J. Cryptology (1989) 1: 151–158

Journal of Cryptology

A Study of Password Security[1]

Michael Luby[2]
International Computer Science Institute,
Berkeley, CA 94704, U.S.A.
luby@icsi.berkeley.edu

Charles Rackoff[3]
Computer Science Department, University of Toronto,
Toronto, Ontario M5S 1A4, Canada
rackoff@theory.toronto.edu

Abstract. We prove relationships between the security of a function generator when used in an encryption scheme and the security of a function generator when used in a UNIX-like password scheme.

Key words. Password security, UNIX, Pseudorandom function generator.

1. Introduction

Our work is motivated by the question of whether or not the password scheme used in UNIX is secure. The following password scheme is a somewhat simplified version of the actual password scheme used in UNIX. We feel that this simplified version captures the essential features of the actual password scheme used in UNIX. The first time a UNIX account is used the user enters his user name together with a randomly chosen password. The system creates an "encryption" of the password using the Data Encryption Standard (DES) and stores the encryption (not the password) together with the user name in a password file. Thereafter, whenever the account is used the user enters his user name and password, the system computes the encryption of the password and only allows the user to successfully log in if the encryption matches the entry stored with the user name in the password file.

The system model we assume allows any user to read the password file, but guarantees that the password file cannot be changed by unauthorized users. We do not attempt to justify this model of the system, but only remark that perhaps the

[1] Date received: August 19, 1988. Date revised: January 9, 1989.

[2] On leave of absence from the Computer Science Department, University of Toronto. Research partially supported by the Canadian Natural Sciences and Engineering Research Council Operating Grant A8092 and by a University of Toronto research grant.

[3] Research partially supported by the Canadian Natural Sciences and Engineering Research Council Operating Grant A3611.

reasoning behind this model is that an unauthorized user may be able to read the password file when the system is unprotected (e.g., during a crash) but the system is able to keep enough backup copies of the password file so that even if an unauthorized user can write to one copy of the file, he will not be able to update all copies. The password scheme is secure with respect to this model if any unauthorized user who has a copy of the password file cannot generate a password whose encryption is stored in the password file.

The following is a more complete description of the password scheme discussed above. The password is a bit string x of length 56 and the encryption of x is a bit string y of length 64, where y is DES evaluated on a standard message (which is the bit string consisting of 64 zeros) using key x. It is stated informally in [D] that this password scheme is secure if DES is secure when used as a private key cryptosystem. We formally investigate this question by the following approach. Since we cannot even prove that DES is secure in any formal sense when used in a block private key cryptosystem, we study the security of a UNIX-like password scheme when in place of DES we use a pseudorandom function generator.

Let N be the set of positive integers. A password scheme g with password length function $l: N \mapsto N$ is a family of functions $g = \{g^n: n \in N\}$ such that, for all $n \in N$, $g^n: \{0, 1\}^{l(n)} \mapsto \{0, 1\}^n$. The encryption of password $x \in \{0, 1\}^{l(n)}$ is $g^n(x)$. A function generator f with key length function $l: N \mapsto N$ is a family of functions such that for each positive integer n, every $x \in \{0, 1\}^{l(n)}$ is a key that indexes a function $f_x: \{0, 1\}^n \mapsto \{0, 1\}^n$. The UNIX-like password scheme g based on f is defined as follows: the encryption of password $x \in \{0, 1\}^{l(n)}$ is $g^n(x) = f_x(0^n)$.

Very informally stated, what we prove is that if a pseudorandom function generator with key length function l satisfying $l(n) \leq n + O(\log n)$ is used in a UNIX-like password scheme, then the password scheme is secure. On the other hand, we describe a pseudorandom function generator with key length $l(n) = n + h(n)$ (where $h(n)$ is any function that grows asymptotically faster than $\log n$, i.e., the limit as n goes to infinity of $\log n/h(n)$ is zero) such that the UNIX-like password scheme using it is not secure. The implication of this latter result is that if the password is much longer than the encryption of the password, then, even if we have a function generator which is secure when used as a block private key crytosystem, the UNIX-like password scheme using the function generator may not be at all secure. As a more concrete example, we show that a modified version of DES, which we (and some other cryptographers) believe is even more secure than DES when used in a block private key cryptosystem, is not at all secure when used in the UNIX-like password scheme. This is a lesson against the blind philosophy of taking something which is proven secure in one setting and using it in a different setting without a formal investigation of its security in the different setting.

An earlier version of this work appeared in [LR3].

2. Definition of a Pseudorandom Function Generator

Let N be the set of all positive integers. For all $n \in N$, let F^n be the set of all functions from $\{0, 1\}^n$ to $\{0, 1\}^n$. All random choices are with the uniform probability distribution.

