Encrypted Key Exchange: Password-Based Protocols Secure Against Dictionary Attacks

Steven M. Bellovin

AT&T Bell Laboratories
Murray Hill, NJ 07974
smb@ulysses.att.com

Michael Merritt

AT&T Bell Laboratories
Murray Hill, NJ 07974
mischu@research.att.com

Abstract

Classical cryptographic protocols based on user-chosen keys allow an attacker to mount password-guessing *attacks. We introduce a novel combination of asymmetric (public-key) and symmetric (secret-key) cryptography that allow two parties sharing a common password to exchange confidential and authenticated information over an insecure network. These protocols are secure against active attacks, and have the property that the password is protected against off-line "dictionary" attacks. There are a number of other useful applications as well, including secure public telephones.*

1 Introduction

People pick bad passwords, and either forget, write down, or resent good ones. We present a protocol that affords a reasonable level of security, even if resources are protected by bad passwords. Using a novel combination of asymmetric (public-key) and symmetric (secret-key) cryptography — a secret key is used to encrypt a randomly-generated public key — two parties sharing a secret such as a *password* use it to exchange authenticated and secret information, such as a session key or a "ticket" for other services, *a là* Kerberos [1]. This protocol, known as *encrypted key exchange*, or *EKE*, protects the password from off-line "dictionary" attacks.

EKE can be used with a variety of asymmetric cryptosystems and public key distribution systems, subject to a few constraints detailed below. It works especially well with exponential key exchange [2]. Section 2 describes the asymmetric cryptosystem variant and implementations using RSA[3] and ElGamal[4]. Each of those two systems presents unique problems. Section 3 generalizes EKE, and shows how most public key distribution systems can be used. Section 4 considers general issues related to the choice and use of symmetric and asymmetric cryptosystems in EKE. Finally, in Section 5, we describe applications for EKE, and discuss related work in Section 6.

Proceedings of the IEEE Symposium on Research in Security and Privacy, Oakland, May 1992.

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

1.2 Classical key negotiation

Suppose A (Alice) and B (Bob) share a secret, the password P. In order to establish a secure session key, A could generate a random key R, encrypt it with P as key and send the result, $P(R)$, to B. (This is essentially the mechanism used to obtain the initial ticket in the Kerberos authentication system [1].) Now A and B share R and can use it as a session key; perhaps B replies to A with

$$R(\texttt{Terminal type:})$$

But an eavesdropper could record these messages, and run a dictionary attack against P by first decrypting $P(R)$ with candidate password P', and then using the resultant candidate session key

$$R' = P'^{-1}(P(R))$$

Table 1: Notation

A, B	System principals. (*Alice* and *Bob*).
P	The password: a shared secret, often used as a key.
R, S	Random secret keys (for symmetric cryptosystems).
$R(info)$	Symmetric (secret-key) encryption of "$info$" with key R.
$R^{-1}(info)$	Symmetric (secret-key) decryption of "$info$" with key R.
$E_k(X)$	Asymmetric (public-key) encryption of X with (public) key E_k.
$D_k(X)$	Asymmetric (public-key) decryption of X with (private) key D_k.
$challenge_A$	A random challenge generated by A.
$challenge_B$	A random challenge generated by B.
p, q	Prime numbers.

to decrypt R(`Terminal type :`), examining the result for expected redundancy.

The simple protocol above has other flaws, particularly against replay attacks, but illustrates a weakness common to all classical two-party key exchange protocols: the enduring cryptographic secrets are susceptible to off-line, brute-force attacks. This may be fine when these secrets are long random strings, but poses considerable difficulty when the secrets are passwords chosen by naive users [5, 6, 7, 8].

2 EKE using public keys

Consider instead the following simple message exchange:

1. A generates a random public key/private key pair, E_A and D_A, and encrypts the public key in a symmetric cryptosystem with password P, yielding $P(E_A)$.

   A sends
   $$P(E_A) \tag{EKE.1}$$
   to B. (We will defer until later a discussion of how E_A and D_A are generated, and exactly what role P plays.)

2. Sharing the password P, B decrypts to obtain $P^{-1}(P(E_A)) = E_A$, generates a random secret key R, and encrypts it in the asymmetric cryptosystem with key E_A to produce $E_A(R)$. This value is further encrypted with P.

   B sends
   $$P(E_A(R)) \tag{EKE.2}$$
   to A.

3. A, knowing P and D_A, uses them to calculate $D_A(P^{-1}(P(E_A(R)))) = R$.

After this exchange, A and B both know E_A and R. The latter can be used to protect the session: B could send R(`Terminal type :`) to A. Consider, however, the position of an eavesdropper in this context. Knowing $P(E_A)$, $P(E_A(R))$, and R(`Terminal type :`), a candidate password P' can be used to decrypt $P(E_A)$ to produce a candidate public key $E_{A'} = P'^{-1}(P(E_A))$. But determining whether $E_{A'}$ is the public key used in the exchange amounts to determining whether *there exists* a secret key R' such that $E_{A'}(R') = E_A(R)$ and $R'^{-1}(R$(`Terminal type :`)$)$ makes sense. This quantification is the key property of the exchange: a candidate password P' cannot be rejected without doing a brute-force attack on R.[1] Since E_A and R are randomly generated over (presumably) large key spaces, such attacks are expensive, even if the space of passwords is small. So far as naive (non-cryptanalytic) off-line attacks are concerned, the relatively small space from which P is chosen has been effectively *multiplied* by the size of the keyspace from which R is obtained.

