Technical Note
Operating Systems

R. Stockton Gaines*
Editor

Timestamps in Key Distribution Protocols

Dorothy E. Denning and Giovanni Maria Sacco
Purdue University

The distribution of keys in a computer network using single key or public key encryption is discussed. We consider the possibility that communication keys may be compromised, and show that key distribution protocols with timestamps prevent replays of compromised keys. The timestamps have the additional benefit of replacing a two-step handshake.

Key Words and Phrases: encryption, encryption keys, key distribution, communications, security, timestamps.
CR Categories: 3.81, 4.39

I. Introduction

Secure communication between two users on a computer network is possible using either single key (conventional) encryption or public key encryption. In both systems, key distribution protocols are needed so the users can acquire keys to establish a secure channel. In single key systems, the users must acquire a shared communication key; in public-key systems [2], the users must acquire each others' public keys.

Needham and Schroeder propose key distribution protocols for both single key and public key systems based on a centralized key distribution facility called an Authentication Server (AS) [6]. Their protocol for a single key system assumes that the AS is responsible for generating and distributing all communication keys, and that each user has registered a private (secret) key with the AS. The AS uses the private keys to protect (by encryption) the communication keys transmitted to the users. If communication keys and private keys are never compromised (as Needham and Schroeder assume), the protocol is secure (i.e., can be used to establish a secure channel).

We will show that the protocol is not secure when communication keys are compromised, and propose a solution using timestamps. Although the likelihood of such a compromise may be small, the timestamps are useful for another reason: they can replace a two-step handshake designed to prevent replays of (noncompromised) keys.

We also show that timestamps can replace the handshake in the Needham and Schroeder protocol for public key systems. Here the AS is responsible for distributing users' public keys; it does not require access to their private keys. Because there are no secret communication keys, their compromise is not an issue. However, timestamps are valuable in public key systems to ensure the integrity of keys.

Public key systems also provide an alternate method of exchanging communication keys for single-key data encryption. As before, timestamps protect against replays of previously compromised communication keys.

No protocol is secure if users' private keys are compromised. We conclude with a brief discussion of the threats to private keys in both types of systems.

II. Single Key Systems

A. Distribution of Communication Keys

Needham and Schroeder assume that each user A has a private (secret) key K_A which is known only to A and AS. If two users wish secure communication, one of them obtains a secret communication key CK from AS and gives a copy to the other. If a new key is obtained for each conversation, a user need not keep a list of secret communication keys for all his correspondents.

The key distribution protocol is as follows. Let $\{x\}^K$ denote the message x enciphered under key K. For a user A to acquire a key CK to share with another user B, these steps are taken:

$$A \rightarrow \text{AS}: A, B, I_A \tag{1}$$

$$\text{AS} \rightarrow A: \{I_A, B, CK, Y\}^{K_A} \tag{2}$$

where I_A is an identifier chosen by A and used only once, and $Y = \{CK, A\}^{K_B}$. Because I_A is returned by AS, enciphered under A's secret key, A can be sure that the response (2) is not a replay of a previous response. A then sends to B the message Y, which contains a copy of CK enciphered under B's private key:

$$A \rightarrow B: Y. \tag{3}$$

B. The Handshake

After step (3), A can be sure that the key CK is safe to use. However, B cannot be sure that the enciphered message Y and subsequent messages supposedly sent from A are not replays of previous messages. To protect against replays, a *handshake* between B and A follows:

* Former editor of Operating Systems department, of which Anita K. Jones is the current editor.

This research was supported in part by NSF Grant MCS77-04835.

Authors' present address: Computer Sciences Dept., Purdue Univ., W. Lafayette, IN 47907.

$$B \rightarrow A: \{I_B\}^{CK} \quad (4)$$

$$A \rightarrow B: \{f(I_B)\}^{CK} \quad (5)$$

where I_B is an identifier chosen by B. A signals his intention to use CK by returning an agreed function f of I_B; f could be something simple like $f(I) = I - 1$. The complete sequence of steps (1)–(5) establishes a secure channel between A and B as long as previous communication keys and private keys have not been compromised.

C. Compromises of Communication Keys

If the encryption algorithms are strong and keys random, it is unlikely that communication keys will be compromised by cryptanalysis. We are more concerned with the communication key's direct exposure due to negligence or a design flaw in the system, i.e., an intruder may be able to break into the AS or into A's or B's computer and steal a key.

