A Study of Password Security

Michael Luby †
Charles Rackoff ‡
University of Toronto

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. When a user logs in for the first time he creates a random password and types his user name together with the password into the system. The system creates an encryption of the password using the Data Encryption Standard (DES) and stores this (only the encryption, not the password) together with the user name in a password file. Thereafter, whenever the user logs in and types in 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 guarantees that the password file is write protected but not necessarily read protected. We do not formally justify this model of the system, but only remark that perhaps the 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. Informally, the password scheme would be secure 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 0^{64} (the bit string consisting of 64 zeroes) using key x. It is stated informally in [De] that this password scheme is secure if DES is secure when used in a private key cryptosystem. We formally investigate this question by the following approach.

† research partially supported by the Canadian Natural Sciences and Engineering Research Council operating grant A8092 and by a University of Toronto research grant

‡ research partially supported by the Canadian Natural Sciences and Engineering Research Council operating grant A3611

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 pseudo-random function generator (see [GGM1], [GGM2], [LR1], [LR2] for definitions, existence of, and applications of pseudo-random function generators and pseudo-random permutation generators).

Let n be the length of the encryption of a password and let $l(n)$ be the length of the password. Very informally stated, what we prove is that if a pseudo-random function generator with key length $l(n) \leq n + O(\log n)$ is used in a UNIX-like password scheme then the password scheme is secure. On the other hand, we describe a pseudo-random function generator with key length $l(n) \geq n + h(n)$ where $\log n = o(h(n))$, such that the UNIX-like password scheme it generates is not at all secure. The implication of this latter result is that if the password is too much longer than the encryption of the password, then even if we have a cryptosystem which is secure when used as a block private key crytosystem, the UNIX-like password scheme it generates may not be at all secure. As a more concrete example, we show that a modified version of DES, which most people believe is even more secure than DES when used in a block private key cryptosystem and also more pseudo-random, 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 sense and using it where it must be secure in a different sense and assuming without actually proving that it is secure in the different sense.

2. Definition of Secure Password Scheme

Let *poly* be the set of all polynomial functions.

Password Scheme Definition: g is a password scheme with password length function $l \in poly$, where $g = \{g^n : n \in N\}$, where 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.

Security Definition: A password scheme g is secure if there is no polynomial size family of circuits which can for infinitely many values of $n \in N$, receive an encryption of a randomly chosen password and output a password which has the same encryption with probability greater than $\frac{1}{n^c}$ for some constant $c > 0$. More precisely, g is secure if there is no polynomial size family of circuits (where the n^{th} circuit in the family is to be used to evaluate inputs of length n) A which has the following properties. On an input of length n, A produces an output of length $l(n)$. We say that A is **successful** on input y if A produces a string z such that $g^n(z) = y$. For infinitely many $n \in N$, A has the property that when x is a bit string of length $l(n)$ chosen randomly the probability that A is successful on input $g^n(x)$ is at least $\frac{1}{n^c}$.

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 pseudo random function generator.

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 alternative sense defined below, which reflects this informal notion of security.

Alternative Security Definition: Exactly the same as for the security definition except the properties of the circuit family A are modified as described here. Let $b \in poly$ and let A be a polynomial size family of circuits similar to A described above except that it has $b(n)$ inputs of length n and one output of length $l(n)$. A is successful on a particular input if it successfully produces a string z of length $l(n)$ such that $g^n(z)$ is equal to one of the $b(n)$ input strings.

UNIX-like Password Scheme: Let f be a function generator with key length $l \in poly$ (see [GGM1], [GGM2], [LR1] or [LR2] for a definition of a function generator and the definition of a pseudo-random function generator). For each $n \in N$, the encryption of a password x of length $l(n)$ is $f_x^n(0^n)$.

2. The Main Theorem

Theorem: If f is a pseudo-random function generator and $l(n) \leq n + d\log n$ for some constant d and for all sufficiently large $n \in N$, then the UNIX-like Password Scheme is secure.

Proof: We assume for contradiction that the password scheme is not secure. Thus, for some constant $c > 0$ there is a polynomial size family of circuits A which for some infinite $N_1 \subseteq N$ has the following characteristics. For each $n \in N$, let $\varepsilon(n)$ be the probability that A is successful when a password x of length $l(n)$ is chosen at random and the input to A is $f_x^n(0^n)$. For each $n \in N_1$, $\varepsilon(n) \geq \frac{1}{n^c}$. We show how to construct a polynomial size family of circuits C, one circuit for each $n \in N_1$, which demonstrates that f is not pseudo random.

For each $n \in N$, define $\varepsilon'(n)$ to be the probability that A is successful when a string y of length n is chosen at random and the input to A is y. For each $n \in N_1$, we consider two cases:

Case 1: $\varepsilon'(n) \le \frac{\varepsilon(n)}{2}$. In this case $\varepsilon(n) - \varepsilon'(n) \ge \frac{1}{2n^c}$. Let F^n be the set of all functions from n bits to n bits. We define the circuit with index n in the circuit family C in terms of A, which is to distinguish F^n from f^n as follows:

Circuit C: The input to C is a function f_1.
C computes $\alpha = f_1(0^n)$.
C computes $x = A(\alpha)$.
C computes $y = f_x^n(0^n)$.
If $y = \alpha$ then C outputs 1, otherwise C outputs 0.

