# Augmented Encrypted Key Exchange:
## a Password-Based Protocol Secure Against Dictionary Attacks and Password File Compromise

Steven M. Bellovin
smb@research.att.com

Michael Merritt
mischu@research.att.com

*AT&T Bell Laboratories*

## Abstract

The encrypted key exchange (EKE) protocol is augmented so that hosts do not store cleartext passwords. Consequently, adversaries who obtain the one-way encrypted password file may (i) successfully mimic (spoof) the host to the user, and (ii) mount dictionary attacks against the encrypted passwords, but cannot mimic the user to the host. Moreover, the important security properties of EKE are preserved—an active network attacker obtains insufficient information to mount dictionary attacks. Two ways to accomplish this are shown, one using digital signatures and one that relies on a family of commutative one-way functions.

## 1 Introduction

Bellovin and Merritt [1] presented a protocol that allowed two parties sharing a password to communicate without exposing that password. That protocol, encrypted key exchange, or EKE, required that both parties have cleartext versions of the shared password, a constraint that cannot (or ought not) always be met. In particular, consider the problem of a user logging in to a computer that does not rely on a secure key server for authentication. As shown in [13, 4], it is inadvisable for most hosts to store passwords in either clear form or in a reversibly-encrypted form. Rather, some one-way hash function $H(P)$ is stored for each user password $P$. Password validation is performed by calculating $H(P')$ on the typed password $P'$, and seeing if it matches the stored value.

We show how to extend the EKE protocol to handle this situation. The new protocol, augmented encrypted key exchange (A-EKE), works by choosing particular functions $H(P)$ for host storage of passwords. Both sides use $H(P)$, which should be secret, as the shared password in the EKE exchange. However, since we assume that under some comparatively-rare circumstances, $H(P)$ might be compromised, the user must send an additional message containing a different one-way function of the password; this value, together with $H(P)$ and the session key, is used by the host to validate the login sequence.

We start by reviewing EKE, in Section 2. Section 3 presents the modified protocol and two different ways to implement the it, one using public-key cryptography, and the other using commutative one-way hash functions. Section 4 analyzes the security of the new protocol.

### 1.1 Assumptions and Constraints

Most of the fundamental assumptions of EKE apply here as well. Specifically, we assume that the user's sole means of authentication—and sole long-term storage—is a simple password, rather than a bulky private key. Furthermore, we assume that the password must be protected from dictionary attacks; historically, such attacks are quite successful. See, for example, [13, 9, 10, 12], among others.

### 1.2 Summary of Notation

Our notation is shown in Table 1. To avoid confusion, we use the word *"symmetric"* to denote a conventional cryptosystem; it uses *secret* keys. A public-key, or *asymmetric*,

Table 1: Notation

| | |
|---|---|
| $A, B$ | System principals. ($Alice$ and $Bob$). |
| $P$ | The password: a shared secret, often used as a key. |
| $K$ | A random secret key (for symmetric cryptosystems). |
| $R_A, R_B$ | Random exponents. |
| $K[\mathit{info}]$ | Symmetric (secret-key) encryption of "*info*" with key $K$. |
| $K^{-1}[\mathit{info}]$ | Symmetric (secret-key) decryption of "*info*" with key $K$. |
| $S_k(M)$ | Digital signature of $M$ with (private) key $S_k$. |
| $V_k(X, M)$ | Verification of signature $X$ of message $M$ with (public) key $V_k$. |
| $challenge_A$ | A random challenge generated by $A$. |
| $challenge_B$ | A random challenge generated by $B$. |
| $F, G, H$ | One-way hash functions. |
| $H_0, H_K$ | Commutative, one-way hash functions. |
| $\alpha, \beta$ | Base and modulus for discrete exponentiation. |

cryptosystem has *public* encryption keys and *private* decryption keys.

## 2 A Review of EKE

Assume that two parties, $A$ and $B$ (*Alice* and *Bob*), wish to establish a secret, authenticated session key. Initially, the only secret they share is $P$, a password that may be subject to dictionary attacks.

1. $A$ picks a random number $R_A$ and calculates $P[\alpha^{R_A} \ (mod\ \beta)]$.

