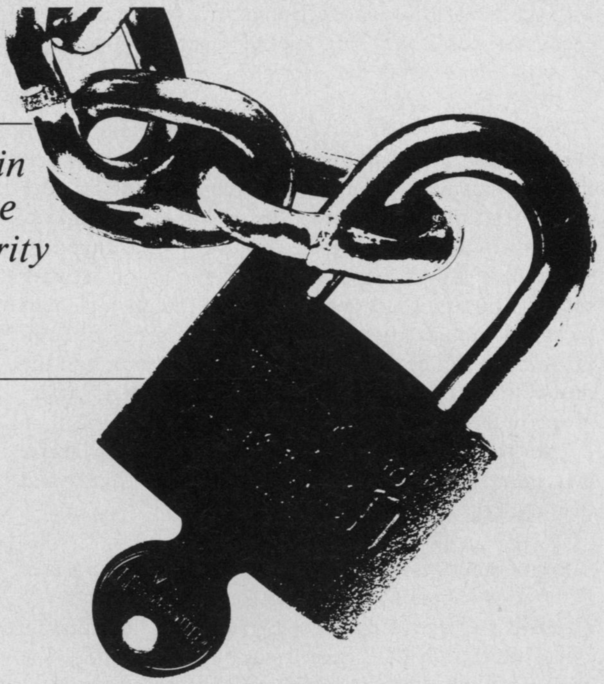

Public-key cryptography offers certain advantages, providing the keys can be adequately protected. For every security threat there must be an appropriate countermeasure.

Protecting Public Keys and Signature Keys

Dorothy E. Denning, Purdue University

With conventional one-key cryptography, the sender and receiver of a message share a secret encryption/decryption key that allows both parties to encipher (encrypt) and decipher (decrypt) secret messages transmitted between them. By separating the encryption and decryption keys, public-key (two-key) cryptography has two attractive properties that conventional cryptography lacks: the ability to transmit messages in secrecy without any prior exchange of a secret key, and the ability to implement digital signatures that are legally binding. Public-key encryption alone, however, does not guarantee either message secrecy or signatures. Unless the keys are adequately protected, a penetrator may be able to read encrypted messages or forge signatures.

This article discusses the problem of protecting keys in a nationwide network using public-key cryptography for secrecy and digital signatures. Particular attention is given to detecting and recovering from key compromises, especially when a high level of security is required.

Public-key cryptosystems

The concept of public-key cryptography was introduced by Diffie and Hellman in 1976.[1] The basic idea is that each user A has a public key E_A, which is registered in a public directory, and a private key D_A, which is known only to the user. E_A is the key to a public enciphering transformation, which is also written as E_A. D_A is the key to a private deciphering transformation D_A, which is related to E_A but cannot be computationally determined from E_A. We assume the public-key system is unbreakable; in particular, a cryptanalyst cannot determine a secret key from intercepted ciphertext even when the corresponding plaintext is known to or chosen by the cryptanalyst.

Consider an application environment in which each user has an intelligent terminal or personal workstation where his private key is stored and all cryptographic operations are performed. This terminal is connected to a nationwide network through a shared host, as shown in Figure 1. The public-key directory is managed by a network key server. Users communicate with each other or

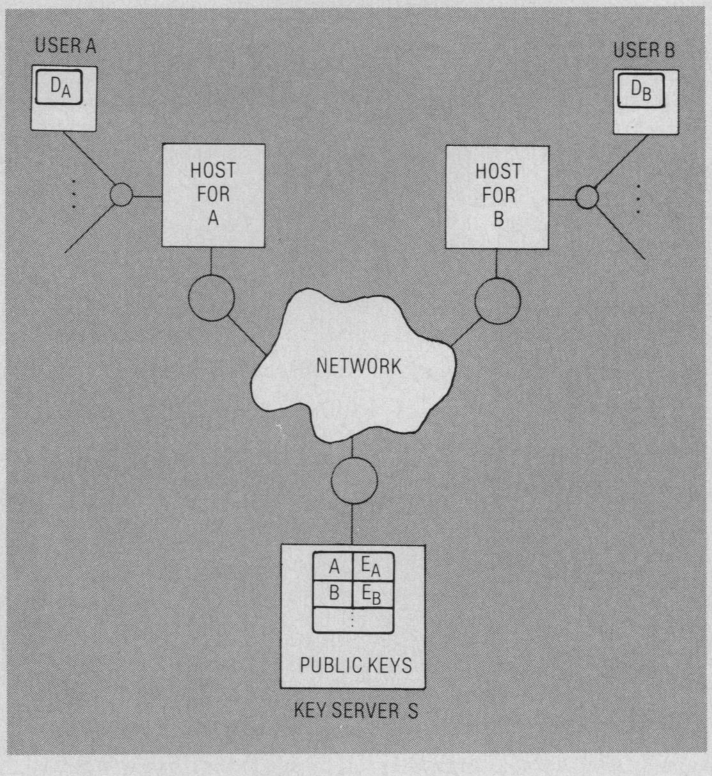

Figure 1. Network with key server.

with the server through electronic mail. When a user A needs the public key for another user B, A sends a message to the server requesting E_B. A can then keep a record of E_B for future use.

We assume the key server is trustworthy and returns correct and current public keys. We do not, however, make any assumptions about the security of the network or the hosts on the network. The system may be vulnerable to wiretapping along the communication channels, or possible subversion of operating system or network software on some host (e.g., through "Trojan horse" attacks). Although little can be done to prevent wiretapping, much can be done to strengthen the security of the hosts.[2] Many of the threats to be described will not affect properly secured systems.

The public-key system is used for sending secret messages, signing plaintext (cleartext) messages, and sending signed secret messages.

