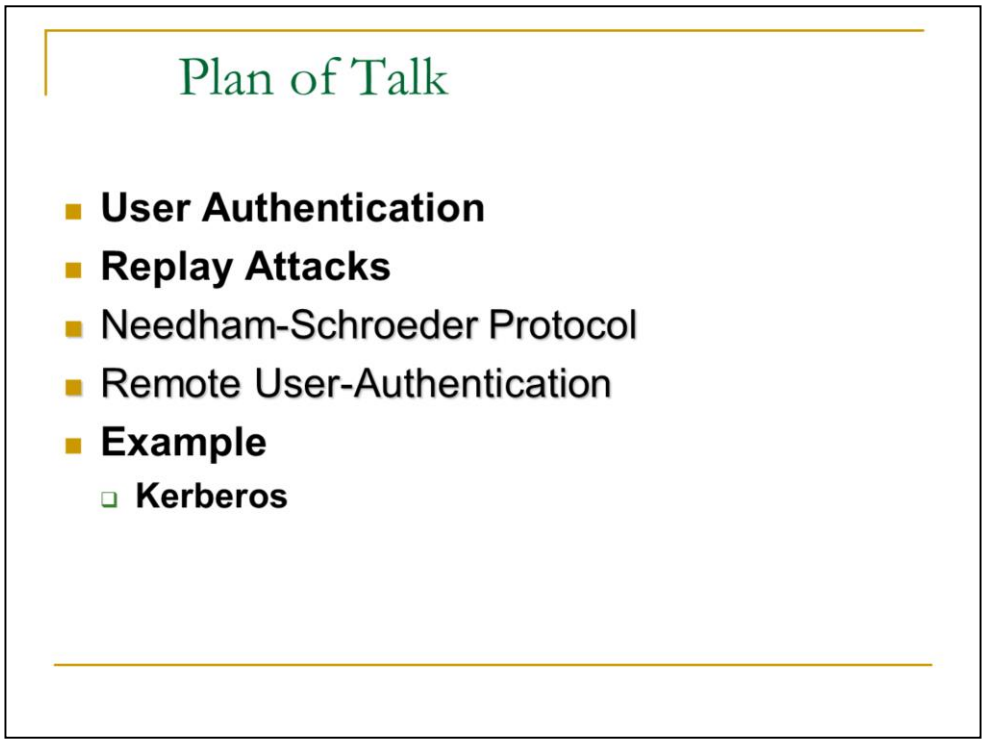
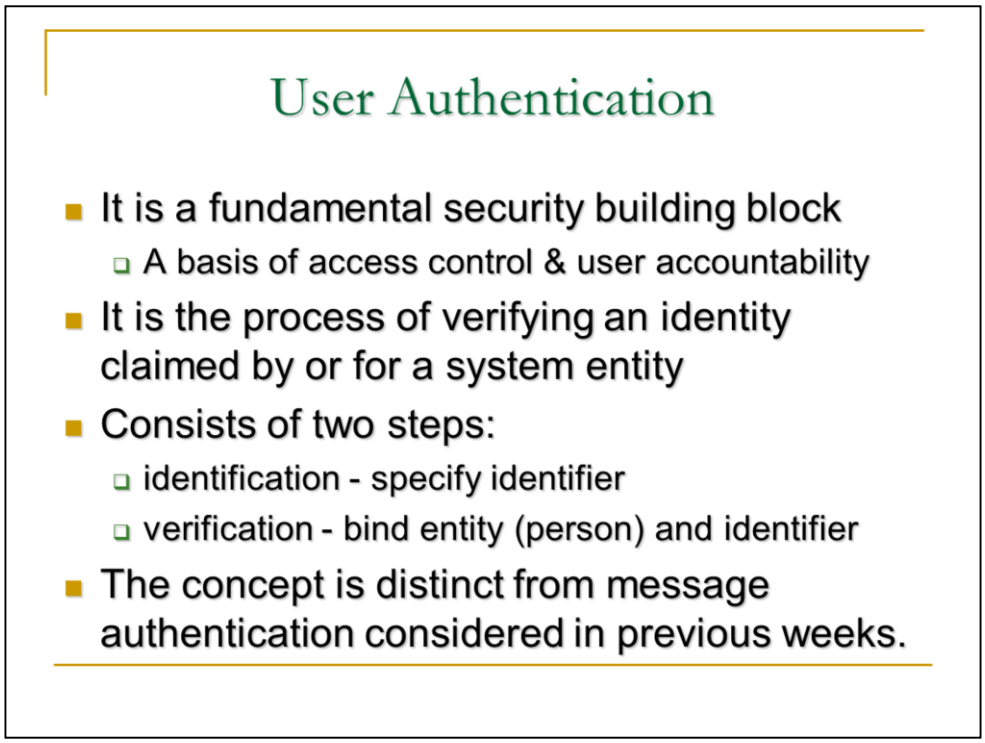


This lecture covers some important topics from Chapter 15: User Authentication.