   $A$ sends
   $$A, P[\alpha^{R_A} \ (mod\ \beta)] \qquad \text{(EKE.1)}$$
   to $B$; note that her name is sent in the clear.

2. $B$ picks a random number $R_B$ and calculates $\alpha^{R_B} \ (mod\ \beta)$. $B$ also uses the shared password $P$ to decrypt $P[\alpha^{R_A} \ (mod\ \beta)]$, and calculates
   $$(\alpha^{R_A R_B}) \ (mod\ \beta).$$
   The session key $K$ is derived from this value, perhaps by selecting certain bits. Finally, a random challenge $challenge_B$ is generated.

   $B$ transmits
   $$P[\alpha^{R_B} \ (mod\ \beta)], K[challenge_B]. \qquad \text{(EKE.2)}$$

3. $A$ uses $P$ to decrypt $P[\alpha^{R_B} \ (mod\ \beta)]$. From this, $K$ is calculated; it in turn is used to decrypt $K[challenge_B]$. $A$ then generates her own random challenge $challenge_A$.

   $A$ sends
   $$K[challenge_A, challenge_B]. \qquad \text{(EKE.3)}$$

4. $B$ decrypts $K[challenge_A, challenge_B]$, and verifies that $challenge_B$ was echoed correctly.

   $B$ sends
   $$K[challenge_A]. \qquad \text{(EKE.4)}$$

5. $A$ decrypts to obtain $challenge_A$, and verifies that it matches the original.

Since the quantities encrypted with $P$ are random numbers, an attacker cannot validate any guesses as to the password. The fact that $K$ depends on inputs from both $A$ and $B$ defeats man-in-the-middle attacks. Furthermore, neither party can control the choice of $K$; to do so is equivalent to solving the discrete log problem.

There is another major variant of EKE, which uses asymmetric cryptosytems rather than exponential key exchange. In general, that version is not suitable for use with A-EKE; this is discussed further in Section 4.

## 3 EKE with Hashed Passwords

The modified protocol requires some new definitions. First, there is $H(P)$, the password-hashing function. Naturally, this must be a one-way function; there must be no feasible way to recover $P$ from $H(P)$. By our assumptions, hosts should try to keep $H(P)$ secret, but this cannot be guaranteed. The goal is to prevent an intruder who has captured $H(P)$ from succeeding in learning $P$ or in mimicking Alice. Moreover, attackers who have not obtained $H(P)$ must remain unable to mount dictionary attacks against $P$, as in the original $EKE$ protocol. (Note that any intruder with $H(P)$ can already mount a dictionary attack, so we need not guard further against that possibility.)

The second function, $F(P,K)$ is a one-way function that depends on both the password and the previously-negotiated session key. The user calculates this quantity, encrypts it with $K$, and sends

$$K[F(P,K)]$$

to the host.

Finally, define a predicate,

$$T(H(P),F(P,K),K);$$

this evaluates to **true** if and only if the genuine password $P$ was used to create both $H(P)$ and $F(P,K)$. (That is, $T(X,Y,Z)$ is true if and only if $X = H(P')$ and $Y = F(P',Z)$, for some $P'$.)

The basic idea is to first run EKE with $H(P)$ in place of the cleartext password, $P$. The result is a session key, $K$, that should be known only by $A$ and $B$, and can in practice only be known by someone who knows $H(P)$. Most attackers will not get this far, but one who has captured $H(P)$ will be able to impersonate either the host, the user, or both. Consequently, the user must supply $F(P,K)$ to persuade $B$ of her identity. Simply sending $P$, either in the clear or as $K[P]$, would be unsafe against an enemy who was mimicking the genuine $B$.

Below is the protocol in full—the first five steps are simply EKE with $H(P)$ used in place of $P$:

1. $A$ picks a random number $R_A$ and sends

   $$A, H(P)[\alpha^{R_A}\ (mod\ \beta)] \qquad \text{(A-EKE.1)}$$

   to $B$. Note that $A$ has $P$, and hence can easily calculate $H(P)$.