Secret messages. To send a secret message M to a user B, user A obtains B's public key E_B, encrypts the plaintext message M as $C := E_B(M)$, and transmits the ciphertext C to B. (If M exceeds the block size for E_B, it is broken into blocks and each block is encrypted separately.) B's private transformation D_B is the inverse of E_B, so that B can decipher C and obtain M by computing $D_B(C) = M$ (see Figure 2). If the cryptosystem is secure, secrecy is possible under the following two conditions:

(1) No other user knows D_B.

(2) There is enough uncertainty about M (i.e., M has enough entropy). The encryption key E_B is public, so if only a few likely candidates $M_1, \ldots, M_n$ exist for M, then M can be found by enciphering these candidates until an M_i is found that enciphers to the same C; that is, $E_B(M_i) = C$, wherefore $M = M_i$. (Simmons and Holdridge show how this might be done with voice encryption.[3]) One-key systems are not vulnerable to such exhaustive searches for M because the encryption key is secret.

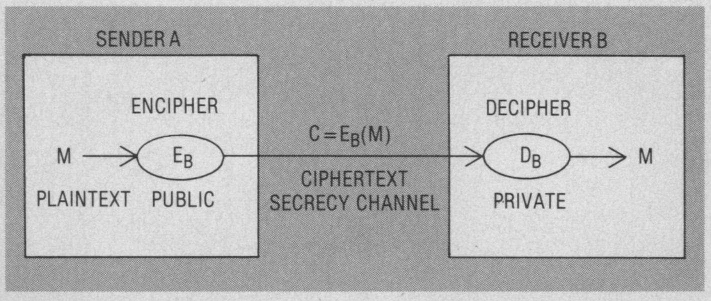

Figure 2. Using a public-key system for message secrecy.

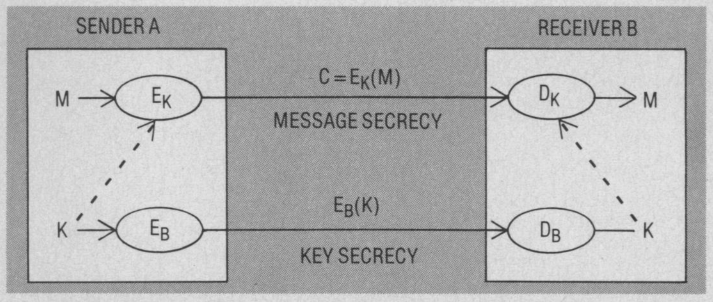

Figure 3. Using a single-key system for message secrecy and a public-key system for key exchange.

With encryption alone, B cannot be sure the received message is the one sent from A, because an active wiretapper could obtain E_B and alter A's message. He might even impersonate A. To give B this assurance, the message must be signed by A.

Signed messages. To send a signed message M to B, user A applies the private transformation D_A to M. Ignoring the issue of secrecy for the moment, A computes and transmits to B the digital signature $X := D_A(M)$. A's public transformation E_A is the inverse of D_A, so that B (or a judge) can validate A's signature on an alleged message M by checking whether $E_A(X) = M$. X serves as A's signature on M only if:

(1) No other user knows D_A and hence cannot compute $D_A(M)$ for a message M of the user's choice.

(2) Legitimate messages have enough redundant information to be easily distinguished from forgeries by the receiver and by the judge. This condition is necessary because E_A is public; thus, any user could claim an arbitrary X to be the signature for a message M computed by $M := E_A(X)$, since this implies $X = D_A(M)$. But a message M computed in this manner will be meaningless and appear random. If legitimate messages are required to be meaningful English-language text or to have some other redundant information such as a time stamp, forgeries are easily detected.

(3) Given the signature X, the odds of finding another meaningful message M' with signature X, using available computing resources, are negligible.

Note that secrecy is not achieved with a digital signature alone, because any user can obtain E_A and compute M. We will consider the problem of transmitting a signed message in secrecy shortly and at that time look at an improved method for signing messages.

Secrecy requires that an enciphering transformation E_A operate on plaintext and that the corresponding deciphering transformation D_A invert E_A. Signatures, on the other hand, require that a deciphering or digital signature transformation D_A operate on plaintext and that the enciphering transformation E_A invert D_A. The only known public-key cryptosystem meeting both of these requirements is the Rivest-Shamir-Adleman, or RSA, scheme,[4] which is based on the difficulty of factoring. For user A, the public enciphering transformation E_A is given by

$$E_A(M) = M^{e_A} \bmod n_A ,$$

and the private deciphering transformation D_A by

$$D_A(C) = C^{d_A} \bmod n_A ,$$

where the modulus n_A is the product of two large secret primes p_A and q_A, and exponents e_A and d_A are mutual multiplicative inverses modulo the product $(p_A - 1)(q_A - 1)$. (This product is the Euler totient function of n_A.) The secret exponent d_A cannot be obtained from the public exponent e_A without knowing the factors p_A and q_A of n_A. Recently, Davida[5] has shown that the RSA scheme has a potential weakness that might allow a cryptanalyst to obtain the plaintext corresponding to intercepted ciphertext or to obtain signatures on hidden messages. The protocol to be described shortly overcomes this weakness.

The other known public-key cryptosystems can be used for either secrecy or signatures, but not both. These include the Merkle-Hellman[6] and Graham-Shamir[7,8] trapdoor knapsack schemes, Shamir's signature-only knapsack scheme,[9] McEliece's error-correcting code scheme,[10] and Janardan's matrix cover scheme.[11] Recently, Shamir has published a polynomial time algorithm for breaking the Merkle-Hellman scheme,[12] raising serious doubts about any knapsack-based or related scheme, including the matrix cover scheme. Moreover, since these systems are incapable of providing both secrecy and signatures,[13] we will assume that the public-key system is the RSA system.