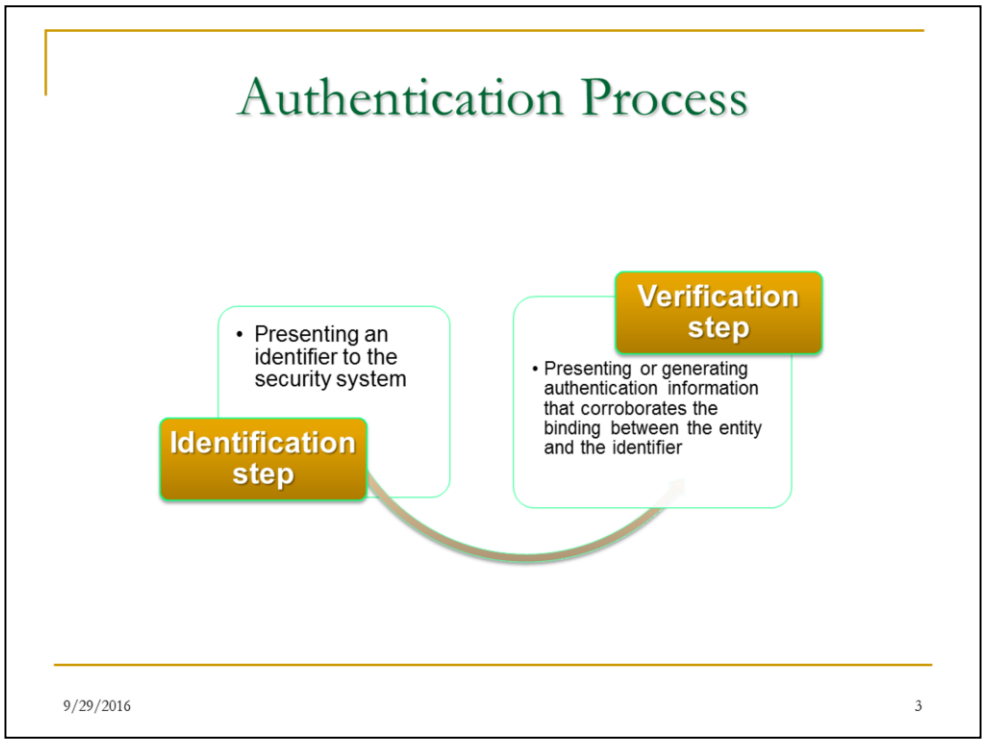


This chapter examines some of the authentication functions that have been developed to support network-based use authentication. In most computer security contexts, user authentication is the fundamental building block and the primary line of defense. User authentication is the basis for most types of access control and for user accountability. RFC 2828 defines user authentication as the process of verifying an identity claimed by or for a system entity. An authentication process consists of two steps:

•**Identification step:** Presenting an identifier to the security system. (Identifiers should be assigned carefully, because authenticated identities are the basis for other security services, such as access control service.)

