PROJECT MAC
Computer Systems Research Division

June 4, 1973
Request for Comments No. 25

"SECURITY CODE" by George Purdy
from J. H. Saltzer

The enclosed note by George Purdy of the University of Illinois describes an interesting method of one-way encrypting passwords. Particularly of interest is his attempt to analyze the work factor required to break the code. The interesting question is whether or not the work factor calculation considers all possibilities.

by

George Purdy
University of Illinois
Center for Advanced Computation
May 21, 1973

SECURITY CODE

In what follows, we describe a security coding system which enables the user to log onto a system by using his code number X_i which is immediately transformed into a pseudo code word $Y_i = f(X_i)$ by the machine. Even though Y_i and f are public knowledge it is not possible to log onto the system without knowing X_i, and the equation $Y = f(X)$ cannot be solved for X, even if Y is known.

§1 the function f(X)

The code is of the form $Y = f(X)$, and the cracker knows f. We have arranged that the equation $f(X) = Y$ is very unlikely to be solved in fewer than 10^6 seconds of processor time. The function f is a polynomial modulo a prime P.

$$f(X) = X^n + a_4 X^m + a_3 X^3 + a_2 X^2 + a_1 X + a_o \pmod{P},$$

where $P = 2^{64} - 59$, $m = 2^{24} + 3$, $n = 2^{24} + 17$,

and the a_i are 19 - digit numbers. The cracker has essentially two approaches for solving $Y = f(X)$ given Y. He can use trial and error, or Berlekamp's and similar algorithms. It was necessary to make n and P fairly large in order to defeat both approaches.

§2 Berlekamp's Algorithm as a Threat

Berlekamp's method for completely factoring polynomials modulo P can be applied to the polynomial $f(X) - Y = g(X)$ and it requires [1] at least $n^3 (\log P)^2$ operations. There are no algorithms known which are faster than this, so it seems safe to say that $n^2 (\log P)^2$ operations are required to find just a single root.

Now $n \cong 10^7$ and $P \cong 10^{19}$, so more than 10^{16} operations are required. Let us say that the speed s of the crackers machine is 10^{10} operations per second (faster than ILLIAC IV). Then it would still take more than $T = 10^6$ seconds $\approx$ two weeks.

§3 the Trial and Error Threat

We assume that the cracker has a list of all assigned Y_i and he keeps trying values of X until $f(X) = Y_i$ for some i. Let c be the number of X_i assigned to users. A theorem of Lagrange guarantees that no more than n of the X's will map into one Y. Thus, if the cracker chooses an X at random between 1 and P, his probability of success is at most $\frac{cn}{P}$.

The probability P_k of failure on the kth trial is at least

$$1 - \frac{cn}{P - k + 1} \approx 1 - \frac{cn}{P} = a.$$

the expected number of trials K_e before success is

$$K_e = \sum_{k=1}^{\infty} K\ (P_1\ P_2\ \ldots\ P_k) = \sum_{k=1}^{\infty} k\ a^k = \frac{a^2}{(1 - a)^2} \cong \frac{P^2}{c^2 n^2}.$$

The expected cracking time is $T_e = \frac{K_e\ Q}{s} = \frac{P^2\ Q}{c^2 n^2 s}$ where Q is the number of operations needed to compute f(X).

Even if Q = 1, and $s = 10^{10}$, c = 1000, we have

$$T_e \cong \frac{10^{38}}{10^6\ 10^{14} \times 10^{10}} = 10^8$$

seconds, or about 3 years.

§4 Implementation

The implementation of the algorithm for f uses multiprecision arithmetic in the form of some Fortran subroutines. It is operational on a PDP-10 and requires about half a second to compute f(X).

§5 Remarks about the use of the code

When user-codes are assigned, a random number generator should be used to choose an X_i and then one should verify that $f(X_i) \neq Y_j$ for any previous Y_j. This is extremely unlikely, but it could happen.

[1] D. E. Knuth, The Art of Computer Programming, Vol. 2, p. 381-397, Addison-Wesley, 1969.