Let us suppose that a third party C has intercepted and recorded all the messages between A and B in steps (3)–(5), and that C has obtained a copy of the communication key CK. C can then later trick B into using the CK as follows: First C replays the message Y to B:

$$C \rightarrow B: \{CK, A\}^{K_B}.$$

Thinking that A has initiated a new conversation, B requests a handshake from A:

$$B \rightarrow A: \{I'_B\}^{CK}.$$

C intercepts the message, deciphers it, and impersonates A's response:

$$A \rightarrow B: \{f(I'_B)\}^{CK}.$$

Thereafter, C can send bogus messages to B that appear to be from A, intercepting and deciphering B's replies. Note, however, that A can still communicate safely with B by initiating a new conversation (assuming A's attempts are not blocked by C), and that B can communicate safely with other users.

Of course, C cannot impersonate A's response to B in the handshake without knowing the handshake function f. But the problem of securing secret handshake functions is as difficult as the problem of securing communication keys. Users must either privately exchange and remember permanent handshake functions, or they must obtain them from the AS. If the former approach is taken, then users may as well exchange directly their communication keys and dispense with the handshake functions. If the latter approach is taken, the problem of distributing handshake functions is identical to the problem of distributing keys.

D. Timestamps

If private keys are secure, we can solve the above problem and eliminate the handshake by adding a *timestamp* T to steps (2) and (3). The new protocol is thus:

$$A \rightarrow \text{AS}: A, B \quad (1)$$

$$\text{AS} \rightarrow A: \{B, CK, T, Y\}^{K_A} \quad (2)$$

$$A \rightarrow B: Y \quad (3)$$

where $Y = \{A, CK, T\}^{K_B}$.

A and B can verify that their messages are not replays by checking that

$$|\text{Clock} - T| < \Delta t1 + \Delta t2$$

where Clock gives the local time, $\Delta t1$ is an interval representing the normal discrepancy between the server's clock and the local clock, and $\Delta t2$ is an interval representing the expected network delay time. If each node sets its clock manually by reference to a standard source, a value of about one or two minutes for $\Delta t1$ would suffice. As long as $\Delta t1 + \Delta t2$ is less than the interval since the last use of the protocol, this method will protect against replays. Since the timestamp T is enciphered under the private keys K_A and K_B, impersonation of the AS is impossible.

Needham and Schroeder suggested caching old communication keys to reduce the number of steps required to initiate a conversation. Caching is not recommended here, because the timestamps would become obsolete.

Needham and Schroeder suggested timestamps as a way to validate the time integrity of one way communications such as computer mail. They rejected timestamps in their key distribution protocol because there might not be a network-wide reliable source of time. As we have argued, timestamps can be used reliably even if the settings of local clocks are not completely reliable. However, this approach may not be suitable for all types of connections—e.g., connections to terminals without local day–date clocks.

III. Public Key Systems

A. Distribution of Public Keys

Timestamps can also be added to Needham's and Schroeder's protocols for public key systems. Here the AS stores and distributes users' public keys; it does not necessarily have access to their private (secret) keys. As in single key systems, the handshake is no longer needed. Furthermore, if the AS distributes public keys inside "certificates" [3], the entire protocol can be reduced to three steps (from the original seven). Letting P_A and S_A denote A's public key and secret (signature) key, respectively, the complete protocol is

$$A \rightarrow \text{AS}: A, B \quad (1)$$

$$\text{AS} \rightarrow A: CA, CB \quad (2)$$

$$A \rightarrow B: CA, CB \quad (3)$$

where

$$CA = \{A, P_A, T\}^{S_{AS}} \text{ and } CB = \{B, P_B, T\}^{S_{AS}}$$

are certificates containing, respectively, A's and B's pub-

lic keys. The certificates are signed by the AS using its secret key S_{AS} to prevent forgery. Both A and B are given copies of their own certificates, so they can validate their own public keys.

Because public keys are readily available to all users, their exposure is not an issue. However, their integrity is essential, so they still require a high level of protection. The use of timestamps as well as signed certificates helps protect against replays of old keys or substitution of bogus keys. Popek and Kline included both in their public key distribution protocols [7].