Another way to look at it is to examine the results of a trial decryption $E_{A'} = P'^{-1}(P(E_A))$. Is this a comprehensible quantity? If E_A is indeed random — a question to which we shall return later — there is no way to tell if P' is correct without cracking E_A. And, since E_A is chosen from a much larger key space than is P, cracking it is much more difficult.

[1] This discussion presumes the eavesdropper uses only non-cryptanalytic attacks.

2.1 A complete protocol

Real protocols are not as simple as the basic concepts outlined above. For example, an important concern is the possibility of replay attacks. That is, an attacker with control of the communications channel may insert old, stale messages. Protocols must incorporate safeguards, typically in the form of random challenges. Let us consider a full-blown version.

1. A generates a random public key E_A and encrypts it in a symmetric cryptosystem with key P to produce $P(E_A)$.

   A sends
   $$A, P(E_A) \tag{RPK.1}$$
   to B. This message includes her name in the clear.

2. Sharing the password P, B decrypts to obtain E_A, generates a random secret key R, and encrypts it in both the asymmetric cryptosystem with key E_A and in the password key to produce $P(E_A(R))$.

   B sends
   $$P(E_A(R)) \tag{RPK.2}$$
   to A.

3. A decrypts the message to obtain R, generates a unique challenge $challenge_A$ and encrypts it with R to produce $R(challenge_A)$.

   A sends
   $$R(challenge_A) \tag{RPK.3}$$
   to B.

4. B decrypts the message to obtain $challenge_A$, generates a unique challenge $challenge_B$, and encrypts the two challenges with the secret key R to obtain $R(challenge_A, challenge_B)$.

   B sends
   $$R(challenge_A, challenge_B) \tag{RPK.4}$$
   to A.

5. A decrypts to obtain $challenge_A$ and $challenge_B$, and compares the former against her earlier challenge. If it matches, she encrypts $challenge_B$ with R to obtain $R(challenge_B)$.

   A sends
   $$R(challenge_B) \tag{RPK.5}$$
   to B.

6. If the challenge-response protocol in steps RPK.1-RPK.5 is successful, login is successful and the parties proceed with the login session, using the symmetric cryptosystem and session key R for protection.

The challenge-response portion of the protocol, in steps 3–5, is a standard technique for validating cryptographic keys. (If a party sends challenge c encrypted by R, where c was never used before, and receives another encrypted message containing c in reply, it follows that the message originator has the ability to encrypt messages with R.) This portion of the protocol could be replaced by other mechanisms for validating R. For example, the time could be exchanged encrypted by R, under the security-critical assumption that clocks are monotonic and synchronized, as in Kerberos [1].

2.2 When to encrypt with the password

In the protocol presented above, the password was used for encryption twice, in messages RPK.1 and RPK.2. Often, one of those two encryptions may be omitted. Which one can be skipped will vary, depending on the particular asymmetric cryptosystem chosen.

The most obvious constraint is that the message to be encrypted by the password must be indistinguishable from a random number. If, for example, some cryptosystem required the use of prime numbers as public keys, it would not be possible to encrypt message RPK.1: an attacker would find it trivial to validate a guess at P by testing the resulting decryption for primality. Similarly, if an encrypted message always had some particular characteristics, message RPK.2 could not be the one encrypted. We will see this point illustrated with RSA. Other considerations may apply as well; an example is presented in Section 2.4.

The choice of which message to encrypt also has some subtle implications for the detailed protocol design. Specifically, the party that transmits in the clear cannot be allowed to generate the first challenge. Otherwise, an attacker can receive a known quantity — the challenge — encrypted with a value derivable solely from the user's password and information known to the attacker. Put another way, each party must be forced to demonstrate knowledge of P, either by encrypting a message to be read by the other side, or by responding to a challenge.

2.3 Implementing EKE using RSA

Actually implementing EKE can be somewhat trickier than it appears at first glance. We will use RSA[3] to illustrate the difficulties. Elaboration of some of the subtler points, though, is deferred until Section 2.5.

The public key E_A for the RSA cryptosystem consists of a pair of large natural numbers (e, n), where n is the product of two large primes p and q, and e is relatively prime to

$$\varphi(n) = \varphi(p)\varphi(q) = (p-1)(q-1).$$

The private decryption key d is calculated such that

$$ed \equiv 1 \ (mod\ (p-1)(q-1)). \tag{1}$$

A message m is encrypted by calculating

$$c \equiv m^e \ (mod\ n);$$

the ciphertext c is decrypted by

$$m \equiv c^d \ (mod\ n).$$

It is not clear how to encode efficiently a pair $<e, n>$ so that it is indistinguishable from a random string; an intruder could easily verify that most possible values of n' have small prime factors, and hence were not correct. Without such an encoding, we must encrypt only e. It is encoded by beginning with the binary encoding of e, and adding 1 with probability 1/2; the addition is done because all possible values of e are odd. Additionally, some mechanism must be provided to bring the length of this encoding to a block boundary for the symmetric cipher.