2. $B$ picks a random number $R_B$, and uses the shared, encrypted password $H(P)$ to decrypt $H(P)[\alpha^{R_A}\ (mod\ \beta)]$, calculates $(\alpha^{R_A R_B})\ (mod\ \beta)$, and derives $K$ from this value. Finally, $challenge_B$ is generated.

   $B$ transmits

   $$H(P)[\alpha^{R_B}\ (mod\ \beta)], K[challenge_B]. \qquad \text{(A-EKE.2)}$$

3. $A$ uses $H(P)$ to decrypt $H(P)[\alpha^{R_B}\ (mod\ \beta)]$, uses this to calculate $K$, which in turn is used to decrypt $K[challenge_B]$. $A$ then generates $challenge_A$.

   $A$ sends

   $$K[challenge_A, challenge_B]. \qquad \text{(A-EKE.3)}$$

4. $B$ decrypts $K[challenge_A, challenge_B]$, and verifies that $challenge_B$ was echoed correctly.

   $B$ sends

   $$K[challenge_A]. \qquad \text{(A-EKE.4)}$$

5. $A$ decrypts to obtain $challenge_A$, and verifies that it matches the original.

6. This ends the execution of EKE using $H(P)$ in place of the cleartext $P$. We are concerned that an adversary that obtained $H(P)$ still not be able to spoof the host as $A$. The protocol is extended by a single message:

   $A$ sends

   $$K[F(P,K)]. \qquad \text{(A-EKE.5)}$$

7. Upon receipt, $B$ decrypts to obtain $F(P,K)$, and concludes the protocol successfully only if the predicate $T(H(P),F(P,K),K)$ evaluates to true.

Note that an attacker who obtains $H(P)$ from the host could successfully mimic the host to $A$. But without knowledge of $P$, the attacker still cannot mimic $A$ to the host.

The scheme can be strengthened against dictionary attacks by letting Bob pick a per-user random value $s$, and store the pair $\langle H(s,P), s\rangle$ instead of simply $H(P)$. Then Bob would send $s$ (in the clear) to Alice as an added initial step of the EKE protocol, and $H(s,P)$ would be used in place of $H(P)$. This provides us with two of the three advantages of the "salt" used by Morris and Thompson [13]: it is impossible to tell if two hashed passwords use the same plaintext, and it is impossible to build a dictionary of pre-hashed password guesses. However, this change allows an attacker an additional degree of freedom over the protocol, which must be considered carefully in designing a secure implementation.

There may be several different secure choices for $H()$, $F()$, and $T()$ that will satisfy the protocol requirements. We present two here: digital signatures and commutative one-way functions. Other possibilities include functions based on Gifford's primitives [8].

## 3.1 Digital Signatures

Conceptually, the most straightforward way to augment EKE is to define $H(P)$ as the public key in a digital signature scheme. In turn, to compute $F(P,K)$, Alice signs $K$ with her private key. At user registration time, the host (Bob) keeps only the public key; at login time, the user (Alice) uses the public key for the EKE exchange, and the private key for the A-EKE extension.

In signature schemes such as ElGamal [7] or NIST's [14], calculating the public key $V_P$ can much more expensive than calculating the private key; the user shouldn't have to do it at login time. Any number (such as $P$, or some simple function of it) can be a private key, but the public key must be calculated. Thus, it may be more efficient for the host to store *both* a one-way hash $H(P)$ and a public key $V_P$ derived from $P$. Then an easier-to-compute $H(P)$ can be used for the EKE exchange, and $V_P$ can be used to verify the signature. Moreover, Alice's generation of the private key, $S_P$, from $P$ could proceed in parallel with the EKE exchange.

Using RSA[16] for the signature step is inadvisable. The public key includes a number of very special form, to wit the product of two large primes. To generate this, $P$ would have to act as the seed in a cryptographically-strong random-number generator (i.e., as described in [2]). Achieving consistent implementations could be tricky. Both parties must calculate exactly the same pseudo-random numbers and perform exactly the same probabilistic primality tests.

However it is achieved, as $F(P, K)$ the user simply transmits

$$K[S_P(K)],$$

where $K$ is the negotiated session key. Then the predicate $T(X, Y, Z)$ evaluated by the host is true only if $V_X(Y, Z^{-1}[Y])$ is true. (In a successful execution, this amounts to checking $V_P(K^{-1}[K[S_P(K)]], K)$.)