B. Distribution of Communication Keys

Public key distribution protocols can also be used to distribute communication keys for single-key data encryption [2, 5]. The above protocol becomes

$$A \rightarrow \text{AS: } A, B \quad (1)$$

$$\text{AS} \rightarrow A\text{: } CA, CB \quad (2)$$

$$A \rightarrow B\text{: } CA, CB, \{\{CK, T\}^{S_A}\}^{P_B} \quad (3)$$

The key CK is then used for encrypting messages transmitted between A and B. Because the CK is chosen and encrypted by A, there is no risk of its exposure by the AS; however, it is still vulnerable to compromise in A's or B's computer. Here again, timestamps protect against replays of compromised keys.

IV. Compromises of Private Keys

The exposure of users' private keys poses a much more serious threat which is not eliminated by the timestamp protocols. In a single key system, private keys are used by the AS to encipher all communication keys transmitted to the user. (Private keys may be used for other purposes as well, but this is not pertinent to the present discussion.) If an intruder compromises a user's private key, he can decipher any communication key sent to the user; he can also impersonate the AS and trick the user into accepting and using a false communication key. All information transmitted to or from the user is thus vulnerable to attack. Indeed, the intruder can even impersonate the user.

There are several methods by which a user's private key may be compromised:

(a) An intruder may record messages enciphered with the key and compromise it by cryptanalysis.
(b) The AS may be untrustworthy and leak the key.
(c) An intruder may exploit a design flaw in the system and steal the key.

As with communication keys, the first method is unlikely to succeed if the encryption algorithm is strong and keys are random—especially if the private keys are used only to encipher random communication keys. The second method is also unlikely to succeed if the AS is verified for correctness and security. We are primarily concerned about the third method of attack. An intruder, for example, might be able to break into the AS, or discover the user's password and log directly onto his computer. In this sense, private keys are more vulnerable than communication keys, because the AS must keep a list of all private keys, and they have a longer lifetime than communication keys. Although the keys can be encrypted under a master key at the AS and stored in special hardware (e.g., see [4]), they are never absolutely safe. Because of these vulnerabilities, a mechanism is needed to generate and distribute new private keys.

Exposure of private keys in public key systems poses an equally serious threat. There are two possibilities: exposure of user's private keys, and exposure of the private key used by the AS to sign certificates.

If a user's private key is exposed, all messages sent to him enciphered under his corresponding public key may be compromised, and his signature may be forged on other messages. Moreover, he may be able to disavow his signature on a previously signed message, claiming that his private key was lost or stolen before the message was signed [8]. (Merkle solves the problems with digital signatures by having a timekeeper affix a timestamp and its own signature to a signed message [5].) If compromises are reported to the AS and public keys are obtained from the AS immediately prior to use, timestamps are useful for validating the time integrity of keys.

Because the AS does not have access to users' private keys, these keys cannot be leaked or stolen from the AS, eliminating one of the threats of single key systems. Moreover, users' private keys need not be stored in the system at all; they could be recorded in ROM or magnetic stripe cards, and inserted into a special reader attached to the users terminal, greatly reducing the threat of any type of attack on the system[1]. The only remaining threats are cryptanalysis and intentional leaks on the part of the user.

There is still the problem of protecting the AS's private signature key, which may be leaked or stolen from the AS. Again, this key need not be stored inside an accessible location of the system. Merkle has suggested a method for producing certificates that does not require a digital signature from the AS[5].

V. Conclusions

We have shown that key distribution protocols with timestamps prevent replays of previously compromised communication keys. Even if the likelihood of compromise is small, the timestamps have the benefit of replacing a two-step handshake. We recommended their inclusion in all key distribution protocols.

Acknowledgments. We wish to thank P. Denning, A. Jones, S. Kent, C. Kline, G. Popek, and the referees for many helpful suggestions, and J. Gray, R. Needham, and M. Schroeder for helpful discussions.