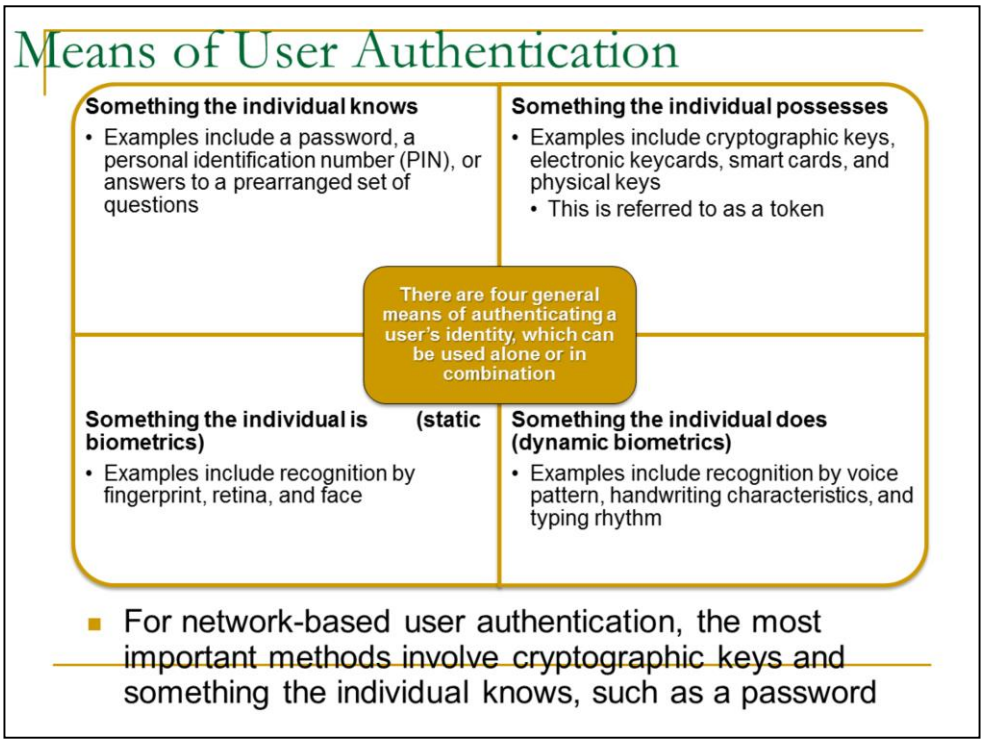
•**Verification step:** Presenting or generating authentication information that corroborates the binding between the entity and the identifier."

In essence, identification is the means by which a user provides a claimed identity to the system; user authentication is the means of establishing the validity of the claim. Note that user authentication is distinct from message authentication.



For example, user Alice Toklas could have the user identifier ABTOKLAS. This information needs to be stored on any server or computer system that Alice wishes to use and could be known to system administrators and other users. A typical item of authentication information associated with this user ID is a password, which is kept secret (known only to Alice and to the system). If no one is able to obtain or guess Alice's password, then the combination of Alice's user ID and password enables administrators to set up Alice's access permissions and audit her activity. Because Alice's ID is not secret, system users can send her e-mail, but because her password is secret, no one can pretend to be Alice.

In essence, identification is the means by which a user provides a claimed identity to the system; user authentication is the means of establishing the validity of the claim. Note that user authentication is distinct from message authentication. As defined in Chapter 12, message authentication is a procedure that allows communicating parties to verify that the contents of a received message have not been altered and that the source is authentic. This chapter is concerned solely with user authentication.



There are four general means of authenticating a user's identity, which can be

used alone or in combination:

• Something the individual knows: Examples include a password, a personal

identification number (PIN), or answers to a prearranged set of questions.

• Something the individual possesses: Examples include cryptographic keys,

electronic keycards, smart cards, and physical keys. This type of authenticator

is referred to as a token .

• Something the individual is (static biometrics): Examples include recognition

by fingerprint, retina, and face.

• Something the individual does (dynamic biometrics): Examples include recognition

by voice pattern, handwriting characteristics, and typing rhythm.

All of these methods, properly implemented and used, can provide secure

user authentication. However, each method has problems. An adversary may be

able to guess or steal a password. Similarly, an adversary may be able to forge or

steal a token. A user may forget a password or lose a token. Furthermore, there is a