Can such a random odd number less than a known n be distinguished from a valid public key e? Assume that p and q are chosen to be of the form $2p' + 1$ and $2q' + 1$, where p' and q' are primes, a choice that is recommended for other reasons [9]. Then an overwhelming majority of the odd integers $(mod\ n)$ will be relatively prime to $(p-1)(q-1) = 4p'q'$, and hence will be valid candidate public keys e. Consequently, a dictionary attack on $P(e)$ will provide extremely little information about P. [2]

The fact that n is sent in the clear introduces some complexity to the analysis of the protocol. In particular, an adversary could substitute another number, n', for n in the first message, so that B receives $<P(e), n'>$. The resulting message from B will be of the form

$$(R, challenge_B)^e \ (mod\ n').$$

Now, from a candidate password P' the adversary can compute

$$e' = P'^{-1}(P(e)).$$

Assuming the adversary knows the factorization of n', the corresponding private key d' is easily computed and can be used to decrypt

$$(R, challenge_B)^e \ (mod\ n'),$$

obtaining

$$(R, challenge_B)^{ed'} \ (mod\ n').$$

If $e \neq e'$, this is a random number, but so is $(R, challenge_B)$. So a dictionary attack is of no help at this point, and the adversary must still deliver a message of the form

$$R(challenge_A, challenge_B),$$

but knows neither $challenge_B$ nor R. Unable to do so, the attack stops at this point and (time-out) alarms will ring at both A and B.

One more aspect of sending n in the clear is worth noting: it exposes the user to the risk of cryptanalysis. More precisely, if n is available to the attacker, it could be factored; that in turn would disclose R and expose P to attack. Without knowing n, an enemy cryptanalyst would be reduced to solving a system where the only plaintext was random. That task is essentially impossible.

Given the difficulty of encoding and encrypting the public key, it is tempting to suggest that it be sent in the clear, and that only the second message (RPK.2) be protected by P. However, that variant is susceptible to attack with RSA.

Let the enemy impersonate A. That person would then select p and q, and hence e and n. If e is chosen so that it does not satisfy equation (1), the space of encryptions collapses. That is, the possible values of

$$E_A(R) = R^e \ (mod\ n)$$

(the e-residues $(mod\ n)$) are a fraction of the interval $[0, n-1]$. An attack on EKE can be launched if the enemy can determine if a trial decryption

$$P'^{-1}(P(E_A(R)))$$

produces an e-residue. Such determinations may be feasible.

[2] In [3], the authors suggest satisfying equation (1) by choosing e to be a prime greater than $\max(p, q)$. Clearly, we cannot follow that advice here.

Defending against this attack would require that B be able to detect fraudulent values of e. However, he does not know the factorization of n, and hence does not know $\varphi(n)$; without such knowledge, it does not appear to be practical to validate e directly. One approach (suggested to us by Joan Feigenbaum) is to have B verify e interactively, by asking A to decrypt a number of random messages encrypted by e. That is, B generates a random r, sends r^e to A, and expects r as the reply. Since r^e is only invertible when e is relatively prime to $\varphi(n)$, correct replies to a number of such random challenges shows that e has the proper form. This variant is expensive in messages, encryptions and decryptions, and of course, must be shown to be immune to attack by B.[3]

2.4 Using the ElGamal asymmetric cryptosystem

The ElGamal cryptosystem[4] can also be used with EKE. Encryption with ElGamal has some interesting properties; these make its mode of employment rather different. In particular, under certain circumstances we must encrypt the second message, rather than the first.

ElGamal's algorithm is derived from the Diffie-Hellman exponential key exchange protocol[2]; accordingly, we will review the latter first. Briefly, A and B each pick random exponents R_A and R_B. Assuming they agree on a common base α and modulus β, A computes $Y_A \equiv (\alpha^{R_A}\ (mod\ \beta))$ and B computes $Y_B \equiv (\alpha^{R_B}\ (mod\ \beta))$. Both of these quantities are transmitted in the clear. A, knowing R_A and $\alpha^{R_B}\ (mod\ \beta)$, can compute

$$(\alpha^{R_B})^{R_A}\ (mod\ \beta) \equiv \alpha^{R_B R_A}\ (mod\ \beta)$$

Similarly, B can compute

$$(\alpha^{R_A})^{R_B}\ (mod\ \beta) \equiv \alpha^{R_A R_B}\ (mod\ \beta)$$

This quantity is used as the key. An intruder, knowing only $\alpha^{R_A}\ (mod\ \beta)$ and $\alpha^{R_B}\ (mod\ \beta)$, cannot perform the same calculation; no better solution is known than computing discrete logarithms in the field $GF(\beta)$, a problem that is believed to be hard for suitable values of β.

[3]Note that B — or an intruder —could as easily send any message m in place of r^e, obtaining m^d in reply. This is the signature of m by A, using the public/private key pair (e, d). While apparently not a problem with this variant of EKE using RSA, this is an example of how a slight change in a cryptographic protocol may have profound and unforeseen implications for the security of that protocol.

To convert this into an asymmetric encryption system, let the simple exponential

$$Y_X \equiv \alpha^{R_X}\ (mod\ \beta)$$

be the public key for X. To send a message m to B, A picks a random number k uniformly distributed between 0 and $\beta - 1$. Then she computes

$$c_1 \equiv \alpha^k\ (mod\ \beta)$$

and

$$\begin{aligned} c_2 &\equiv m(Y_B)^k\ (mod\ \beta) \\ &\equiv m(\alpha^{R_B})^k\ (mod\ \beta) \\ &\equiv m\alpha^{R_B k}\ (mod\ \beta). \end{aligned}$$

The encrypted message consists of the pair $<c_1, c_2>$.

Bob, knowing R_B, can decrypt the message by first calculating

$$\begin{aligned} K &\equiv {c_1}^{R_B}\ (mod\ \beta) \\ &\equiv \alpha^{R_B k}\ (mod\ \beta). \end{aligned}$$

He can then divide c_2 by K, yielding m.