We could use separate cryptosystems for secrecy and signatures, in which case each user would have a pair of transformations, one for receiving secret messages and one for sending signed messages. The procedures described in this article are easily modified to accommodate this.

We can also use a one-key system such as the Data Encryption Standard[14] for message secrecy and reserve the public-key system for signatures and key exchange. To use a one-key system for message secrecy, the sender A picks a key K to a pair of invertible transformations, E_K for enciphering and D_K for deciphering. The key K is then transmitted to the receiver B in secrecy by encrypting it under B's public key; that is, A computes and transmits $E_B(K)$ to B. A then enciphers M using E_K, and B deciphers the received ciphertext using D_K (see Figure 3). Similarly, B can encipher a reply using E_K, which A can decipher using D_K. The advantage of this approach over the public-key cryptosystem for message secrecy is primarily speed. The encryption/decryption rate of the RSA scheme, for example, is several orders of magnitude slower than that of the DES. Note, however, that a one-key system cannot by itself provide signatures, because the receiver of a message, knowing the encryption key, could forge the message. But it does provide message authenticity in that the receiver can be sure the message sent from A has not been altered. In this sense, it provides a greater level of protection than using a public-key system for message secrecy only.

Signed secret messages. There are two approaches to sending a signed secret message M from A to B. The first composes the secrecy and signature transformations, while the second keeps the transformations separate.

Composed secrecy and signature functions. With this approach, A first encrypts M using B's public key and then signs the result, getting $Y := D_A(E_B(M))$. B recovers M by first applying A's public key E_A to Y and then applying D_B to the result. This approach can also be employed when a one-key system is used for message secrecy. Here, A encrypts M under a key K shared with B and then signs the result, getting $Y := D_A(E_K(M))$. (Note that if a dispute arises, B must reveal K to a judge to prove that applying E_A and then D_K to Y yields the alleged message M.)

With either type of cryptosystem, this approach is simple though somewhat inconvenient; B must remove the signature transformation to access M. If the signature must be retained by B over an extended period to settle possible disputes, B can speed up processing by also storing the unsigned message $E_B(M)$ (or just M if encryption is not needed for secure storage). This approximately doubles the storage requirements for the message, however.

Separate secrecy and signature functions. This approach keeps the secrecy and signature transformations separate and uses another strategy that conceals the message M in the signature but has minimal storage requirements, namely one block of output from D_A. Davies and Price[15] show how this can be done by computing a "digest" or "checksum" of M, which we denote by $\overline{M}$. A picks a random initialization seed I and first computes the digest $\overline{M} := h(M,I)$, where h is a publicly available hashing function, described shortly. A then signs the digest $\overline{M}$ together with I as $X := D_A(\overline{M}, I)$, and transmits the signature X to B. (If the pair $(\overline{M},I)$ is shorter than the block size used by D_A, it is replicated as many times as necessary to fill the block; the extra bits are discarded by B after computing $E_A(X)$.) In addition, A computes and transmits to B the encrypted message $C := E_B(M)$ (or $E_K(M)$ if a one-key system is used for secrecy). After recovering M, B (or a judge) validates A's signature X by computing $E_A(X) = (\overline{M},I)$ and checking that this $\overline{M}$ is the same as that obtained by computing $h(M,I)$ using the alleged message M.

Figure 4 illustrates the procedure for sending a signed secret message using the public-key system for message secrecy. If a single-key system is used for secrecy, then the secrecy channel would be that shown in Figure 3. If secrecy is not needed or is undesirable, the secrecy channel can be omitted entirely and M can be transmitted in the clear.

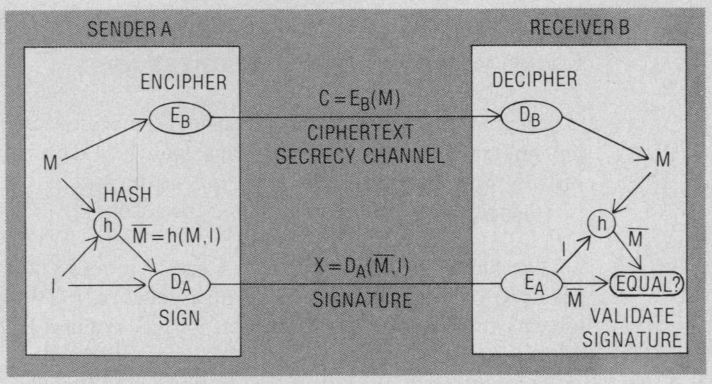

Figure 4. Sending a signed secret message from *A* to *B*.

The hashing function. The hashing function h must satisfy two properties. First, to ensure the secrecy of M, h must be a one-way function of M. Otherwise, the contents of M could be determined by an eavesdropper, who could obtain E_A, compute $E_A(X) = (\bar{M}, I)$, and finally compute $h^{-1}(\bar{M}, I) = M$. Second, to prevent forgeries, it must not be possible to find another message with the same digest $\bar{M}$ for the same seed I.

Finding such a function h that satisfies both properties is tricky because of the need to make h public. Davies and Price show that many techniques that are secure when h is secret are vulnerable to attack when h is public. They suggest a possible hashing function h that uses the DES (or some other block encryption algorithm) as follows: First, a random 64-bit seed I is picked; next, M is blocked into 56-bit blocks $M_1, \ldots, M_t$; finally, $h(M, I)$ is computed using each message block as a key to the DES:

$$\bar{M} = h(M, I) = E_{M_t} \circ \cdots \circ E_{M_1} \circ E_{M_t} \circ \cdots \circ E_{M_1}(I),$$

where E_{M_i} denotes the DES keyed to message block M_i, and "o" denotes functional composition. (The message is used twice to prevent a "meet-in-the-middle" forgery; this can also be achieved with a one-pass scheme.[16])

This method of signing a message M has four advantages over the standard technique of computing $D_A(E_B(M))$ (or just $D_A(M)$ if secrecy is not needed).