significant administrative overhead for managing password and token information

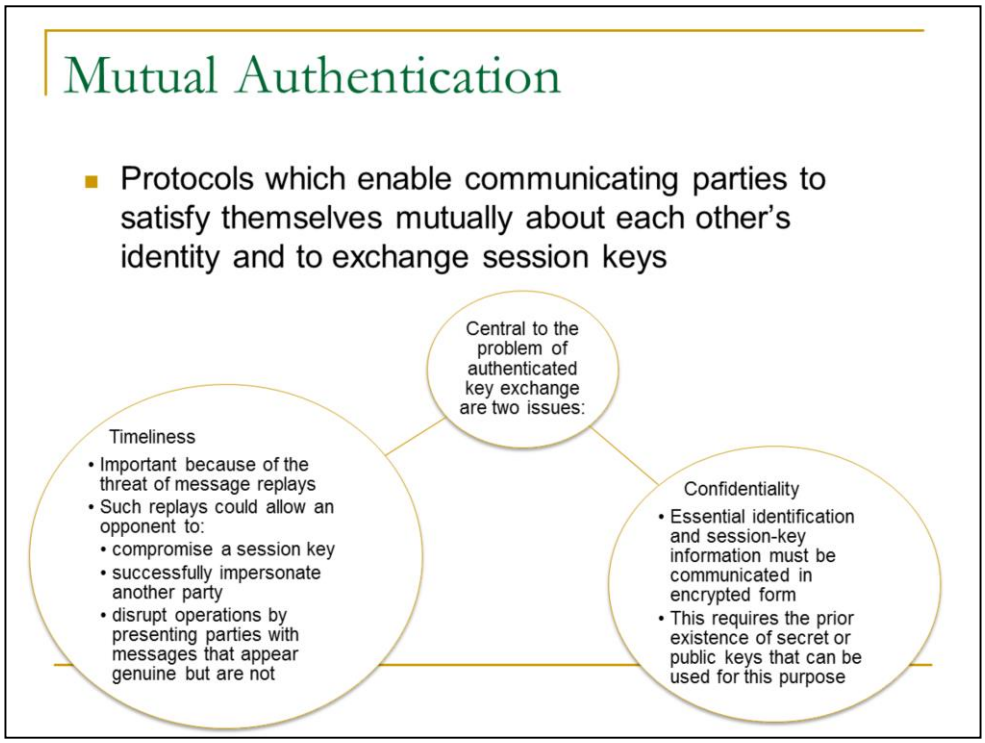
on systems and securing such information on systems. With respect to biometric authenticators,

there are a variety of problems, including dealing with false positives

and false negatives, user acceptance, cost, and convenience. For network-based user

authentication, the most important methods involve cryptographic keys and something

the individual knows, such as a password.



An important application area is that of mutual authentication protocols. Such

protocols enable communicating parties to satisfy themselves mutually about each

other's identity and to exchange session keys. This topic was examined in Chapter 14.

There, the focus was key distribution. We return to this topic here to consider the

wider implications of authentication.

Central to the problem of authenticated key exchange are two issues: confidentiality

and timeliness. To prevent masquerade and to prevent compromise of session

keys, essential identification and session-key information must be communicated in encrypted

form. This requires the prior existence of secret or public keys that can be used

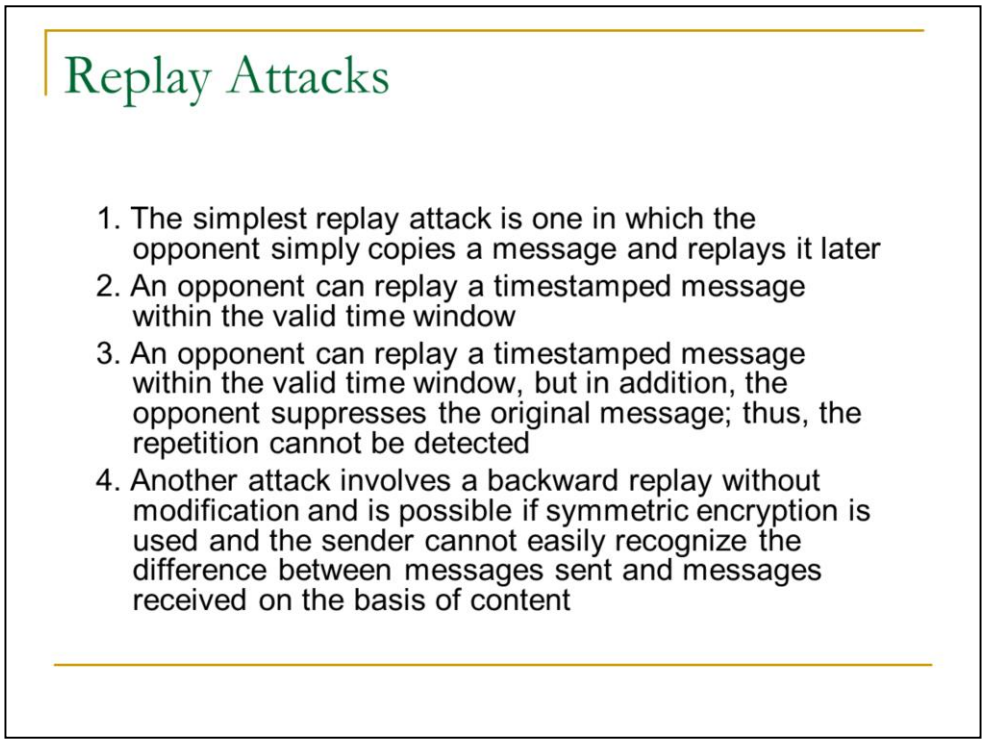
for this purpose. The second issue, timeliness, is important because of