Since Alice does not unilaterally select $K$, this cannot be a replay; since Bob cannot control it either, there is no danger of a chosen ciphertext attack.

Below is a full version of this form of the protocol. The public key $V_P$ for the signature algorithm is held by the host, and is used as $H(P)$. The user's end calculates $V_P$ from the supplied password $P$. Note carefully the distinction between $V_P(X, M)$ and $V_P[X]$; the former is a signature operation, possibly via an *asymmetric* cryptosystem, while the latter uses the value of the key from a signature system as the key to a *symmetric* cryptosystem.

1. $A$ picks a random number $R_A$ and sends
$$A, V_P[\alpha^{R_A}\ (mod\ \beta)] \qquad \text{(SA-EKE.1)}$$
to $B$.

2. $B$ picks a random number $R_B$, and uses the shared, public signature key $V_P$ to decrypt (via a *symmetric* cryptosystem) $V_P[\alpha^{R_A}\ (mod\ \beta)]$, calculates $(\alpha^{R_A R_B})\ (mod\ \beta)$, and derives $K$ from this value. Finally, $challenge_B$ is generated.
$B$ transmits
$$V_P[\alpha^{R_B}\ (mod\ \beta)], K[challenge_B]. \qquad \text{(SA-EKE.2)}$$

3. $A$ uses $V_P$ to decrypt $V_P[\alpha^{R_B}\ (mod\ \beta)]$, uses this to calculate $K$, which in turn is used to decrypt $K[challenge_B]$. $A$ then generates $challenge_A$.
$A$ sends
$$K[challenge_A, challenge_B]. \qquad \text{(SA-EKE.3)}$$

4. $B$ decrypts $K[challenge_A, challenge_B]$, and verifies that $challenge_B$ was echoed correctly.
$B$ sends
$$K[challenge_A]. \qquad \text{(SA-EKE.4)}$$

5. $A$ decrypts to obtain $challenge_A$, and verifies that it matches the original.

6. $A$ sends
$$K[S_P(K)]. \qquad \text{(SA-EKE.5)}$$

7. Upon receipt, $B$ decrypts to obtain $S_P(K)$, and concludes the protocol successfully if and only if $V_P(K^{-1}[K[S_P(K)]], K)$ is true.

### 3.2 Commutative Hash Functions

Let $H(P)$ be defined as $H_0(P)$, a member of a family of commutative one-way hash functions, $\{H_0, H_1, ...\}$. After the EKE transfer using $H(P) = H_0(P)$, both $A$ and $B$ generate the hash function $H_K()$. Then, $A$ computes $F(H, K) = H_K(P)$ and sends it to $B$. Since $B$ knows $H_0(P)$, he can compute both $H_0(H_K(P))$ and $H_K(H_0(P))$, the first by hashing the received message $H_K(P)$ with $H_0()$ and the second by hashing the stored $H_0(P)$ with $H_K()$. If these two values are the same[1], $B$ can conclude that the other party knows *both* $H_0(P)$ and $P$.

The security of both this and the public-key variant depend critically on the information-hiding properties of the component transformations (the hash functions and cryptosystems). The next section discusses these requirements. At present, we do not know of any family of commutative one-way functions that satisfy the protocol requirements, while hiding sufficient information.

## 4 Security Analysis

Implicit in our descriptions of both variants of A-EKE is that the component transformations do not "leak" useful information to a potential adversary. That is, it should be computationally infeasable for an active or passive adversary to obtain useful information about $P$, or to successfully spoof either protocol participant, via replays or other strategies.

There are three, complementary approaches to formalizing this requirement. The first postulates that the algebraic identities *required* by the protocol are the *only* identities satisfied by the component cryptosystems and message transformations [5, 6, 3]. The second weakens these assumptions and attempts to find a minimal set of (often complexity-theoretic) assumptions that suffice to prove the protocol secure, and the third takes a much more pragmatic approach of examining the concrete implementations (e.g. DES or NIST) for known (or discovered) flaws.