Assuming that proper values are chosen for α and β (see Section 3.2), the important quantities in this cryptosystem fit nicely into the EKE scheme. The generated public key α^{R_A} is uniformly distributed in the interval $[0, \beta - 1]$; thus, no information is leaked when it is encrypted. The components of the encrypted message c_1 and c_2 are similarly distributed. Thus, message RPK.1 becomes

$$P(\alpha^{R_A}\ (mod\ \beta)),$$

while message RPK.2 becomes

$$P(\alpha^k\ (mod\ \beta), R\alpha^{R_A k}\ (mod\ \beta)).$$

At first glance, it appears that either encryption with P may be omitted. Depending on the exact format of the challenge/response messages, though, an attack may be possible if message RPK.2 is sent in the clear. Consider the following scenario, where a type code is used for the challenge/response messages as per Section 4.1. Alice sends Bob an encrypted public key:

$$P(\alpha^{R_A}\ (mod\ \beta)). \qquad \text{(XEG.1)}$$

The enemy intercepts this message. Without knowing P, it is not possible to decrypt the first message, so it is not possible to compute $(\alpha^{R_A})^k$. Accordingly, a random quantity X is substituted. It is possible to assign

$$c_1 \equiv \alpha^k\ (mod\ \beta).$$

Message RPK.2 thus becomes

$$\alpha^k \ (mod\ \beta), RX\ (mod\ \beta). \tag{XEG.2}$$

Alice, unaware of the imposture, computes

$$K \equiv \alpha^{R_A k}\ (mod\ \beta)$$

and hence

$$R' \equiv \frac{RX}{\alpha^{R_A k}}\ (mod\ \beta).$$

This value R' is used to encrypt the first challenge message:

$$R'(challenge_A). \tag{XEG.3}$$

Now, the attacker cannot calculate R' directly. But any guess at P yields a candidate value for α^{R_A}, and hence a candidate R'. If message XEG.3 contains any redundancy — i.e., if the challenge is typed, or if there is a checksum — a trial decryption using R' can be validated. And that in turn permits the enemy to validate guesses at P.

The attack we have just given does not succeed against RSA; it is instructive to analyze what the difference is. Intuitively, the problem with ElGamal is that the sender of a message has enough information to decrypt it again without knowing the recipient's private key. The random variable k is an additional secret; one who knows it, along with the recipient's public key, can decrypt the message.

More formally, let $\mathcal{K}$ be the space of all encryption keys, $\mathcal{K}^{-1}$ the space of decryption keys, $\mathcal{M}$ the space of plaintext messages, and $\mathcal{C}$ the space of ciphertext messages. For RSA, there exist two functions, E and D:

$$\begin{aligned} E: &\quad \mathcal{K} \times \mathcal{M} \rightarrow \mathcal{C} \\ D: &\quad \mathcal{K}^{-1} \times \mathcal{C} \rightarrow \mathcal{M} \end{aligned}$$

such that D is the inverse of E.

With ElGamal, there is an additional parameter, $k \in \mathcal{S}$, the "secret space", and an additional decryption function D':

$$D' : (\mathcal{K} \times \mathcal{S}) \times \mathcal{C} \rightarrow \mathcal{M}.$$

It is the existence of D', a second inverse function computable by the attacker, that forces us to encrypt at least message RPK.2. We call a cryptosystem with such a second inverse a *disclosing encryption system.*

2.5 Security considerations

2.5.1 Partition attacks

We have stated that the encryptions using P must leak no information. This is often quite difficult, simply because of the numerical properties of the cryptosystems used. For example, we noted that public keys in RSA are always odd; if no special precautions are taken, an attacker could rule out half of the candidate values P' if $P'^{-1}(P(e))$ were an even number. At first blush, this is an unimportant reduction in the key space; in fact, if left uncorrected, it can be a fatal flaw.

Recall that each session will use a different public key, independent of all others previously used. Thus, trial decryptions resulting in illegal values of e' will exclude different values of P' each time. Put another way, each session will partition the remaining candidate key space into two approximately-equal halves. The decrease in the keyspace is logarithmic; comparatively few intercepted conversations will suffice to reject all invalid guesses at P. We call this attack a *partition attack.*

For some cryptosystems, one may choose to accept a minimal partition. Consider a situation where one must encrypt, with P, integers modulo some known prime p. Clearly, if n bits are needed to encode p, trial decryptions yielding values in the range $[p, 2^n - 1]$ can be used to partition the password space. However, if p is very close to 2^n, perhaps even $2^n - 1$, very few candidates are excluded. Conversely, values of p near 2^{n-1} are quite bad. For any value of p, it is obviously possible to calculate how many interceptions are necessary to analyze any given size password space.

Another area of possible exposure comes from trying to encrypt a given number with a cryptosystem that demands a larger blocksize. The straight-forward way to do this — inserting high-order zero bits — poses an obvious risk. Instead, those bits should be filled with random data.

Often, we can solve both problems in one operation. Again, let assume that one is encrypting integers modulo p. Further assume that the desired input encryption block size is m bits where $2^m > p$. Let

$$x = \left\lfloor \frac{2^m}{p} \right\rfloor.$$

The value x is the number of times our legal interval fits into the encryption block size. We can thus choose a random value $j \in [0, x - 1]$ and add jp to the input value using *non-modulo* arithmetic. (If the input value is less than $2^m - xp$, use the interval $[0, x]$ instead.)

The recipient, knowing the modulus, can easily reduce the decrypted value to the proper range.

2.5.2 Tacit assumptions

The security of EKE rests on several assumptions. The most obvious is that the symmetric and asymmetric cryptosystems must not leak any useful information.

This somewhat vague condition may be understood more fully in the context of the particular protocol. Clearly, encryption of a random secret key R by random public key E_A must leak no useful information about either E_A or R.