the threat of message

replays. Such replays, at worst, could allow an opponent to compromise a session

key or successfully impersonate another party. At minimum, a successful replay can

disrupt operations by presenting parties with messages that appear genuine but are not.



[GONG93] lists the following examples of replay attacks:

1. The simplest replay attack is one in which the opponent simply copies a
message and replays it later.

2. An opponent can replay a timestamped message within the valid time window. If both the original and the replay arrive within then time window, this incident can be logged.

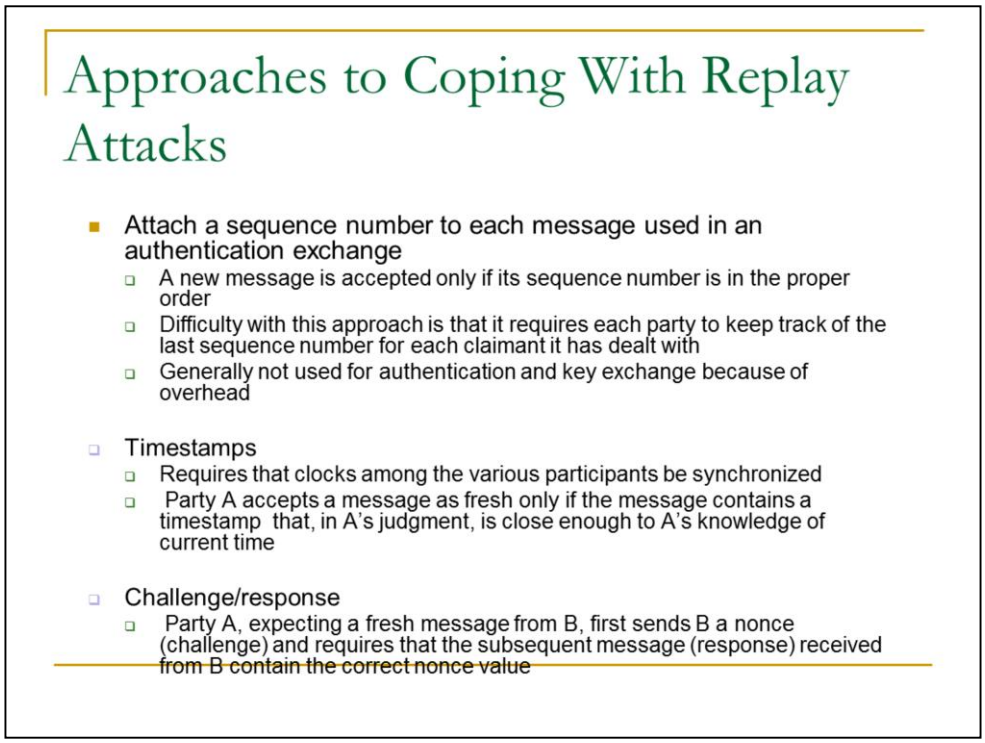
3. As with example (2), an opponent can replay a timestamped message within
the valid time window, but in addition, the opponent suppresses the original
message. Thus, the repetition cannot be detected.

4. Another attack involves a backward replay without modification. This is a

replay back to the message sender. This attack is possible if symmetric encryption

is used and the sender cannot easily recognize the difference between

messages sent and messages received on the basis of content.