If f_1 is randomly chosen from f^n, then the probability that C outputs 1 is $\varepsilon(n)$. If f_1 is randomly chosen from F^n, then the probability that C outputs 1 is $\varepsilon'(n)$. Thus, C distinguishes between F^n and f^n with probability at least $\varepsilon(n) - \varepsilon'(n) \ge \frac{1}{2n^c}$.

Case 2: $\varepsilon'(n) \ge \frac{\varepsilon(n)}{2}$. In this case $\varepsilon'(n) \ge \frac{1}{2n^c}$. We define the circuit with index n in the circuit family C in terms of A, which is to distinguish F^n from f^n as follows:

Circuit C: The input to C is a function f_1.
C computes $\alpha = f_1(0^n)$.
C computes $x = A(\alpha)$.
C computes $y = f_x^n(1^n)$.
C computes $\beta = f_1(1^n)$.
If $y = \beta$ then C outputs 1, otherwise C outputs 0.

If f_1 is randomly chosen from f^n, then by the claim below the probability that C outputs 1 is at least $\frac{1}{2n^{c+d}}$. If f_1 is randomly chosen from F^n, then the probability that C outputs 1 is $\frac{1}{2^n}$. Thus, C distinguishes between F^n and f^n with probability at least $\frac{1}{2n^{c+d}} - \frac{1}{2^n} \ge \frac{1}{4n^{c+d}}$ for sufficiently large n.

Claim: When x is a randomly chosen string of length $l(n)$ then the probability that $A(f_x^n(0^n)) = x$ is at least $\frac{1}{2n^{c+d}}$.

Proof of Claim: Let M be the set of strings x of length $l(n)$ such that A is successful on input $f_x^n(0^n)$. Let M' be the set of string y of length n such that A is successful on input y. Let S be the set of strings x of length $l(n)$ such that $A(f_x^n(0^n)) = x$. Since A is a one to one onto function from M' to S, $|S| = |M'|$. It is easy to see that $\frac{|M|}{2^{l(n)}} = \varepsilon(n)$ and $\frac{|M'|}{2^n} = \varepsilon'(n)$. Thus, since $l(n) \le n + d\log n$,

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

□

From case 1 and case 2 it is easy to see that for almost all $n \in N_1$, the circuit family C distinguishes f^n from F^n with probability at least $\frac{1}{4n^{c+d}}$. Thus, the function generator f is not pseudo-random. □

Comment: Levin [Le] proves a similar theorem to this, in a completely different context, in the case when $l(n) \le \frac{n}{2}$.

3. 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 pseudo-random 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 people believe to be even more secure than the Data Encryption Standard, when used in a UNIX-like password scheme, produces a totally insecure password scheme.

The first example is as follows. Let f be a pseudo-random function generator with key length $k(n)=n$. We define a function generator g in terms of f as follows. The key length function for g is $l(n)=n+\log^2 n$ ($\log^2 n$ was chosen to be such that for all $h \in poly$, for sufficiently large n, $\frac{1}{2^{\log^2 n}} \le \frac{1}{h(n)}$). 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 $a \ne 0^n$ of length n, $g_x^n(a)=f_{x_2}^n(a)$. $g_x^n(0^n)$ is defined to be $f_{x_2}^n(0^n)$ if $x_1 \ne 0^{\log n^2}$ and is defined to be x_2 if $x_1 = 0^{\log^2 n}$. It is not hard to prove that g is pseudo-random. 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.

The second very practical example is the following. 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 the Data Encryption Standard except that it uses 16 independent keys of length 48, and these 16 keys are used in the 16 levels of DES. Most people believe that MDES is at least as secure as DES when used in a block private key cryptosystem. Most people believe that MDES is as pseudo-random as DES. 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 has the same encryption as 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 arbitrarily choosing the first 14 keys of length 48. The 15^{th} key is computed such that the left hand side of the final encryption matches y_1. Because of the way each level of DES is computed, the 15th key can be easily computed for any set of values of the first 14 keys and for any value of y_1. Similarly, the 16th key is computed such that the right hand side of the final encryption matches y_2. For exactly the same reasons, the 16th key can be easily computed for any set of values of the first 15 keys and for any value of y_2.

4. Acknowledgements

We thank Paul Beame, Johan Hastad and Gilles Brassard for some helpful discussions and simplifications to this work.

5. References

[De] Denning, D., *Cryptography and Data Security*, January 1983, Addison-Wesley Publishing Company, Inc.

[GGM1] Goldreich, O., Goldwasser, S., Micali, S., *How to Construct Random Functions*, Proceedings of the 25th *Annual Symposium on Foundations of Computer Science, October 24-26, 1984*

[GGM2] Goldreich, O., Goldwasser, S., Micali, S., *How to Construct Random Functions*, J. for Association of Computing Machinery, Vol. 33, No. 4, October 1986, pp. 792-807 of Computer Science, October 24-26, 1984

[Le] Levin, L.A., *One-Way Functions and Pseudorandom Generators*, Proceedings of the 17th ACM Annual Symposium on Theory of Computing, May 6-8 1985, pp. 363-365.

[LR1] Luby, M., Rackoff, C., *Pseudo-random Permutation Generators and Cryptographic Composition*, Proceedings of the 18th *ACM Annual Symposium on Theory of Computing, May 28-30, 1986*

[LR2] Luby, M., Rackoff, C., *How to Construct Pseudo-random Permutations from Pseudo-random Bits*, to appear in special issue on Cryptography, SIAM J. on Computing