(1) It separates the signature transformation from the secrecy transformation, allowing secrecy to be implemented with a one-key system or to be skipped.[15] At the same time, it conceals messages so that signatures can be publicly disclosed without revealing their corresponding messages. This will be important later in the discussion of why signatures should be recorded in a public log.

(2) It reduces the need for adding redundancy to messages to prevent forgeries, because it is difficult to find even a random message that hashes to any given digest $\bar{M}$ starting with the same seed I.

(3) It reduces the storage requirements for signatures to a single block; thus, retaining signatures in a public log need not require much space.

(4) It seems to overcome the potential weakness Davida discovered in the RSA system,[16] because the hashing function is one-way and destroys the multiplicative structure of messages.

Security threats and countermeasures

The preceding schemes for secrecy and signatures cannot guarantee security if public keys can be faked or private keys compromised. This is true regardless of whether the keys are used for secrecy or signatures.

Fake public keys. Suppose that a user B requests A's public key from the key server but a penetrator Z intercepts and modifies the server's response to contain E_Z instead of E_A. This might be done by active wiretapping or by penetrating B's host. If E_A is requested to transmit a secret message M to A, then B will inadvertently form the ciphertext $E_Z(M)$ and transmit this to A; Z then intercepts the ciphertext and deciphers the message. If A receives an indecipherable message, A will detect that something is wrong and notify B; even so, Z may have already acquired valuable information from B. To protect against this threat, A and B could agree on a handshake procedure[17] whereby the key is tested before valuable information is transmitted. This can introduce long delays into the protocol, however, if A is not logged in when B sends the request for a handshake.

Now, if E_A is requested to validate A's signature on a message M received from A, then B will unwittingly accept Z's signature instead. Even though the signature would not be legally binding, B may feel satisfied that it belongs to A and accept the message.

In both the secrecy and signature cases, Z has acquired some of A's capabilities. In the former case, these capabilities allow Z to receive secret messages intended for A; in the latter case, they allow Z to create documents that appear to have been signed by A. If the system is sufficiently insecure that Z can prevent messages from reaching A and create messages appearing to have originated with A, then Z may be able to masquerade as A. This scenario is not as farfetched as it might seem. If B's host is insecure, it may be simple to modify the headers of messages to reroute them or to change their source.

There are two techniques for protecting against fake public keys: mass distribution and certificates. Mass distribution simply recognizes that because the keys are public, the complete directory can be regularly distributed (e.g., weekly or monthly) as hard copy through a public newsletter or a widely distributed newspaper such as the New York Times. Updates to the public-key directory could be distributed on a cumulative basis.

Public-key certificates. This technique, suggested by Kohnfelder,[18] is for the key server to distribute public keys inside signed certificates. The public-key certificate for user A contains a time stamp T, A's unique identifier (which for simplicity can be written as A), and A's public key E_A, all signed with the server's private key D_S:

$$P := D_S(T, A, E_A).$$

The receiver of certificate P applies the server's public key E_S to P, obtaining T, A, and E_A. Note that the server does not hash the message (T, A, E_A) before signing. This raises some question about whether a penetrator might be able to forge the server's signature using the method discovered by Davida. This seems very unlikely, however, if the only messages signed by the server are time-stamped certificates that it has constructed.

The time stamp T is used for validating the currency of the certificate. Without the time stamp, a user might accept the replay of an old certificate containing the public key E_A for a private key D_A that has been compromised.[19] If the user then enciphers secret messages under E_A, the compromiser could read those messages. (Kohnfelder's proposal did not include time stamps.) Note that off-line publication of public keys in a newspaper automatically time stamps all keys.

Whereas mass distribution of public keys allows off-line validation of keys, public-key certificates allow on-line validation. Both techniques can be used for added security. Even if certificates alone are used for user keys, the server's public key E_S should be publicly broadcast to all users and retained at the users' workstations. Otherwise, if an intruder Z can trick a user into accepting E_Z as the server's key, then Z can send fake public-key certificates to the user.

Compromise of users' private keys. Suppose that a penetrator Z has acquired A's private key D_A. The method by which D_A may have been acquired—by breaking into A's terminal, by sloppy handling procedures for keys, or by cryptanalysis, for instance—does not concern us here. Even if this is unlikely, any practical system must be capable of recovering from such compromises.

Now, if the corresponding public key E_A is used by others to send secret messages to A, then through passive wiretapping Z can intercept and decipher these messages without detection. If D_A is used by A to sign messages, Z can forge A's signature on other messages. Once again Z has acquired some of A's capabilities; in the first case they allow Z to receive secret messages intended for A, and in the second case they allow Z to create documents allegedly signed by A.

Note the parallel between this problem and the problem of fake public keys. If Z can fake E_A or compromise D_A, then Z may be able to read secret messages intended for A or create documents allegedly signed by A. There is an additional problem with private keys that does not arise with public keys: If A can deliberately compromise D_A, then A has a case for disavowing previously signed messages.[20]

To protect against compromises (including deliberate disclosures) of private keys, compromises must be reported to the key server and new public keys registered. If this is done, public-key certificates and mass distribution of public keys protect messages encrypted for secrecy; any user obtaining a public key E_A is assured that the corresponding key D_A is still valid, and therefore a secret message M encrypted under E_A will not be disclosed. They do not, however, protect signatures.

To protect signatures we need a mechanism whereby a signature can be validated even after the signature key has been compromised. This requires binding the signature to the public key that was valid at the time the message was signed. This can be done with a signature certificate, which is like a public-key certificate with a signature inside.