Function Generator Definition. A *function generator* f with key length function l: $N \mapsto N$ is a family of functions where, for all $n \in N$, every key $x \in \{0, 1\}^{l(n)}$ indexes a function f_x: $\{0, 1\}^n \mapsto \{0, 1\}^n$. The additional requirements on a function generator are that the key is not too long, i.e., $l(n)$ is upper bounded by a polynomial in n, and that the function can be evaluated in polynomial time, i.e. there is a polynomial-time algorithm that on input $x \in \{0, 1\}^{l(n)}$ and $\alpha \in \{0, 1\}^n$ outputs $f_x(\alpha)$. We denote by f^n the set of $2^{l(n)}$ functions in the family from $\{0, 1\}^n$ to $\{0, 1\}^n$. Note that f^n is a very small subset of F^n.

Intuition. We view x as a randomly chosen private key of length $l(n)$. The function generator f is *pseudorandom* if there is no polynomial time in n algorithm which, for infinitely many n, is able to distinguish even slightly whether a function was randomly chosen from f^n or from F^n after seeing polynomial in n input/output pairs of the function, even when the algorithm is allowed to choose the next input based on all previously seen input/output pairs.

Pseudorandom function generators were first defined by Goldreich *et al.* [GGM], who prove that the existence of such a generator is implied by the existence of a pseudorandom bit generator [BM], [Y]. Other results and discussions about these generators appear in [LR1], and [LR2].

Definition of a Boolean Oracle Circuit. A *Boolean oracle circuit* C_n *for* f^n is an acyclic circuit that contains Boolean gates of type *and*, *or*, and *not*. In addition, C_n also contains *oracle gates*. Each oracle gate has an input and an output that are both strings of length n. All of the oracle gates are to be evaluated using the same function selected from F^n, that for now is left unspecified and is to be thought of as a variable of the circuit. The output of C_n is a single bit. The *size of* C_n is the total number of gates plus the number of connections between gates. We let $\Pr[C_n(F^n)]$ be the probability that the output bit of C_n is 1 when a function is randomly chosen from F^n and used to evaluate the oracle gates. We let $\Pr[C_n(f^n)]$ be the probability that the output bit of C_n is 1 when a key $x \in \{0, 1\}^{l(n)}$ is randomly chosen and f_x is used to evaluate the oracle gates. The *distinguishing probability for* C_n is $|\Pr[C_n(F^n)] - \Pr[C_n(f^n)]|$.

Definition of a Distinguishing Circuit Family. A *distinguishing circuit family for* f is a family $C = \{C_n: n \in I\}$, where I is an infinite subset of N and, for each $n \in I$, C_n is a Boolean oracle circuit for f^n. In addition, for some pair of constants s and c, for each $n \in I$, the size of C_n is at most n^s and the distinguishing probability for C_n is at least $1/n^c$.

Definition of Pseudorandom. A function generator f is *pseudorandom* if there is no distinguishing circuit family for f.

3. Definition of a Secure Password Scheme

Password Scheme Definition. $g = \{g^n: n \in N\}$ is a *password scheme* with password length function l: $N \mapsto N$ (where $l(n)$ is upper bounded by a polynomial in n) if, for each $n \in N$, g^n is a function from $l(n)$ bits to n bits.

Intuition. For each $n \in N$, the password is of length $l(n)$ and the encryption of the password is of length n.

Definition of a Password Finding Circuit Family. A *password finding circuit family for* g is a family $A = \{A_n : n \in I\}$, where I is an infinite subset of N and, for each $n \in I$, A_n is a Boolean circuit with n input bits and $l(n)$ output bits. The *success probability of* A_n is the probability that A_n outputs a y such that $g^n(y) = g^n(x)$ on input $g^n(x)$, where $x \in \{0, 1\}^{l(n)}$ is randomly chosen. There are two constants s and c such that, for all $n \in I$, the size of A_n is at most n^s and the success probability of A_n is at least $1/n^c$.

Security Definition. A password scheme g is *secure* if there is no password finding circuit family for g.

Comment. Informally, we want the security of g to reflect the fact that no polynomial-size family of circuits, given the encryptions of a polynomial number of randomly chosen passwords, can produce even one password which has the same encryption as one of the given encryptions. It can be easily shown that if g is secure in the sense defined above, then g is secure in the following alternative sense of security, which reflects this informal notion of security. A password scheme g is *secure* if, for all functions b: $N \mapsto N$ such that $b(n)$ is upper bounded by a polynomial in n, there is no polynomial-size family of Boolean circuits which, for infinitely many values of $n \in N$, on input $b(n)$ encryptions $g^n(x_1), \ldots, g^n(x_{b(n)})$ of randomly chosen passwords $x_1, \ldots, x_{b(n)} \in \{0, 1\}^{l(n)}$, produces an output y such that $g^n(y) = g^n(x_i)$ for some $i = 1, \ldots, b(n)$ with probability greater than $1/n^c$ for some constant $c > 0$.