One approach to coping with replay attacks is to attach a sequence number to

each message used in an authentication exchange. A new message is accepted only if

its sequence number is in the proper order. The difficulty with this approach is that

it requires each party to keep track of the last sequence number for each claimant it

has dealt with. Because of this overhead, sequence numbers are generally not used

for authentication and key exchange. Instead, one of the following two general approaches

is used:

• Timestamps: Party A accepts a message as fresh only if the message contains a

timestamp that, in A's judgment, is close enough to A's knowledge of current

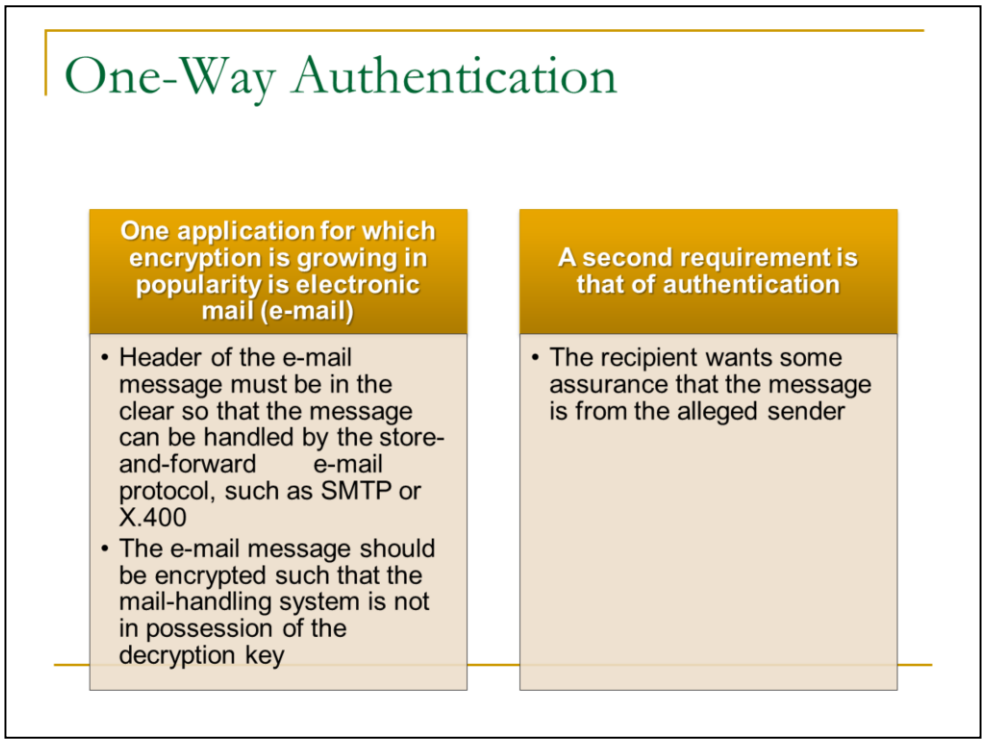
time. This approach requires that clocks among the various participants

be
synchronized.

• Challenge/response: Party A, expecting a fresh message from B, first sends B a nonce (challenge) and requires that the subsequent message (response) received from B contain the correct nonce value.

It can be argued (e.g., [LAM92a]) that the timestamp approach should not be used for connection-oriented applications because of the inherent difficulties with this technique. First, some sort of protocol is needed to maintain synchronization among the various processor clocks. This protocol must be both fault tolerant, to cope with network errors, and secure, to cope with hostile attacks. Second, the opportunity for a successful attack will arise if there is a temporary loss of synchronization resulting from a fault in the clock mechanism of one of the parties. Finally, because of the variable and unpredictable nature of network delays, distributed clocks cannot be expected to maintain precise synchronization. Therefore, any timestamp-based procedure must allow for a window of time sufficiently large to accommodate network delays yet sufficiently small to minimize the opportunity for attack.