Signature certificates. Let $X = D_A(\overline{M},I)$ be the signature computed by user A for message M, where $\overline{M} = h(M,I)$ is the digest of M as described earlier and shown in Figure 4. The signature certificate for X is given by

$$G := D_S(T,A,E_A,X) \ .$$

The receiver of the certificate G obtains T, A, E_A, and X by computing $E_S(G)$ and checks that the time stamp is current.

The time stamp in the signature certificate is essential to determine whether a message was signed before the signature key was compromised. This time stamp must be affixed by the key server. If A affixed the time stamp, A could intentionally affix an incorrect time. Moreover, if A's key is compromised, someone else could forge a message and affix a time when the key was valid. Instead, the key server affixes the time stamp to the signed message and signs the result,[21] thereby playing the role of a "notary public."[22]

The key server is not given access to A's private key D_A or any of the secret messages signed by A, and as noted earlier, the hashing function is one-way so that M cannot be determined from the digest $\overline{M}$. Thus, the server cannot disclose secret information or forge A's signature on a message but simply appends T, A, and E_A to whatever information is sent by A. The receiver of the certificate is responsible for validating A's signature. Note also that the signature certificate must be retained to resolve any disputes that may arise later.

Sending a signed secret message with certificates. Figure 5 gives a high-level protocol using public-key and signature certificates whereby user A can send a signed secret message to user B. The protocol is illustrated in Figure 6.

Compromise of server's private key. There is a potentially serious security problem with certificates if the server's signature key D_S is ever compromised. The compromiser, say Z, can then forge public-key certificates. The problem becomes extremely serious if Z knows a past (or present) signature key D_A belonging to some user A, because Z can then forge signature certificates as well. In particular, Z can take any message M, use D_A to compute the signature X, and then use D_S to create a valid

```
1. $A$ generates a random value $I$ and computes the following:
       $\overline{M} := h(M,I)$              "compute digest"
       $X := D_A(\overline{M},I)$            "sign"
   $A$ then sends the following messages to $S$:
       "Send public-key certificate for $B$."
       "Send signature certificate for $X$."

2. $S$ computes and returns the certificates to $A$:
       $P := D_S(T_1, B, E_B)$          "public-key certificate of $B$"
       $G := D_S(T_2, A, E_A, X)$       "signature certificate for $X$"

3. Using the public key of the server, $A$ computes
       $E_S(P) = (T_1, B, E_B)$
       $E_S(G) = (T_2, A, E_A, X)$
   and checks the time stamps $T_1$ and $T_2$. Using $E_B$, $A$ computes the
   ciphertext
       $C := E_B(M)$                    "encipher $M$."
   $A$ transmits the encrypted message $C$ and signature certificate $G$ to $B$.

4. $B$ computes
       $D_B(C) = M$                     "decipher $C$"
       $E_S(G) = (T_2, A, E_A, X)$
   and checks the time stamp $T_2$. Using $E_A$, $B$ computes
       $E_A(X) = (\overline{M}, I)$.
   Finally, $B$ validates $A$'s signature by checking the equality
       $h(M, I) = \overline{M}$.
```

Figure 5. Protocol for sending a signed secret message *M* from *A* to *B*.

signature certificate with time T when D_A was valid. In short, if Z has compromised both D_S and D_A for some user A, then Z can create messages that are legally binding to A. Moreover, Z can continue to do this even after the server's key has been changed.

Even if the probability of compromise is small, the consequences are so severe that if digital signatures are to be used at all, some mechanism for detecting and recovering from compromises should be installed.

An on-line certificate log

One method for protecting against forgeries and compromises of all keys, including the server's, is for the key server to keep a log or audit trail of the following events:

(1) Registration of public key, noted by entry of public-key certificate in the log.
(2) Registration of signature, noted by entry of signature certificate in the log. (Signatures are not legally binding unless they are logged.)
(3) Notification of key compromise.

The log is stored sequentially on a write-once device such as an optical disk so that records in the log cannot be altered and new records cannot be inserted in the middle. Extra copies of the log can be kept in physically separate locations to protect against destruction or loss. All information placed in the log is time-stamped by the server, and entries are in ascending order by time.

Only the server should be capable of writing into the log. This will be achieved if the server is physically isolated so that no process on the network can access the log without going through the server. It might also be achieved if the server and log are logically isolated in a separate virtual machine on a secure host.

None of the information in the log is secret, and anyone should be allowed to read it. The log is not encrypted. This prevents loss of the log in case the server's key is destroyed, allows a third party to audit the server or settle a dispute, and provides a recovery mechanism in case the server is subverted.

Recording events in the log. Every event in the log can be recorded both with and without the server's signature, with the unsigned portion to be used for retrieval of public keys and signatures.