We consider the first and third approaches in what follows, as the most pragmatically useful, and leave the second as an open problem. (The protocol and its security requirements

[1] That is, the predicate $T(X, Y, Z)$ is true if and only if $H_Z(X) = H_0(Y)$.

can be understood as abstractly specifying "the weakest assumptions which suffice to render the protocol secure".)

## 4.1 Requirements

The different protocol variants are designed to withstand two different kinds of attacks: first, against an attacker with no knowledge of $H(P)$, they must not provide enough information to mount a dictionary attack against $P$ or $H(P)$. Second, an attacker that knows $H(P)$ should neither be able to mimic the user to the host, nor be able to learn useful information about $P$ (without running a dictionary attack.)

## 4.2 A Ping-Pong Analysis

Following [5, 6, 3], Alice and Bob can be formalized as simple state machines, sending and receiving inputs from and to an adversary, and ringing alarms in any inputs fail to satisfy appropriate tests. In addition to obtaining new messages from Alice and Bob, the adversary has certain computational abilities. (For example, the function $H()$ is presumed known, so given a message $M$, the adversary can compute $H(M)$, $H(H(M))$, etc.) A security analysis explores all possible actions by the adversary, substituting arbitrary available messages in protocol executions with Alice and Bob. Despite these actions, the adversary should neither learn the password $P$, nor any verifiable function of $P$ that could be used to mount a dictionary attack. Moreover, if the adversary is assumed to know $H(P)$, he or she should still not be able to use attacks on the protocol to learn $P$. And of course, classic authentication properties must be preserved by the protocol: for example, if Alice terminates the protocol correctly (without ringing an alarm), with a particular value for the key $K$, then either she has been executing the protocol correctly with $B$, or with an adversary who has knowledge of $H(P)$. The analysis is outlined here–formal details are easily instantiated.

### 4.2.1 Adversaries Without $H(P)$

We begin with an assumption that Alice knows the password, $P$, Bob knows $H(P)$, and several functions and predicates are common knowledge, including $H()$, $F()$ and $T()$. Thus, we assume that an adversary can compute $H()$, $F()$ or $T()$ on any known arguments (tuples of the appropriate arity). We assume that these arguments are unrelated to $P$ or any legitimately negotiated session key.

Taking the initial EKE phase as a black box, we have a mechanism by which Alice and Bob can negotiate a random session key, $K$, each assured that his or her correspondent knows $H(P)$, and without divulging any useful information about $H(P)$ or $K$. Now Alice sends $K[F(P,K)]$ to Bob, who decrypts and evaluates $T(H(P), F(P,K), K)$. Clearly, this simple addition to EKE affords little new opportunity for an attacker. An active attack could attempt to substitute another message, but none available will pass Bob's test. Since the session key is different with each protocol execution, the adversary learns new messages, $K_1[F(P,K_1)], K_2[F(P,K_2)], \ldots$, but none are useful in attacking later instantiations. Moreover, these messages are protected by the (cryptographically strong) encryption by $K_i$. Hence, the adversary has no simple test to determine whether a prospective key $P'$ has been used in these protocols: to determine whether $P' = P$ in $K[F(P,K)]$ is to answer the query "does there exist a $K'$ such that $K'[F(P',K')] = K[F(P,K)]$?"

### 4.2.2 Adversaries with $H(P)$

Even if a host is subverted and an adversary obtains $H(P)$, we are interested in protecting $P$, and preventing the adversary from mimicking the user. Hence, we augment the initial state in the previous analysis, by assuming the adversary knows $H(P)$. The first phase, running EKE, depends only on $H(P)$, not otherwise on $P$, so the adversary learns nothing new there. Now Alice may or may not be communicating with Bob, but she still sends only the new message $K[F(P,K)]$. In this case, the adversary may know $K$, and so obtain $F(P,K)$. But there is no mechanism for the adversary to obtain $P$ from $F(P,K)$. Moreover, since Bob ensures the session key $K$ is unique to each protocol execution, even given $F(P,K_1), F(P,K_2), \ldots$ an adversary has insufficient information to compute $K_i[F(P,K_i)]$ for a new $K_i$.

