Technical Correspondence

PASS-ALGORITHMS: A USER VALIDATION SCHEME BASED ON KNOWLEDGE OF SECRET ALGORITHMS

Haskett's article on "pass-algorithms" [1] presents a stimulating idea: the use of a secret *algorithm* in place of a secret password. We sympathize with the need for an "immediately implementable system that can be enforced on an as-needed basis," but we do not believe that pass-algorithms are as attractive as the article suggests.

It is not at all clear that one can construct a large number of pass-algorithms, each of which is easy to remember. Certainly the "large" and "highly parametrized" libraries of pass-algorithms mentioned by Haskett do not sound easy to remember. The hints he gives about constructing such algorithms make us wonder if a legitimate user would ever successfully login, since these might include addressing modes, changes on odd-numbered days, and different responses at different sites. Compounding the difficulty of a legitimate user's login attempts is the suggestion that the login program lead the user on a wild goose chase when it detects some illegal response. It might be possible for a legal user to spend many minutes before even realizing that something had gone wrong.

A conventional secondary password is immediately vulnerable to an eavesdropper. Haskett's proposal has the advantage of less vulnerability of this kind, but with his early examples an eavesdropper could easily deduce the algorithm. Unless one moves to complex *user specific* pass-algorithms, the eavesdropper would know the pass-algorithm for everyone. Thus, the class of easily remembered pass-algorithms should also be *hard to deduce.* (See [2] for a safeguard against eavesdropping based on one-way functions.)

Some of Haskett's ideas are reasonable, but have nothing to do with pass-algorithms. For example, a user should not be validated until a secondary password or pass-algorithm has been presented. Whichever of the two is employed, the date and time, etc., could be recorded after accepting the primary password.

One sees from examples late in [1] that a pass-algorithm is more or less a set of unconventional responses demanded of a user, perhaps in an interactive session. This could work well initially, but as word gets out, an intruder might have a better chance of guessing an *easily remembered* unusual response than an equally easily remembered traditional password.

Our own recommendation is to use a fairly long, easily remembered string as a secondary password. This might seem contradictory, but one can take a familiar phrase and warp it slightly in some silly way, for example, "A snitch in thyme," "Rolling bones gather," etc. In this way, one can choose from a large space of passwords which are all fairly easy to remember. Moreover, such passwords would not occur in any standard lists of words or phrases. The one disadvantage is that these require accurate keying in, but the same is true of Haskett's longer interactive sessions.

If greater security is desired, one should also use a piece of hardware in the user's possession, or a personal feature of the user, or both.

Paul Putter
Neal Wagner
Department of Mathematics and Computer Science
Drexel University
Philadelphia, PA 19104

REFERENCES

1. Haskett, J.A. Pass-algorithms: A user validation scheme based on knowledge of secret algorithms. *Commun. ACM 27*, 8 (Aug. 1984), 777–781.
2. Lambert, L. Password authentication with insecure communication. *Commun. ACM 24*, 11 (Nov. 1981), 770–772.

AUTHORS' RESPONSE

Let me explain the concept behind pass-algorithms in a different manner from that which appeared in the original article.

In discussing secret messages, cryptologists talk of a plaintext message P that is encrypted with a key k and an algorithm E to produce a ciphertext (i.e., a secret message) C.

$$C = E_k(P)$$

If we look at computer login passwords in this framework, then

$$P = \text{password}$$

$$k = \text{null}$$

$$E = \text{the unity operator}$$

$$\text{and} \quad C = E_k(P) = 1_{\text{null}}(\text{password})$$

$$\text{or} \quad C = \text{password}.$$

That is, the secret message is identical to the plaintext message. That hardly seems a good way to keep a secret.

Pass-algorithms are an attempt to recognize that in login validation, k need not be null and E need not be the unity operator. k (the key) might be a (sub)string of the prompt sent to the user's terminal and E (the encryption algorithm) might be one of those suggested below. Note that since at least the time of Caesar, cryptologists have recognized that the key plays a major

role. In fact, the Data Encryption Standard makes the encryption algorithm public and depends on the key for secrecy. A particularly easy-to-use pass-algorithm might be used the same way, with the prompt providing part of the key, and each user remembering another, unique, part.