But consider an attack on A in which message X is sent to A in step 2, where X may or may not be a message of the form $E_A(R, challenge_B)$ obtained from B. Let $D_1(X)$ denote the first part of the decryption of X, interpreted by A as a key, and $D_2(X)$ the second part of the decryption of X, interpreted by A as a challenge. Then A will respond with

$$(D_1(X))(challenge_A, D_2(X)),$$

and expect a message of the form

$$(D_1(X))(challenge_A)$$

in reply. Unless X is in fact $E_A(R, challenge_B)$ and

$$(D_1(X))(challenge_A) = R(challenge_A)$$

was obtained from B, the adversary should neither be able to produce a message of the form

$$(D_1(X))(challenge_A)$$

nor to obtain any useful information about E_A or R.

The adversary has messages $P(E_A)$, X, and $(D_1(X))(challenge_A)$ to work with in mounting such an attack. We will permit the adversary to consider the dictionary of all possible P's exhaustively in mounting this attack. This means in particular that for all (or an overwhelming majority) of the dictionary entries P', $P'^{-1}(P(E_A))$ must be (or appear to be) a valid public key.

A similar analysis, considering possible attacks on B, shows that an adversary should not be able to produce a message of the form

$$R(challenge_A, challenge_B)$$

from messages X, and $P^{-1}(X)(R, challenge_B)$, unless X is obtained from A and is of the form $P(E_A)$.

2.5.3 Strengthening EKE against cryptanalytic attacks

Suppose that a cryptanalyst has recovered some session key R. This provides a hook for attacks on P. A direct cryptanalytic attack on E_A could be attempted; alternatively, knowledge of R can be used to validate guesses P' at P. That is, we can test whether $E_A(R)$, which is transmitted, matches $(P'^{-1}(P(E_A)))(R)$. A minor variation in the protocol can prevent this.

During the challenge/response dialog, let A and B generate random subkeys S_A and S_B. These are transmitted encrypted by R. Message (RPK.3) then becomes

$$R(challenge_A, S_A),$$

while message (RPK.4) becomes

$$R(challenge_A, challenge_B, S_B).$$

The two parties then calculate a true session key $S = f(S_A, S_B)$ for some suitable function f. This key is used for all subsequent exchanges; R is reduced to the role of of a *key exchange key*.

Observe how this protects us. A recovered value of S tells us nothing about P, because P is never used to encrypt anything that leads directly to S. Nor is a cryptanalytic attack on R feasible; R is used only to encrypt random data, and the one hint — S — never appears in any message.

Conceivably, a sophisticated cryptanalyst might be able to use the presence of challenges and responses in different messages to attack R. This seems unlikely; however, if it is of concern, we can modify the responses to contain a one-way function of the challenges, rather than the challenges themselves. Thus, message (RPK.4) would become

$$R(g(challenge_A), challenge_B, S_A).$$

A similar change would be made to message (RPK.5).

3 EKE using exponential key exchange

The use given above for asymmetric encryption — simply using it to pass a key for a symmetric encryption system — is an example of what Diffie and Hellman[2] call a *public key distribution system.* In the same publication, they describe the use of exponential key exchange as a public key distribution system. It is in some sense a weaker paradigm than asymmetric encryption; exponential key exchange does not provide authentication. Furthermore, it is vulnerable to active wiretaps and "man in the middle" attacks [10]. However, by encrypting the transmitted quantities with a secret key P, we can solve both of these problems.

1. As before, A calculates $\alpha^{R_A}\ (mod\ \beta)$, but transmits
$$A, P(\alpha^{R_A}\ (mod\ \beta)) \qquad \text{(DH.1)}$$
Since R_A is random, $\alpha^{R_A}\ (mod\ \beta)$ is random; hence guesses at P yield no information.

2. Similarly, B transmits
$$P(\alpha^{R_B}\ (mod\ \beta)) \qquad \text{(DH.2)}$$

3. Both sides, knowing P, can retrieve the exponentials, and calculate the session key. An intruder cannot, and hence cannot sit in the middle. Nor are attempts to guess P off-line useful; even a successful guess will yield only $\alpha^{R_A}\ (mod\ \beta)$ and $\alpha^{R_B}\ (mod\ \beta)$, which by our assumptions provide no useful information about the session key.

3.1 A complete protocol

Using EKE with exponential key exchange is quite similar to using it with any conventional asymmetric cryptosystem. However, since the key exchange process in itself produces a random session key, no separate transmission of R is needed. Without further ado, we present the protocol.

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

   A sends
$$A, P(\alpha^{R_A}\ (mod\ \beta)) \qquad \text{(RDH.1)}$$
   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
$$P(\alpha^{R_B}\ (mod\ \beta)), K(challenge_B). \qquad \text{(RDH.2)}$$

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
$$K(challenge_A, challenge_B). \qquad \text{(RDH.3)}$$

4. B decrypts $K(challenge_A, challenge_B)$, and verifies that $challenge_B$ was echoed correctly.

   B sends
$$K(challenge_A). \qquad \text{(RDH.4)}$$

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

As before, it is possible to omit encryption of one of the exponentials. For example, in the protocol shown above, message (RDH.1) could be replaced by
$$A, \alpha^{R_A}\ (mod\ \beta).$$
An attacker will not be able to decode the response from B, and hence will gain no information. Nor will an active attack succeed, substituting a new value for A's exponential; the enemy cannot respond to the challenge in message (RDH.2) without knowing the true value of R_A.