## 4.3 Implementations

An implementation of a cryptographic protocol approximates the information-hiding properties of the abstract protocol with actual cryptosystems and messages spaces. That is, the implementation is a *model* of the abstract protocol. In particular, the implementation of the cryptographic operations must satisfy the algebraic properties of the abstract operations. Unfortunately, they will almost certainly satisfy additional algebraic properties, which may provide an adversary opportunities for defeating the protocol. No known concrete complexity theory suffices to demonstrate that such an implementation is secure against an appropriately defined adversary. However, the abstract analysis provides guidance as to specific information that must remain hidden, or capabilities that must be denied the adversary.

### 4.3.1 A-EKE Using Digital Signatures

For example, an implementation of A-EKE that uses digital signatures implements the function $F(X,Y)$ as $S_X(Y)$, the digital signature of $Y$ using a private key derived from $X$. Since $Y$ (or information about $Y$) can be obtained from $S_X(Y)$ using the corresponding public key, this implementation introduces new inference capabilities not considered in the previous analyses. In particular, an adversary who has compromised the host will have obtained the public key derived from $P$. Hence, when the user sends

$K[F(P,K)] = K[S_P(K)]$, the adversary can decrypt with $K$ to obtain $S_P(K)$ and check the signature using the public key.

Different signature schemes leak differing information about the signed message. For example, if the signature scheme is RSA, $Y$ can be obtained from $S_X(Y)$ using the public key associated with $X$. Hence, the adversary can in this case obtain $K$ from $S_P(K)$. But this is already known. So in this case, these new inferences provide no new information. (Of course, this considers only a single possible attack. An exhaustive analysis of an adversary's potential actions is required, to ensure a security flaw is not introduced by the implementation.)

#### 4.3.2 A-EKE Using Commutative Hash Functions

A less satisfactory result occurs if we choose the RSA public-key scheme to provide a family of commutative hash functions [16]. While the consequent transformations satisfy the identities needed for the protocol to make syntactic sense, such use of RSA is not secure for our purposes.

This implementation uses just the encryption part of RSA. Specifically, let $n = pq$ for some pair of large primes $p$ and $q$. These primes are then discarded. Define

$$H_0(x) = x^{h_0} \ (mod\ n)$$

for some constant $h_0$, and

$$H_K(x) = x^{\kappa} \ (mod\ n),$$

where $\kappa$ is $K$ or a simple function of $K$.

Thus, predicate $T$ becomes:

$$H_0(H_K(P)) = (P^{\kappa})^{h_0} \ (mod\ n)$$

$$\stackrel{?}{=} (P^{h_0})^{\kappa} \ (mod\ n) = H_K(H_0(P)),$$

which reduces to

$$P^{\kappa h_0} \ (mod\ n) \stackrel{?}{=} P^{h_0 \kappa} \ (mod\ n).$$

Assume, then, that an adversary has a copy of $H_0(P)$, and captures $H_K(P)$ for some $K$. These correspond to $P^{H_0} \ (mod\ n)$ and $P^{\kappa} \ (mod\ n)$. But that is two RSA encryptions of the same message, using a common modulus and different exponents. If $\kappa$ and $h_0$ are relatively prime—and the probability [11, Section 4.5.2] that they are is $6/\pi^2$, or about .61—Simmons' attack [17] will immediately yield $P$. If $\kappa$ and $h_0$ are not relatively prime, the attacker will be able to solve for $P^g$, where $g = \gcd(h_0, \kappa)$. This value can then be used in calculating $H_{K'}(X)$ if $g$ divides $\kappa'$. (This latter property dooms any attempt to substitute Rabin's variant on RSA [15], whose solution is equivalent in difficulty to factoring $n$: the attacker can retrieve only $P^2$, but that value is itself usable in calculating any even power of $P$, as is required by Rabin's function.)

### 4.4 Public-Key EKE

An alternative implementation of the original EKE protocol [1] relies on a public-key cryptosystem, instead of exponential key exchange. It has the drawback that the session key $K$ is chosen by one of the parties, as opposed to the exponenetial key exchange variant, in which $K$ is collaboratively generated.