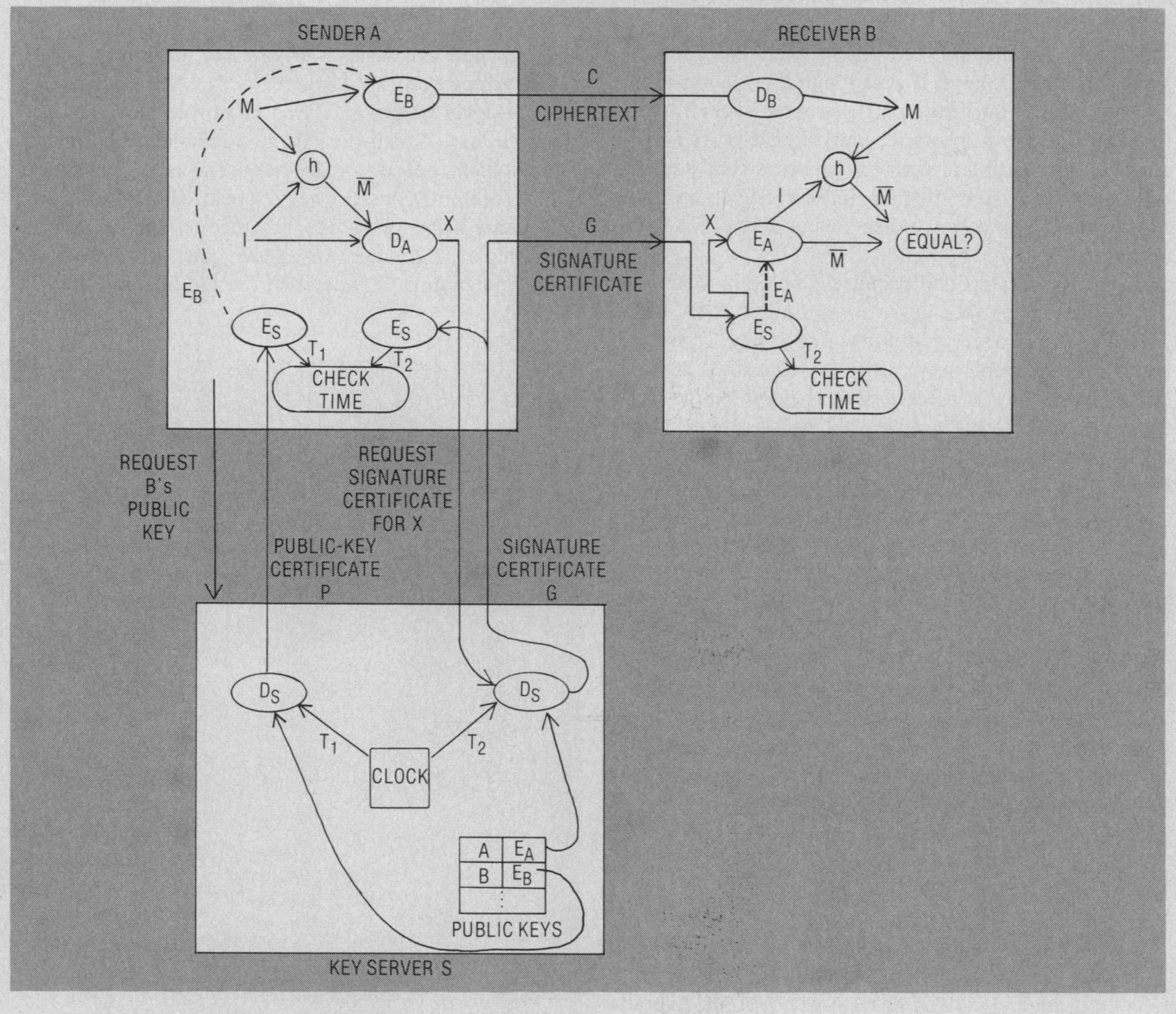

Figure 6. Sending a signed secret message with certificates.

Public-key registration. At any given time, each user has at most one valid public key, used for both secrecy and signatures; the procedure is easily modified to allow for separate key pairs. To change keys, the user simply registers a new public key. (If the reason for the key change is compromise of the previous key, this event is recorded first.) The same procedure is followed with the key server. The correct public key of a user A is thus given by the most recent public-key certificate $P = D_S(T,A,E_A)$ in the log (as long as this record is not followed by a "compromise" message for E_A). The log is initialized with a record containing the server's public key.

When A registers E_A, a copy of the public-key certificate P containing E_A is returned to A. A can check the validity of P using the server's public key. If another user B wishes E_A, then B can obtain P directly from A. Alternatively, B can request a current certificate from the server, in which case the server sends B a certificate for E_A with the current time. This certificate is not logged.

Signature registration. Signature certificates are automatically appended to the log when they are requested. In particular, when the server computes a signature certificate $G := D_S(T,A,E_A,X)$ for a signature X of A (as in step 2 of the protocol in Figure 5), the server appends G to the log.

Notification of key compromise. If A's private key D_A is compromised, A signs the message $M =$ "compromise" in the usual way (i.e., by hashing M and then applying D_A). With $XCOMP$ denoting the signature, M and $XCOMP$ are transmitted to the server, which appends the record $D_S(T,A,E_A,XCOMP)$ to the log. The purpose of this record is to limit A's liability on messages signed with D_A just before its compromise was detected and a new public key established. If a dispute arises over a signature X with certificate $D_S(T_1,A,E_A,X)$, and the log contains a subsequent "compromise" record for E_A with time stamp T_2 falling within some specified time interval dT from T_1, then A wins the case.

If the server's key is compromised, this event is also recorded in the log. Although this may not prevent false information from being appended to the log between the time of compromise and the time of detection (if the compromiser can obtain access to the log, which may not be possible), it limits the time stamps that can be placed in those entries to that time period. In particular, it will not be possible to forge a signature certificate for a previously compromised key D_A with a time stamp T when D_A was valid.

Table 1 shows a sample log recording keys for S, A, and B; signatures for A; and compromise of A's key. For clarity, time stamps are written in the form year:month:day:hour:minute:second. Note that compromise of A's key was reported the day after the signature X was registered. If dT exceeds one day, then A has a case for disavowing X.

Table 1.
Sample log.