Note that I do not suggest the replacement of the operating system's primary password protection with pass-algorithms. I would rather see them used as an additional layer of protection.

Do Snow, Walker (writers of earlier letters commenting on pass-algorithms), Putter, and Wagner think pass-algorithms have value? Apparently not a great deal.

Do I believe they have value? Yes, for three reasons. First, they allow us to break the restrictive bonds on our thinking that have dictated that the typed password (perhaps after encryption) must identically match the password stored in the valid-user file. Second, pass-algorithms might allow us to use the work of cryptology to investigate new login security techniques. Finally, pass-algorithms are cheap to install. No hardware need be purchased. They can be installed with, at most, a few hours time. No invaded installation can claim that it could not afford additional login security.

Do others believe pass-algorithms have value? Apparently so. Since the article on pass-algorithms was published, I have found perhaps a half-dozen ads for security devices that use the same concept as pass-algorithms. In each of these hardware and software devices, the computer sends a prompt to the user's terminal, the user's handheld device encrypts the prompt, and the user keys in the device's resultant message for the computer to validate. In one of these devices, the customer can even specify the encryption algorithm. Apparently, some people believe in the concept behind pass-algorithms enough to invest a great deal of money. The difference between these devices and pass-algorithms is that the devices are executed in the hardware and pass-algorithms are executed in the human mind. The concept is the same. In keeping with the security view of things that you have, things that you are, and things that you know, these devices are things that you have and pass-algorithms are things that you know.

Is it possible to find simple, effective algorithms that can be executed in the human mind? I think probably so. If I want to protect myself from the casual bandit, I need only find some bit of esoterica from my own hobbies or fields of interest and build a key and an encryption algorithm around it.

For example, a systems programmer might create his own personal pass-algorithm that would (on a particular login) write the following to the terminal:

```
d2G5;c
u$$i7Vf:*
```

Having devised the pass-algorithm, he knows that the 2 and 5 in the first line tell him to key in the string in the second line beginning with the fifth character having a length of 4. (The 4 is a second key he must remember. It purposely does not appear in the prompt for additional security.) The proper response this time is 7Vf:. The next time he logs in, the two (or more) strings will contain other, randomly chosen characters and the response will be different.

A person interested in CAI might select his personal pass-algorithm for this semester from the departmental pass-algorithm library. This pass-algorithm is based on a children's board game called "How the West Was Won" that has been written for a number of CAI systems. The game requires the players to provide the mathematical operators needed to manipulate randomly generated numbers that will result in their landing on "good" squares and thus win the game. In this adaptation, all nonalphabetic characters are separators. When the programmer selected the pass-algorithm, he provided the parameters necessary to specify printing three numbers in the range (−10 to +10), with division and parentheses not permitted. When he or she logs in, the prompt might be:

```
9b2_8,10
```

The proper response for this prompt is −*++ because $-9 * 2 + 8 + 10 = 0$ as the selected algorithm and parameters require.

The system manager of a computer used by linguists might create an installation pass-algorithm that randomly selects a question written in any one of a number of human languages from a question database and writes it to the terminal. The proper response might be a single word correct answer written in the world's most popular language in January, the world's second most popular language in February, etc.

Let us adapt a couple of ideas used by Dan Crutcher and William Thomas of Actors Theatre of Louisville in their 1983 fund-raising mystery, "Three Stars Fall."

A ham radio operator might write a random character string to the terminal and require that the proper Morse code "dots and dashes" be typed within an appropriately short period of time. To make it more confusing to an onlooker, "dots" might be any key on the left half of the keyboard and "dashes" might be any key on the right half of the keyboard.

And how about the telephone dial as an encryption aid? It can be used as an ever-present table to allow substitution of letters for numbers and numbers for letters.

Are these algorithms so difficult that the legitimate user would be prevented from logging in? I do not think so. Are they too simple? Only if inappropriate for the bandits you're trying to protect yourself from.

Is it possible to find a really good algorithm that can be executed in the human mind? (Perhaps called DES/carbon?) I do not know. But the problem sounds as exciting to me as the search for Diffie–Hellman trapdoor functions.

Jim Haskett
Bloomington Academic Computing Center
Indiana University
Bloomington, IN 47405