versions of `rshd`, `rexecd`, and `ftpd` in order to provide this additional protection. They behave just as `login` does -- by providing a challenge for which you must supply an appropriate response. In addition, while S/KEY use is currently optional, we are considering requiring the use of S/KEY authentication for all connections to EECSNet which do not originate on EECSNet.

We also plan to acquire an encrypting version of telnet as soon as possible. This will enable users to encrypt their telnet connections *in toto*, using triple DES. We will also be experimenting with an encrypting session layer, but its deployment is impossible to predict at this time. Any modification to the current access policy will be announced in advance in the `eecs.announce` newsgroup.

References

[CERT, 1994] Computer Emergency Response Team, CERT Advisory 94:01, Carnegie Mellon University, Pittsburgh, Feb 3, 1994.

[Haller, 1994] Neil M. Haller, "The S/KEY One-time Password System". *Proceedings of the Internet Society Symposium on Network and Distributed System Security*, San Diego, Feb 3, 1994. (ftp://thumper.bellcore.com/pub/skey/docs/ISOC.symp.ps)

[Rubin, 1995] Aviel D. Rubin, "Independent One-Time Passwords", *Proceedings of the Fifth USENIX UNIX Security Symposium*, Salt Lake City, June 5-7, 1995.

Notes

1. This document is based in part on a similar document entitled *Improving Remote Security with S/Key*, and distributed internally at Argonne National Laboratory. Therefore, LeFebvre is credited as a co-author. Any errors may be credited to Walsh.