A caveat should be mentioned. If the attacker can select 0 as an exponent, causing
$$\alpha^{R_A R_B}\ (mod\ \beta) \equiv 1.$$
This gives away K, and permits imposture. Fortunately, this attack is easily detected; however, we do not know if other special exponents exist.

3.2 Choosing α and β

Thus far, we have said nothing about how to choose α and β, either for exponential key exchange or for ElGamal. There are a variety of possibilities, offering a range of tradeoffs between cost and security.

Although there are a number of possible choices for the modulus, fairly large prime values of β are more secure [11]. Furthermore, it is desirable that α be a primitive root of the field $GF(\beta)$. If we choose β such that
$$\beta = 2p + 1$$

for some prime p, there are $(\beta - 1)/2 = p$ such values; hence, they are easy to find. We assume those restrictions in the discussion that follows.

Our basic problem, when deciding how A and B know which values of α and β to use, is to avoid leaking information. As noted, we obviously cannot transmit $P(\beta)$; testing a random value for primality is too easy. One good choice for EKE is to make α and β fixed and public. There is thus no risk of information leakage or partition attacks. The disadvantage is that implementations become less flexible, as all parties must agree on such values. Furthermore, to maintain security, β must be quite large, which in turn makes the exponentiation operations expensive.

Some compromise in the length of the modulus is possible, however. Though LaMacchia and Odlyzko[12] suggest 1000-bit values, they are assuming that the exponentials are available to the attacker. With EKE, the password P is used to superencrypt such values; it is not possible to essay a discrete logarithm calculation except for all possible guesses of P. Our goal is thus to select a size for β sufficient to make guessing attacks far too expensive. Using 200 bits, for which discrete logarithm solutions are estimated to take several minutes (after modulus-specific preprocessing), could suffice.

Another consideration inclines one towards larger moduli, however. If the user's password is ever compromised, recorded exponentials will be available to the attacker; these, if solved, will permit reading of old conversations. If a large modulus value is used, all such conversations will remain secure.

Size requirements for β are derived from a desire to prevent calculations of discrete logarithms in the field $GF(\beta)$. The current best algorithms for such calculations all require large amounts of precalculation. If a different β is used each time, an attacker cannot build tables in advance; thus, a much smaller, and hence cheaper, modulus can be used. Therefore, we suggest that A generate random values of β and α, and transmit them in cleartext during the initial exchange. There is little security risk associated with an attacker knowing these values; the only problem would be with cut-and-paste attacks. And even this risk is minimal if B performs certain checks to guard against easily-solvable choices: that β is indeed prime, that it is large enough (and hence not susceptible to precalculation of tables), that $\beta - 1$ have at least one large prime factor (to guard against Pohlig and Hellman's algorithm[13]), and that α is a primitive root of $GF(\beta)$. The latter two conditions are related; we must know the factorization of $\beta - 1$ in order to validate α. Requiring that β be of the form $kp + 1$, where p is prime and k a very small integer, solves both problems.

Recent results[14] do suggest that it may be possible to choose a β that contains a hidden trap door. At the moment, this attack does not seem to be practical. If that should change, other choices would, of course, be preferable.

Thus far, we have said nothing about choosing α. But if a suitable value of β is chosen, finding primitive roots is very easy. There is no reason not to examine the integers starting with 2; the density of primitive roots guarantees that one will be found quite quickly.

4 The encryption layers

4.1 Selecting symmetric cryptosystems

Symmetric encryption is used in three places with EKE: to encrypt the initial asymmetric key exchange, to trade challenges and responses, and to protect the ensuing application session. Each of these has different requirements, though in general the same cryptosystem can be used.

In the initial exchange, there are severe constraints on the plaintext encrypted. Fairly obviously, the messages must *not* use ASN.1 [15, 16] or any other form of tagged data representation; if they did, the sanity of the decrypted tags could be used to validate a guess at P.

More subtly, the original plaintext message cannot contain any non-random padding to match the encryption blocksize, nor can it contain any form of error-detecting checksum [17]. Otherwise, an attacker could use these indicators when guessing at P. Protection against communications errors must be provided by lower-layer protocols. While one normally employs cipher block chaining or some similar scheme to tie together multiple blocks, such mechanisms are not particularly important here; the bits being transmitted are random, and cannot profitably be manipulated by an attacker. The challenge/response protocol provides the necessary defense against such manipulation of the messages.

Curiously enough, the encryption algorithm may be quite weak; even as simple an operation as *XOR*-ing the password with the public key will suffice. The reasons are simple. Anything that obscures the public key will provide the necessary level of authentication. And, since the key being sent is random, it provides the necessary level of concealment of the password.

There is, however, a significant disadvantage to using such a simple scheme. If the public key, random and transient though it may be, should ever be disclosed, the intruder will instantly know the password. Consequently, we recommend using a stronger encryption algorithm.

Similarly, the challenge/response messages do not need to be protected by a strong cipher system. However, we have tacitly assumed that it is not feasible for an attacker to perform useful cut-and-paste operations on encrypted messages. For example, when we say that A sends $R(challenge_A, challenge_B)$ to B, and that B replies with $R(challenge_A)$, one might conclude that the attacker could snip out $R(challenge_A)$ from the first message, and simply echo it in the second. This must be prevented, of course. Thus, if necessary in the particular cryptosystem being used, standard techniques such as cipher block chaining should be employed. Alternatively, A and B could use R to derive distinct subkeys R_A and R_B, each used in only one direction. Other possibilities include employing message typing or adding message authentication codes; however, these may introduce redundancy undesirable in the face of a cryptanalytic attack. (Note also the potential problems when using typed messages with disclosing encryption systems.) In such situations, the one-way functions mentioned in Section 2.5.3 may be preferable.