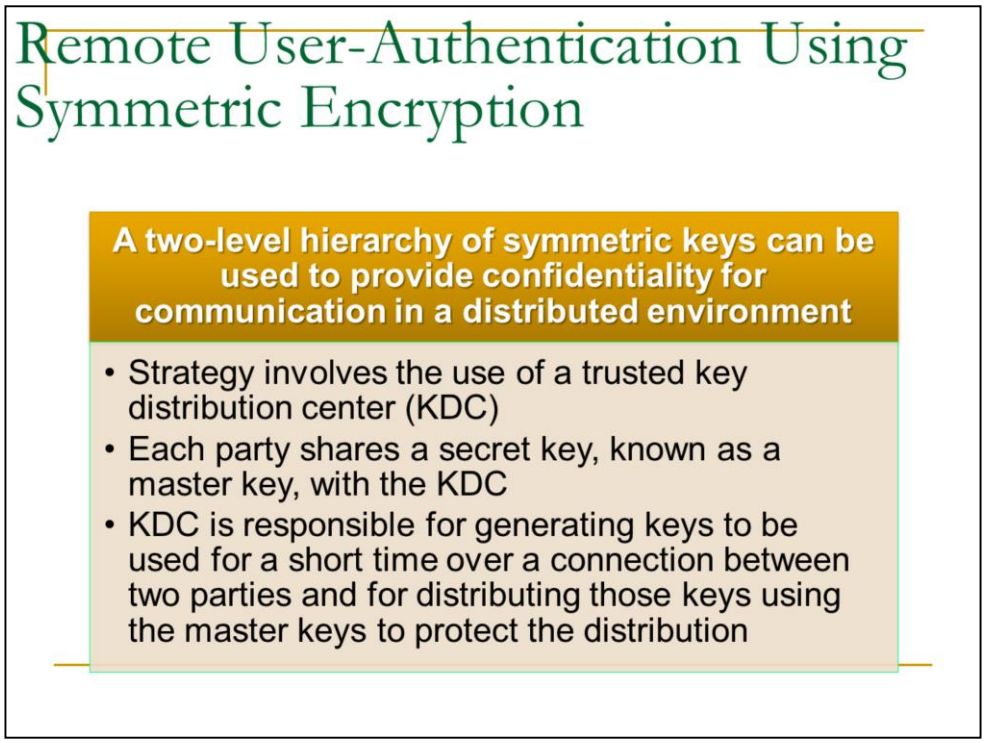
On the other hand, the challenge-response approach is unsuitable for a connectionless type of application, because it requires the overhead of a handshake before any connectionless transmission, effectively negating the chief characteristic of a connectionless transaction. For such applications, reliance on some sort of secure time server and a consistent attempt by each party to keep its clocks in synchronization may be the best approach (e.g., [LAM92b]).



One application for which encryption is growing in popularity is electronic mail (e-mail). The very nature of electronic mail, and its chief benefit, is that it is not necessary for the sender and receiver to be online at the same time. Instead, the e-mail message is forwarded to the receiver's electronic mailbox, where it is buffered until the receiver is available to read it.

The "envelope" or header of the e-mail message must be in the clear, so that the message can be handled by the store-and-forward e-mail protocol, such as the Simple Mail Transfer Protocol (SMTP) or X.400. However, it is often desirable that the mail-handling protocol not require access to the plaintext form of the message, because that would require trusting the mail-handling mechanism. Accordingly, the e-mail message should be encrypted such that the mail-handling system is not in possession of the decryption key.

A second requirement is that of authentication . Typically, the recipient wants some assurance that the message is from the alleged sender.



As was discussed in Chapter 14, a two-level hierarchy of symmetric encryption keys

can be used to provide confidentiality for communication in a distributed environment.

In general, this strategy involves the use of a trusted key distribution center

(KDC). Each party in the network shares a secret key, known as a master key, with

the KDC. The KDC is responsible for generating keys to be used for a short time

over a connection between two parties, known as session keys, and for distributing

those keys using the master keys to protect the distribution. This approach is quite

common. As an example, we look at the Kerberos system in Section 15.3. The discussion

in this subsection is relevant to an understanding of the Kerberos mechanisms.

Needham-Schroeder Protocol

- Original third-party key distribution protocol for session between A B mediated by KDC
- Protocol overview is:
  1. A->KDC: $ID_A \| ID_B \| N_1$
  2. KDC -> A: $E(K_a,[K_s \| ID_B \| N_1 \| E(K_b,[K_s \| ID_A])])$
  3. A -> B: $E(K_b, [K_s \| ID_A])$
  4. B -> A: $E(K_s, [N_2])$
  5. A -> B: $E(K_s, [f(N_2)])$

The Needham-Schroeder Protocol is the original, basic key exchange protocol, as was shown in Stallings Figure 14.3 (previous chapter). Used by 2 parties who both trusted a common key server, it gives one party the info needed to establish a session key with the other. Note that since the key server chooses the session key, it is capable of reading/forging any messages between A&B, which is why they need to trust it absolutely!