But the A-EKE protocol fails in the presence of an active attacker $Z$ if either $A$ or $B$ can control the choice of $K$. Suppose that $A$ can choose $K$. Then the attacker can impose that value of $K$ on the real $B$. And that, in turn, permits use of the legitimately-generated $K[F(P,K)]$ to authenticate $Z$'s connection to $B$.

Similarly, suppose that the host chooses $K$. In that case, $Z$ will impose that value on $A$, again obtaining $K[F(P,K)]$ to pass on to $B$.

## 5 Conclusions

The original EKE protocol protected passwords being sent over the network, but required a trusted key distribution center. We have now extended EKE so that it may be used when talking to a single host. Local users' hashed passwords are protected by the operating system's file protection mechanisms; remote users' passwords are protected by the new protocol. No third parties are necessary.

It does not appear to be possible to protect passwords against an intruder who has captured the host's copy of the authentication data. Fundamentally, a password-guessing attack is equivalent to the attacker playing both roles, that of the user supplying a password, and that of the host verifying it. Thus, if the host has enough information to validate the legitimate user, the intruder would be able to validate a guess.

## References


[1] BELLOVIN, S. M., AND MERRITT, M. Encrypted key exchange: Password-based protocols secure against dictionary attacks. In *Proc. IEEE Computer Society Symposium on Research in Security and Privacy* (Oakland, May 1992), pp. 72–84.

[2] BLUM, M., AND MICALI, S. How to generate cryptographically strong sequences of pseudo-random bits. *SIAM J. Comput. 13*, 4 (November 1984), 850–864.

[3] DEMILLO, R., LYNCH, N., AND MERRITT, M. Cryptographic protocols. In *Proc. 14th ACM Symp. on the Theory of Computing* (May 1982), pp. 383–400.

[4] DIFFIE, W., AND HELLMAN, M. E. New directions in cryptography. *IEEE Transactions on Information Theory IT-11* (November 1976), 644–654.

[5] DOLEV, D., EVEN, S., AND KARP, R. On the security of ping-pong protocols. *Information and Control 55* (1982), 57–68.

[6] DOLEV, D., AND YAO, A. On the security of public key protocols. *IEEE Transactions on Information Theory IT-29*, 2 (March 1983), 198–208.

[7] ELGAMAL, T. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE Transactions on Information Theory IT-31* (July 1985), 469–472.

[8] GIFFORD, D. K. Cryptographic sealing for information secrecy and authentication. *Communications of the ACM 25*, 4 (1982), 274–286.

[9] GRAMPP, F. T., AND MORRIS, R. H. Unix operating system security. *AT&T Bell Laboratories Technical Journal 63*, 8, Part 2 (October 1984), 1649–1672.

[10] KLEIN, D. V. "Foiling the cracker": A survey of, and improvements to, password security. In *Proceedings of the USENIX UNIX Security Workshop* (Portland, August 1990), pp. 5–14.

[11] KNUTH, D. E. *Seminumerical Algorithms*, vol. 2 of *The Art of Computer Programming*. Addison-Wesley, 1969.

[12] LEONG, P., AND THAM, C. Unix password encryption considered insecure. In *Proc. Winter USENIX Conference* (Dallas, 1991).

[13] MORRIS, R. H., AND THOMPSON., K. Unix password security. *Communications of the ACM 22*, 11 (November 1979), 594.

[14] A proposed Federal Information Processing Standard for digital signature standard (DSS). Docket No. 910807-1207, RIN 0693-AA86.

[15] RABIN, M. Digitalized signatures and public-key functions as intractable as factorization. Tech. Rep. MIT/LCS/TR-212, Massachusetts Institute of Technology Laboratory for Computer Science, Cambridge, MA, January 1979.

[16] RIVEST, R. L., SHAMIR, A., AND ADLEMAN, L. A method of obtaining digital signatures and public-key cryptosystems. *Communications of the ACM 21*, 2 (February 1978), 120–126.

[17] SIMMONS, G. J. A "weak" privacy protocol using the RSA crypto algorithm. *Cryptologia 7*, 2 (1983), 180–182.