Finally, the use of R in the ensuing login session must not reveal useful information about R. If the system is cryptanalyzed and R is recovered, the attacker can then mount a password-guessing attack on the EKE exchange. Furthermore, since this protocol is being suggested for protecting arbitrary sessions between parties, it is best to be cautious, and examine the particular symmetric system under the assumption that the adversary may mount chosen-ciphertext attacks against the session. If there is any doubt, the separate key exchange key should be used.

4.2 Selecting an asymmetric cryptosystem

In principle, EKE can be used with any asymmetric cryptosystem. In reality, some systems may be ruled out on practical grounds. For example, a system that used many large primes would be infeasible. RSA requires at least two such primes; dynamic key generation might be too expensive on today's hardware.

A second consideration is whether or not a particular system's public keys can be encoded as a random-seeming bit string. We have already seen how this can be an issue for RSA; conceivably, asymmetric systems exist for which there is no easy solution.

It is tempting to finesse the issue by instead transmitting the seed of the random number generator used to produce the public key. Unfortunately, that does not work. Apart from the expense involved — both sides would have to go through the time-consuming process of generating the keys — the random seed will yield both the public and private keys. And that in turn would allow an attacker to validate a candidate password by retrieving the session key.

The same option does work with exponential key exchange. Since the prime modulus may be public anyway, there is nothing to be concealed. Unfortunately, it requires both parties to go through the expense of generating large prime numbers, albeit while saving on the size modulus required. The tradeoff may be worth reconsidering if very fast solutions to the discrete logarithm problem are found.

Regardless, we do recommend careful analysis of whichever asymmetric encryption system is chosen. The constraint we impose — that encryption of a random quantity not leak information — is rather different than has been required in the past. Put another way, the questions we are asking have not been asked in the past; hence, the answers are not readily available. For example, we are unaware of any other discussion of disclosing encryption systems. Additionally, it is entirely possible that number-theoretic attacks would succeed against particular cryptosystems when used with EKE, even if they are secure for other applications.

One last caveat should be mentioned. It is vital that the symmetric and asymmetric cryptosystems used not be associative. That is, they must be chosen so that in general

$$P(E_A(R)) \neq E_{P(E_A)}(R).$$

Otherwise, an attacker can use the value learned from message (EKE.1) to encrypt a selected R in the following message, with obvious deleterious consequences. Associativity does not appear to be a concern with the cryptosystems we have discussed, but interactions are certainly conceivable.

5 Applications

As noted earlier, a primary motivation for the creation of EKE was the problem of authenticating a user to a host. However, there are other uses as well. Perhaps the most interesting application for EKE is secure public phones.

Let us assume that encrypting public telephones are deployed. If someone wishes to use one of these phones, some sort of keying information must be provided. Conventional solutions — i.e., the STU-III secure voice/data telephone — require that the caller possess a physical key. This is undesirable in many situation. EKE permits use of a short, keypad-entered password, but uses a much longer session key for the call.

EKE would also be useful with cellular phones. Fraud has been a problem in the cellular industry; EKE can defend against it (and ensure the privacy of the call) by rendering a phone useless if a PIN has not been entered. Since the PIN is not stored within the phone, it is not possible to retrieve one from a stolen unit.

EKE also provides a replacement for Rivest and Shamir's *Interlock Protocol* [18]. This protocol is designed to detect active eavesdroppers. If the interlock protocol is used for authentication, as suggested by Davies and Price [19, page 222], certain attacks are possible, as we have shown elsewhere [20]. Our attack does not succeed against EKE.

From a general perspective, EKE functions as a *privacy amplifier*. That is, it can be used to strengthen comparatively weak symmetric and asymmetric systems when used together. Consider, for example, the key size needed to maintain security when using exponential key exchange. As LaMacchia and Odlyzko have shown [12], even modulus sizes once believed to be safe (to wit, 192 bits) are vulnerable to an attack requiring only a few minutes of computer time. But their attack is not feasible if one must first guess a password before applying it.

Conversely, the difficulty of cracking exponential key exchange can be used to frustrate attempts at password-guessing. Password-guessing attacks are feasible because of how rapidly each guess may be verified. If performing such verification requires solving an exponential key exchange, the total time, if not the conceptual difficulty, increases dramatically. Assume, for example, that a modulus size was picked so that LaMacchia and Odlyzko's method would take 5 seconds. Testing all possible passwords composed solely of five lower-case letters would then take more than two years. (Note, though, that password-guessing programs rely on more sophisticated techniques, such as lists of common names. One should still use a longer modulus length to maintain security.)

6 Related work

Lomas *et al.* [21] present a different protocol with the same goals. They introduce the valuable concept of *verifiable plaintext*, a more formal definition of the random plaintext constraint we mandate. The paper also presents a very clear and compelling argument for why protocols that prevent password-guessing attacks are needed. Gong refines the definition of verifiable plaintext in [22]. The protocols in these two papers are designed to operate via a trusted third party whose public key is known to both A and B. They could be simplified if one assumes that the server and B are one and the same, as our model assumes; however, that would require that A know B's public key. For some of the applications we have described above, this is not feasible. As noted, EKE simply requires that the two parties share a password. The variation presented in [21] requires that the two parties have roughly synchronized clocks; again, this is not always possible.

The essential insight in these papers is that if a plaintext block containing the user's password also contains a random quantity, the encryption of that block via an asymmetric cryptosystem and the key server's secret key is immune to password-guessing. No direct decryption by the enemy is possible, of course, and attempts to validate a guess at the password by trial encryptions will fail, since the attacker cannot produce the exact plaintext block. As in our scheme, extra complexity is needed to guard against replays, known plaintext attacks, etc.