EVENT	CERTIFICATE LOGGED
REGISTER KEY FOR S	(82:10:20:19:37:26. S. E_S)
REGISTER KEY FOR A	D_S(82:10:21:14:25:05. A. E_A)
REGISTER KEY FOR B	D_S(82:10:21:15:36:52. B. E_B)
REGISTER SIGNATURE FOR A	D_S(82:11:15:17:02:41. A. E_A. X_1)
REGISTER COMPROMISE FOR A	D_S(82:11:16:10:19:20. A. E_A. $XCOMP$)
REGISTER KEY FOR A	D_S(82:11:16:10:20:47. A. E_A')

Authentication. Registration of public keys and signatures raises an important security question: How can the server be sure of any user's identity? If the users' hosts are secure and securely linked to the server, then the server can rely on information provided by the hosts (as long as users are reliably authenticated by the hosts). But if the hosts are insecure, the server cannot rely on any information they transmit. Although information transmitted from a user's terminal can be encrypted with the server's key, authenticating the identity of the user at that terminal is difficult, especially if keys (or passwords) have been compromised. However, masqueraders can be detected if public keys and signatures placed in the log are publicly broadcast.

Key compromises. The log does not eliminate the problems that arise from key compromises, but it does confine them substantially. More important, it allows for their detection. The following discussion of cases in which a penetrator compromises a user's key, the server's key, or both is summarized in Table 2.

Table 2.
What a penetrator can do with successful compromise of private keys.

COUNTER-MEASURE	PRIVATE KEYS COMPROMISED			
	NONE	A's KEY	SERVER'S KEY	BOTH KEYS
NOTHING	SEND FAKE PUBLIC KEYS	READ A's MAIL. FORGE A's SIGNATURE	N/A	N/A
CERTIFICATES WITHOUT LOG	- -	READ A's MAIL	FORGE PUBLIC-KEY CERTIFICATE	FORGE A's SIGNATURE & SIGNATURE CERTIFICATE
CERTIFICATES WITH LOG	-	READ A's MAIL	-	-

User's key only. Suppose that Z has compromised A's current private key D_A but A has not yet detected the compromise. Then Z can forge A's signature on any message M. For such a signature to be legally binding, however, Z must get a signature certificate from the key server. As long as Z can successfully masquerade as A to the server, the signature certificate will be recorded in the log. If the server notifies users when records are logged on their behalf (as mail programs notify users of incoming mail), or if the server publicly broadcasts new entries in the log, then A can detect the misuse of D_A, report the compromise, and register a new key. Of course, Z could also report the compromise of D_A and attempt to install a new key for A that would be unknown to A, but this is detectable as well.

Server's key. Suppose that Z has compromised the server's private key D_S but the server has not yet detected the compromise. Then Z can create fake public-key certificates. But unless Z can write into the log, the public keys in these certificates will not be properly registered. In the unlikely event that Z is able to write into the log, the server can detect the fake certificates by comparing the last record in the log with the last one it has written. The server can then enter a "compromise" record into the log, install and broadcast a new public key, and notify the users identified in the certificates.

Server's and user's keys. If Z has also compromised the private key D_A of a user A, then Z can forge A's signature on arbitrary messages and subsequently forge the server's signature on a signature certificate. Again, however, for such a certificate to be valid, it must be written into the log, which makes the compromise detectable. Note also that once A changes D_A and E_A, Z cannot forge D_A on a message even by time-stamping the message with a time T when both D_A and D_S were valid, because Z would have to insert the certificate into the log in its proper time sequence, which is physically impossible.

We might imagine a worst-case scenario in which a penetrator subverts the key server. Even then the possible damage is confined because the penetrator will not be able to create legally binding certificates with past time stamps.

Conclusions

It would be unwise to use a public-key cryptosystem for secrecy and legally binding signatures without installing mechanisms for detecting and recovering from key compromises. An on-line log provides such a mechanism. Although the log does not prevent forgeries of messages or certificates, it does permit their detection and confine the damage. Security can be further enhanced by off-line publication of public keys and signatures.

Keeping the log of signature certificates has other advantages as well. If a user loses a signature certificate, it can be recovered from the log. The log also provides a convenient mechanism for handling contracts signed by more than one user. If a contract M requires n signatures from users $A_1, \ldots, A_n$, the contract becomes legally binding only after all n users have registered signatures on M in the log. To prevent some user from withholding a signature for an indefinite period after the others have signed, we can require that all signature certificates in the log have time stamps falling within a predetermined interval. (Contracts can also be signed without recourse to a key server.[23,24])

The key server is a potential bottleneck in the network if there are many requests for public keys or the creation of new certificates. This problem can be partially mitigated by regularly downloading all or part of the public-key directory to the users' workstations (or host systems, if secure).[25] The problem can also be alleviated by distributing the server over the network so that all requests are not addressed to the same site. Each server component might keep a complete list of all public keys but not necessarily a complete log of all certificates. Encryption can be used to securely connect the components of the server. Distributing the server has the additional benefit of providing redundancy in case of failure.

Keeping a log or audit trail should be an important part of any security program. Many systems already keep a log to detect unauthorized activity on the system, and its existence alone can be a powerful deterrent to a would-be penetrator. High-capacity write-once devices such as optical disks are ideally suited for security logs because the information they contain cannot be altered. ■

Acknowledgments

I am grateful to Peter Denning for helpful discussions and comments, and to the referees for helpful suggestions. This research was supported in part by NSF grant MCS80-15484.