UNIX-like Password Scheme. Let f be a function generator with key length function l. The *UNIX-like password scheme for* f is a password scheme g with password length function l, where for each $n \in N$ the encryption $g^n(x)$ of a password $x \in \{0, 1\}^{l(n)}$ is $f_x(0^n)$.

4. The Main Theorem

Theorem. *If f is a pseudorandom function generator with key length function l such that $l(n) \leq n + d \log n$ for some constant d and for all sufficiently large $n \in N$, then the UNIX-like password scheme g for f is secure.*

Proof. We assume for contradiction that the password scheme g is not secure. Thus, there is a password finding circuit family $A = \{A_n : n \in I\}$ for g. Let c be the success probability constant associated with A. For each $n \in I$, let $\varepsilon(n) \geq 1/n^c$ be the success probability for A_n. We show how to construct from A a distinguishing circuit family $C = \{C_n : n \in J\}$ for f, where J is an infinite subset of I.

For each $n \in I$, define $\varepsilon'(n)$ to be the probability that A_n outpu $\_$ $y \in \{0, 1\}^{l(n)}$ such that $g^n(y) = z$ when the input to A_n is a randomly chosen $z \in \{0, 1\}^n$. Intuitively, $\varepsilon'(n)$ is equal to $\varepsilon(n)$ if the distribution $g^n(x) = f_x(0^n)$, defined by randomly choosing a password $x \in \{0, 1\}^{l(n)}$, is uniform on $\{0, 1\}^n$. For each $n \in I$, we consider two cases:

Case 1: $\varepsilon'(n) \leq \varepsilon(n)/2$. In this case $\varepsilon(n) - \varepsilon'(n) \geq 1/2n^c$. The idea in this case is that C_n distinguishes f^n from F^n due to the fact that the distribution $f_1(0^n)$, defined by randomly choosing a password $x \in \{0, 1\}^{l(n)}$ and setting $f_1 = f_x$, is not the uniform distribution on $\{0, 1\}^n$, whereas the distribution $f_1(0^n)$, defined by randomly choosing $f_1 \in F^n$, is the uniform distribution on $\{0, 1\}^n$. Moreover, A_n is used to distinguish between these two distributions. The Boolean oracle circuit C_n is defined as follows:

Circuit C_n. There is one oracle gate in C_n, which is to be evaluated using a function denoted f_1. The input to the oracle gate is 0^n. Let z denote the output of the oracle gate. C_n simulates A_n on input z and the simulation produces an output y. Circuit C_n computes $\alpha = f_y(0^n)$. If $z = \alpha$, then C_n outputs 1, otherwise C_n outputs 0.

If x is randomly chosen from $\{0, 1\}^{l(n)}$ and $f_1 = f_x$, then the probability that C_n outputs 1 is $\varepsilon(n)$. On the other hand, if f_1 is randomly chosen from F^n, then the probability that C outputs 1 is $\varepsilon'(n)$. Thus, the distinguishing probability of C_n is at least $\varepsilon(n) - \varepsilon'(n) \geq 1/2n^c$.

Case 2: $\varepsilon'(n) \geq \varepsilon(n)/2$. In this case $\varepsilon'(n) \geq 1/2n^c$. The Boolean oracle circuit C_n is defined as follows:

Circuit C_n. There are two oracle gates in C_n, one with input 0^n and output z and the other with input 1^n and output z' (any fixed input different than 0^n for the second oracle gate suffices), which are to be evaluated with the same function denoted f_1. Circuit C_n simulates A_n on input z and the simulation produces an output y. Then C_n computes $f_y(1^n)$ and outputs 1 if this is equal to z' and 0 otherwise.

As we prove in the claim below, because of the length restriction on passwords ($l(n) \leq n + d \log n$), if f_1 is chosen by randomly choosing $x \in \{0, 1\}^{l(n)}$ and setting $f_1 = f_x$, then, with probability at least $1/2n^{c+d}$, $y = x$. If $y = x$, then it is also the case that $f_y(1^n) = f_1(1^n) = z'$, in which case C_n outputs 1. On the other hand, if f_1 is chosen randomly from F^n, then $z = f_1(0^n)$ and $z' = f_1(1^n)$ are independent and uniformly distributed in $\{0, 1\}^n$. Thus, independent of the value y produced by A_n on input z, it is only with probability $1/2^n$ that $f_y(1^n) = z'$ and C_n outputs 1. Thus, C_n distinguishes between F^n and f^n with probability at least $1/2n^{c+d} - 1/2^n \geq 1/4n^{c+d}$ for sufficiently large n.