The same idea is used by the SPX[23] authentication system. Additionally, it utilizes two different one-way hashes of a user's password, rather than the password itself; thus, neither the "LEAF" intermediary nor the certificate distribution center itself need know the actual password.

7 Conclusions

We have presented a novel protocol relying on the counter-intuitive notion of using a secret key to encrypt a public key. There are a number of applications for this that are immediately apparent; we speculate that there may be others as well.

Our main goal, however, is to protect users with weak passwords. We expect that some people will object that we have provided a solution without a problem. In a world of smart cards, hand-held authenticators, and zero-knowledge proofs, it seems pointless to be worrying about poorly-chosen passwords. Were the world like that, we might agree. Today, it is not.

Empirically, weak passwords are fact of life. Attempts to strengthen users' passwords by enforcing syntactic restrictions have not been notably successful; audits still turn up many weak passwords. Klein [7], for example, cracked 25% of a database of 15,000 password entries; others report similar success rates. The problem is so serious that many versions of the UNIX[4] operating system have been forced to read-protect the file containing users' passwords, despite the system's use of a one-way function.[5]. But that approach does not protect networked systems. Only a protocol like EKE will solve that problem.

8 Acknowledgements

We would like to thank Jeff Lagarias, Andrew Odlyzko, Joan Feigenbaum, and Jim Reeds for their assistance, especially with number-theoretic problems. Jason De Mont helped us expand the original scope of the idea. Li Gong made a number of helpful observations, especially about the need for non-associativity of the cryptosystems.

References

[1] J. Steiner, B. C. Neuman, and J. I. Schiller, "Kerberos: An authentication service for open network systems," in *Proc. Winter USENIX Conference*, (Dallas), 1988.

[2] W. Diffie and M. E. Hellman, "New directions in cryptography," *IEEE Transactions on Information Theory*, vol. IT-11, pp. 644–654, November 1976.

[3] R. L. Rivest, A. Shamir, and L. Adleman, "A method of obtaining digital signatures and public-key cryptosystems," *Communications of the ACM*, vol. 21, pp. 120–126, February 1978.

[4] T. ElGamal, "A public key cryptosystem and a signature scheme based on discrete logarithms," *IEEE Transactions on Information Theory*, vol. IT-31, pp. 469–472, July 1985.

[5] R. H. Morris and K. Thompson., "Unix password security," *Communications of the ACM*, vol. 22, p. 594, November 1979.

[4] UNIX is a registered trademark of UNIX Systems Laboratories, Inc.

[6] F. T. Grampp and R. H. Morris, "Unix operating system security," *AT&T Bell Laboratories Technical Journal*, vol. 63, pp. 1649–1672, October 1984.

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

[8] P. Leong and C. Tham, "Unix password encryption considered insecure," in *Proc. Winter USENIX Conference*, (Dallas), 1991.

[9] D. E. Denning, *Cryptography and Data Security*. Addison-Wesley, 1982.

[10] R. DeMillo and M. Merritt, "Protocols for data security," *Computer*, vol. 16, pp. 39–50, February 1983.

[11] A. M. Odlyzko, "Discrete logarithms in finite fields and their cryptographic significance," in *Proceedings of Eurocrypt '84*, pp. 225–314, 1984.

[12] B. A. LaMacchia and A. M. Odlyzko, "Computation of discrete logarithms in prime fields," *Designs, Codes, and Cryptography*, vol. 1, pp. 46–62, 1991.

[13] S. C. Pohlig and M. Hellman, "An improved algorithm for computing logarithms over $GF(p)$ and its cryptographic significance," *IEEE Transactions on Information Theory*, vol. IT-24, pp. 106–110, 1978.

[14] A. M. Odlyzko, June 1991. Private conversation.

[15] International Organization for Standardization and International Electrotechnical Committee, *Information Processing Systems – Open Systems Interconnection – Specification of Abstract Syntax Notation One (ASN.1)*, 1987. International Standard 8824.

[16] International Organization for Standardization and International Electrotechnical Committee, *Information Processing Systems – Open Systems Interconnection – Specification of Basic Encoding Rules for Abstract Syntax Notation One (ASN.1)*, 1987. International Standard 8825.

[17] L. Gong, "A note on redundancy in encrypted messages," *Computer Communications Review*, vol. 20, pp. 18–22, October 1990.

[18] R. L. Rivest and A. Shamir, "How to expose an eavesdropper," *Communications of the ACM*, vol. 27, no. 4, pp. 393–395, 1984.

[19] D. W. Davies and W. L. Price, *Security for Computer Networks*. John Wiley & Sons, second ed., 1989.

[20] S. M. Bellovin and M. Merritt, "An attack on password-authenticated exponential key exchange," *IEEE Transactions on Information Theory*, to appear.

[21] T. M. A. Lomas, L. Gong, J. H. Saltzer, and R. M. Needham, "Reducing risks from poorly chosen keys," in *Proceedings of the Twelfth ACM Symposium on Operating Systems Principles*, pp. 14–18, SIGOPS, December 1989.

[22] L. Gong, "Verifiable-text attacks in cryptographic protocols," in *Proceedings of the IEEE INFOCOM '90 – The Conference on Computer Communications*, pp. 686–693, 1990.

[23] J. J. Tardo and K. Alagappan, "SPX: Global authentication using public key certificates," in *Proc. IEEE Computer Society Symposium on Research in Security and Privacy*, (Oakland), pp. 232–244, May 1991.