Summary of notation

C	ciphertext (encrypted message)
D_A	private (signature) key or transformation of user A—used for decryption and signing
E_A	public key or transformation of user A—used for encryption and signature validation
D_S	private key or transformation of key server S
E_S	public key or transformation of key server S
D_K	conventional decryption transformation with key K
E_K	conventional encryption transformation with key K
G	$= D_S(T, A, E_A, X)$; signature certificate for A
h	hashing function
I	random seed for initializing h
K	key for conventional (one-key) encryption algorithm
M	plaintext message
$\overline{M}$	$= h(M, I)$; digest of M
P	$= D_S(T, A, E_A)$; public-key certificate for A
T	time stamp
X	$= D_A(\overline{M}, I)$; signature of A on M

References

1. W. Diffie and M. Hellman, ''New Directions in Cryptography,'' *IEEE Trans. Information Theory,* Vol. IT-22, No. 6, Nov. 1976, pp. 644-654.
2. D. E. Denning, *Cryptography and Data Security,* Addison-Wesley, Reading, Mass., 1982.
3. G. J. Simmons and D. A. Holdridge, ''Forward Search as a Cryptanalytic Tool Against a Public Key Privacy Channel,'' *Proc. 1982 IEEE Symp. Security and Privacy,* Apr. 1982.
4. R. L. Rivest, A. Shamir, and L. Adleman, ''A Method for Obtaining Digital Signatures and Public-Key Cryptosystems,'' *Comm. ACM,* Vol. 21, No. 2, Feb. 1978, pp. 120-126.
5. G. I. Davida, ''Chosen Signature Cryptanalysis of the RSA (MIT) Public Key Cryptosystem,'' TR-CS-82-2, Department of Electrical Engineering and Computer Science, University of Wisconsin, Milwaukee, Oct. 1982.
6. R. C. Merkle and M. E. Hellman, ''Hiding Information and Signatures in Trapdoor Knapsacks,'' *IEEE Trans. Information Theory,* Vol. IT-24, No. 5, Sept. 1978, pp. 525-530.
7. A. Shamir and R. E. Zippel, ''On the Security of the Merkle-Hellman Cryptographic Scheme,'' *IEEE Trans. Information Theory,* Vol. IT-26, No. 3, May 1980, pp. 339-340.
8. A. Lempel, ''Cryptology in Transition,'' *Computing Surveys,* Vol. 11, No. 4, Dec. 1979, pp. 285-303.
9. A. Shamir, ''A Fast Signature Scheme,'' MIT/LCS/TM-107, Massachusetts Institute of Technology Laboratory for Computer Science, Cambridge, Mass., July 1978.
10. R. McEliece, ''A Public-Key Cryptosystem Based on Algebraic Coding Theory,'' DSN progress report 42-44, Jet Propulsion Laboratory, California Institute of Technology, Pasadena, Jan.-Feb. 1978.
11. R. Janardan, ''A Public-Key Cryptosystem Based on the Matrix Cover Problem,'' *Advances in Cryptography,* Proc. Crypto 82.
12. A. Shamir, ''A Polynomial Time Algorithm for Breaking the Basic Merkle-Hellman Cryptosystem,'' *Proc. 23rd Ann. Symp. Foundations of Computer Science,* Nov. 1982, pp. 145-152.
13. A. Shamir, ''On the Cryptocomplexity of Knapsack Systems,'' *Proc. 11th Ann. ACM Symp. Theory of Computing,* May 1979, pp. 118-129.
14. ''Data Encryption Standard,'' FIPS Pub. 46, National Bureau of Standards, Washington, D.C., Jan. 1977.
15. D. W. Davies and W. L. Price, ''The Application of Digital Signatures Based on Public Key Cryptosystems,'' NPL report DNACS 39/80, National Physical Laboratory, Teddington, Middlesex, England, Dec. 1980.
16. D. E. Denning, ''Signature Protocols for RSA and Other Public-Key Cryptosystems,'' Computer Sciences Department, Purdue University, Nov. 1982.
17. R. M. Needham and M. Schroeder, ''Using Encryption for Authentication in Large Networks of Computers,'' *Comm. ACM,* Vol. 21, No. 12, Dec. 1978, pp. 993-999.
18. L. M. Kohnfelder, ''A Method for Certification,'' Massachusetts Institute of Technology Laboratory for Computer Science, Cambridge, Mass., May 1978.
19. D. E. Denning and G. M. Sacco, ''Timestamps in Key Distribution Protocols,'' *Comm. ACM,* Vol. 24, No. 8, Aug. 1981, pp. 533-536.
20. J. Saltzer, ''On Digital Signatures,'' *Operating Systems Review,* Vol. 12, No. 2, Apr. 1978, pp. 12-14.
21. R. C. Merkle, ''Protocols for Public Key Cryptosystems,'' *Proc. 1980 IEEE Symp. Security and Privacy,* Apr. 1980, pp. 122-133.
22. G. J. Popek and C. S. Kline, ''Encryption and Secure Computer Networks,'' *Computing Surveys,* Vol. 11, No. 4, Dec. 1979, pp. 331-356.
23. M. Blum, ''How to Exchange (Secret) Keys,'' UCB/ERL M81/90, Department of Electrical Engineering and Computer Science, University of California, Berkeley, Mar. 1982.
24. M. O. Rabin, ''Transaction Protocols by Beacons,'' Department of Mathematics, The Hebrew University of Jerusalem, 1981.
25. B. P. Schanning, S. A. Powers, and J. Kowalchuk, ''Memo: Privacy and Authentication for the Automated Office,'' *Proc. Fifth Conf. Local Computer Networks,* Oct. 1980, pp. 21-30.

Dorothy E. Denning is an associate professor of computer sciences at Purdue University. Her primary research area is cryptography and data security. She has been an assistant research mathematician at the University of Michigan Radio Astronomy Observatory and a systems programmer and instructor at the University of Rochester. Denning is chairman of the ACM Special Interest Group on Operating Systems. She is on the editorial board of *ACM Transactions on Computer Systems* and the advisory board of *Computer Security Journal.* She has served as an IEEE distinguished visitor and is the author of *Cryptography and Data Security,* Addison-Wesley, 1982.

Denning received the BA and MA degrees in mathematics from the University of Michigan and the PhD in computer science from Purdue University.