Claim. *When $x \in \{0, 1\}^{l(n)}$ is randomly chosen, then the probability that A_n on input $z = f_x(0^n)$ outputs a y such that $y = x$ is at least $1/2n^{c+d}$.*

Proof of Claim. Let S be the set of strings $x \in \{0, 1\}^{l(n)}$ such that A_n on input $z = f_x(0^n)$ outputs y such that $y = x$. Then $|S|/2^{l(n)}$ is the probability in question. Let M be the set of strings $z \in \{0, 1\}^n$ such that A_n on input z outputs a y such that $f_y(0^n) = z$. Thus, for each $z \in M$, A_n on input z outputs a unique $y \in S$, and every $y \in S$ is the output of A_n for some $z \in M$. Thus, A_n defines a one-to-one onto map

from M to S and $|S| = |M|$. Clearly, $|M|/2^n = \varepsilon'(n)$. Thus, since $l(n) \leq n + d \log n$,

$$\frac{|S|}{2^{l(n)}} = \frac{|M|}{2^{l(n)}} \geq \frac{\varepsilon'(n)}{2^{d \log n}} \geq \frac{1}{2n^{c+d}}.$$ □

From Case 1 and Case 2 it is easy to see that there is an infinite $J \subseteq I$ such that, for all $n \in J$, C_n distinguishes f^n from F^n with probability at least $1/4n^{c+d}$. Thus, the function generator f is not pseudorandom. □

Comment. There is another definition of security where "probabilistic polynomial-time algorithm" is substituted for "polynomial-size family of circuits" in the definition. Our theorem is true (using a similar proof) with respect to this definition when "probabilistic polynomial-time algorithm" is substituted for "polynomial-size family of circuits" in the definition of pseudorandom function generator.

5. The Password Should Not Be Too Long

In this section we give examples which show that the theorem proved in the previous section is the strongest general theorem possible with respect to the length of the password and the encryption of the password. Our first example shows why a pseudorandom function generator with a long key length cannot necessarily be used to produce a secure UNIX-like password scheme. The second example, which is a very practical example, demonstrates that a cryptosystem which we (and some other cryptographers) believe to be even more secure than DES when used as a private key block cryptosystem is totally insecure when used as a UNIX-like password scheme.

Example 1. Let f be a pseudorandom function generator with key length l, where, for all $n \in N$, $l(n) = n$. We define a function generator g in terms of f as follows. The key length function for g is l', where, for all $n \in N$, $l'(n) = n + \log^2 n$. (Any function $h(n)$ such that $2^{h(n)}$ is not upper bounded by a polynomial in n can be substituted for $\log^2 n$.) For each $n \in N$, let x be a string of length $l'(n)$, which is the concatenation of a string x_1 of length $\log^2 n$ and a string x_2 of length n. For all strings $\alpha \neq 0^n$ of length n, $g_x(\alpha) = f_{x_2}(\alpha)$. Define $g_x(0^n)$ to be $f_{x_2}(0^n)$ if $x_1 \neq 0^{\log^2 n}$ and to be x_2 if $x_1 = 0^{\log^2 n}$. It is not hard to prove that g is pseudorandom, because the only time g is any different than f is in the event that x_1 just happens to be equal to $0^{\log^2 n}$, which only happens with the tiny probability $1/n^{\log n}$. On the other hand, if g is used in a UNIX-like password scheme, then the resulting password scheme is totally insecure. To see this, note that for any encrypted password y of length n, the password consisting of $0^{\log^2 n}$ concatenated with y always encrypts to y.