Received 11/79; revised 1/81; accepted 3/81

References

1. Denning, D. E. Secure personal computing in an insecure network. *Comm. ACM* 22, 8 (Aug. 1979) 476–482.
2. Diffie, W., and Hellman, M., New directions in cryptography. *IEEE Trans. on Info. Theory IT-22*, 6 (Nov. 1976) 644–654.
3. Konfelder, L. M., A method for certification. Lab. for Computer Science, MIT, Cambridge, Mass. (May 1978).
4. Matyas, S. M., and Meyer, C. H. Generation, distribution, and installation of cryptographic keys. *IBM Syst. J. 17*, 2 (1978) 126–137.
5. Merkle, R. C. Protocols for public key cryptosystems. *Proc. 1980 Symp. on Security and Privacy*, IEEE Catalog No. 80 CH 1522-2 (April 1980) 122–133.
6. Needham, R. M., and Schroeder, M. Using encryption for authentication in large networks of computers. *Comm. ACM 21*, 12 (Dec. 1978) 993–999.
7. Popek, G. J., and Kline, C. S. Encryption and secure computer networks. *Comput. Surv. 11*, 4 (Dec. 1979) 331–356.
8. Saltzer, J. On digital signatures *Oper. Syst. Rev. 12*, 2 (April 1978) 12–14.

technical correspondence

File Updating–Still Once More

☐ I read with great interest Barry Dwyer's article on how to update a master file in the January issue of *Communications* [1]. My interest was based on some work I have done in the past. Although I have not been able to see all the referenced literature (I do not have available McCracken's [2] or Jackson's [3] work), Dijkstra's [4] example stresses that the key to a neat solution is the buildup of "the merged sequence" of all the different records with the same key. The processing of the sequence, once already built, depends on the specific case: types of records, their origin and meaning, and the answer to special cases and sequences. You can think of a record in the old file as equivalent to an addition record in the transactions file.

So, apart from the required precedence for those records, the total "merged sequence" is the basis for the answer: how to build it and how to process it.

The work I referred to initially, "Notas sobre la programación estructurada y la taréa cotidiana: Actualización de archivos" [5], (which I assume almost nobody has read abroad), had a didactic objective—to introduce Structured Programming in everyday activities. In it, I showed how a problem which was faced every day and which usually led to intricate answers could be seen in fresh new ways and be neatly solved. I also gave importance to the synchronization of the (sequential) files involved but considered the problem from different viewpoints: the old file, the transactions file, or even the updated file. Any of them could be privileged.

```
PROCEDURE DIVISION.
UPDATE-SEQUENTIAL-FILES.
OPEN INPUT OLD-FILE, TRANSACTION-FILE,
     OUTPUT MASTER-FILE.

PERFORM READ-OLD-RECORD.
PERFORM READ-A-TRANSACTION.
PERFORM CLEAR-MASTER-RECORD.
PERFORM CHOOSE-NEXT-FILE-KEY
PERFORM PROCESS-ONE-FILE-KEY
     UNTIL CURRENT-KEY=SENTINEL.
CLOSE OLD-FILE, TRANSACTION-FILE, MASTER-FILE.
STOP RUN.

PROCESS-ONE-FILE-KEY.
     IF OLD-KEY=CURRENT-KEY
                    PERFORM INITIAL-STATUS
                    PERFORM READ-OLD-RECORD
        ELSE IF TRANSACTION-KEY=CURRENT-KEY
                    PERFORM APPLY-TRANSACTION-TO-MASTER
                    PERFORM READ-A-TRANSACTION
        ELSE IF MASTER-KEY=CURRENT-KEY
                    WRITE MASTER
                    PERFORM CLEAR-MASTER-RECORD.
     PERFORM CHOOSE-NEXT-FILE-KEY

INITIAL-STATUS.
     MOVE OLD TO MASTER
     MOVE CURRENT-KEY TO MASTER-KEY.

CLEAR-MASTER-RECORD
     MOVE ZEROS TO MASTER
     MOVE SENTINEL TO MASTER-KEY.

READ-OLD-RECORD.
     READ OLD-FILE
          AT END MOVE SENTINEL TO TRANSACTION-KEY.

READ-A-TRANSACTION.
     READ TRANSACTION-FILE
          AT END MOVE SENTINEL TO TRANSACTION-KEY.

CHOOSE-NEXT-FILE-KEY.
     IF TRANSACTION-KEY<OLD-KEY MOVE TRANSACTION-KEY TO CUR-
        RENT-KEY
          ELSE MOVE OLD-KEY TO CURRENT-KEY.
     IF MASTER-KEY<CURRENT-KEY MOVE MASTER-KEY TO CURRENT-KEY
```