Example 2. The key for DES is 56 bits long. This key is expanded in a predetermined way into 16 keys each of length 48, and the 16 keys are used in the 16 levels of encryption in DES. Let MDES (mnemonic for Modified Data Encryption Standard) be the same as DES except that it uses 16 independent keys of length 48, and these 16 keys are used in the 16 levels of DES, i.e., MDES can be thought of as $f_{x_{16}}$

composed with $f_{x_{15}}$ composed with ... composed with f_{x_1}, where $x_1, \ldots, x_{16}$ are 16 independently chosen keys each of length 48 and f_{x_i} is a very simple function. We believe that MDES is at least as secure as DES when used in a block private key cryptosystem. On the other hand, it is easy to see that if MDES is used in a UNIX-like password scheme, then the resulting password scheme is totally insecure. To see this, suppose that we want to find a password which encrypts to a particular string y of length 64. Let y_1 be the first half of y and let y_2 be the second half of y. The password is chosen by first choosing $x_1, \ldots, x_{14}$ arbitrarily. Let the output of $f_{x_{14}}$ composed with ... composed with f_{x_1} on input 0^{64} be s, where s is a bit string of length 64. Let u be the output of $f_{x_{15}}(s)$ and let u_1 be the left half of u and u_2 be the right half of u. Let v be the output of $f_{x_{16}}(u)$ and let v_1 be the left half of v and v_2 be the right half of v. By the way MDES works, the left half of the final output at level 16 is simply the right half of the output at level 15, i.e., $v_1 = u_2$. Thus, our first task is, given an arbitrary s, fix x_{15} so that $y_1 = v_1 = u_2$. Our second task is, given an arbitrary u, fix x_{16} so that $y_2 = v_2$. Both of these tasks are of the same type, i.e., given an arbitrary bit string α of length 64 and an arbitrary bit string β of length 32, find an x of length 48 such that the right half of $f_x(\alpha)$ is equal to β. This turns out to be a straightforward task for the simple functions defined by MDES.

6. Additional Results and Observations

What if f is a pseudorandom function generator, but $l(n)$ is too large, so that our theorem does not apply? We can still obtain a secure password scheme as follows: let $m(n) = \lceil l(n)/n \rceil$ and, for password x, store in the password file $f_x^n(\alpha_1), f_x^n(\alpha_2), \ldots, f_x^n(\alpha_{m(n)})$ where $\alpha_1, \alpha_2, \ldots, \alpha_{m(n)}$ are $m(n)$ fixed distinct strings in $\{0, 1\}^n$. This can be proven secure using ideas similar to the ones used in the proof of the main theorem. In fact, this construction works as long as m is any function that satisfies, for some constant d and for all sufficiently large n, $l(n) \leq n \cdot m(n) + d \log n$.

Now let us assume that f and l satisfy the conditions of the main theorem. What about alternative password schemes? The following password scheme can also be shown to be secure. For password x, a random string α is chosen and the pair $(\alpha, f_x(\alpha))$ is stored in the password file. However, consider the following password scheme where $l(n) = n$. For password x, $f_x(x)$ is stored in the password file. It can be shown (by techniques similar to those used in Example 1 above) that, even if f is a pseudorandom function generator, this password scheme may be *very* insecure. The point of these examples is to emphasize further that the blind application of crytographic tools in situations where they are seemingly secure, but have not been proven secure, is dangerous.

Lastly we raise the well-known point that, contrary to our implicit assumptions, passwords are *not* chosen randomly in practice. We therefore have to model how users really choose passwords. Any version of our main theorem which we can prove under an alternative probability distribution assumption about the choice of passwords (alternative to the assumption that passwords are chosen uniformly at random) will involve f being pseudorandom when keys are chosen according to this alternative probability distribution.

Acknowledgments

We thank Paul Beame, Johan Hastad, and Gilles Brassard for some helpful discussions and simplifications to this work. We would also like to thank both referees for their very careful reading of the submitted paper and for making many very helpful clarifying and simplifying suggestions.

References

[BM] Blum, M., and Micali, S., How to generate cryptographically strong sequences of pseudo-random bits, *SIAM J. Comput.* Vol. 13, 1984, pp. 850–864.

[D] Denning, D., *Cryptography and Data Security*, Addison-Wesley, Reading, MA, 1983.

[GGM] Goldreich, O., Goldwasser, S., and Micali, S., How to construct random functions, *J. Assoc. Comput. Mach.*, Vol. 33, No. 4, October 1986, pp. 792–807.

[LR1] Luby, M., and Rackoff, C., Pseudo-random permutation generators and cryptographic composition, *Proceedings of the 18th ACM Annual Symposium on Theory of Computing*, May 28–30, 1986, pp. 356–363.

[LR2] Luby, M., and Rackoff, C., How to construct pseudo-random permutations from pseudo-random functions, *SIAM J. Comput.*, Vol. 17, 1988, pp. 373–386.

[LR3] Luby, M., and Rackoff, C., A study of password security, *Proceedings of Crypto '87*, Lecture Notes in Computer Science, Vol. 293, Springer-Verlag, Berlin 1988, pp. 392–397.

[Y] Yao, A. C., Theory and applications of trapdoor functions, *Proceedings of the 23rd Symposium on the Foundations of Computer Science*, 1982, pp. 80–91.