Note that all communications is between A&KDC and A&B, B&KDC don't talk directly (though indirectly a message passes from KDC via A to B, encrypted in B's key so that A is unable to read or alter it). Other variations of key distribution protocols can involve direct communications between B&KDC.

Needham-Schroeder Protocol

- Used to securely distribute a new session key for communications between A & B
- But is vulnerable to a replay attack if an old session key has been compromised
  - then message 3 can be resent convincing B that is communicating with A
- Modifications to address this require:
  - timestamps in steps 2 & 3 (Denning 81)(we will cover in next workshop)
  - using an extra nonce (Neuman 93)

There is a critical flaw in the protocol, as shown. The message in step 3 can be decrypted, and hence understood, supposedly only by B. But if an opponent, X, has been able to compromise an old session key, then X can impersonate A and trick B into using the old key by simply replaying step 3. Admittedly, this is a much more unlikely occurrence than that an opponent has simply observed and recorded step 3. It can however be corrected by either using timestamps, or an additional nonce, with respective advantages and limitations, see text for discussion.

This example emphasises the need to be extremely careful in codifying assumptions, and tracking the timeliness of the flow of info in protocols. Designing secure protocols is not easy, and should not be done lightly. Great care and analysis is needed.

Denning 81:

Needham-Schroeder Protocol

- Denning 81 Modification

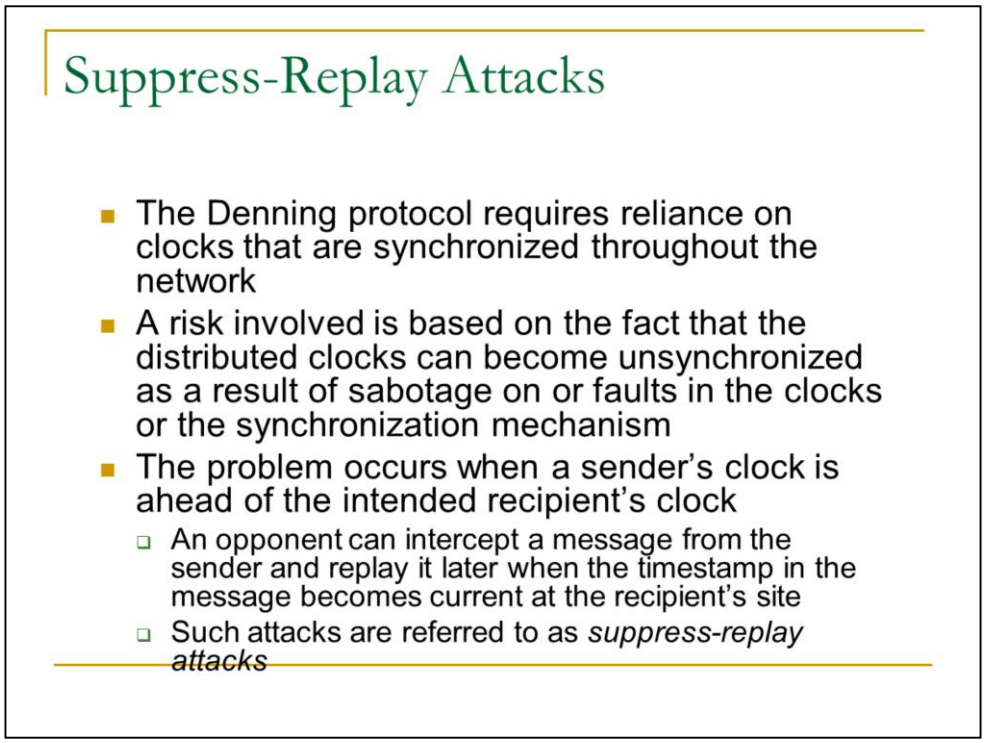
1. A → KDC: $ID_A \| ID_B$
2. KDC → A: $E(K_a, [K_s \| ID_B \| T \| E(K_b, [K_s \| ID_A \| T])])$
3. A → B: $E(K_b, [K_s \| ID_A \| T])$
4. B → A: $E(K_s, N_1)$
5. A → B: $E(K_s, f(N_1))$

- Neuman 93 Modification

1. A → B: $ID_A \| N_a$
2. B → KDC: $ID_B \| N_b \| E(K_b, [ID_A \| N_a \| T_b])$
3. KDC → A: $E(K_a, [ID_B \| N_a \| K_s \| T_b]) \| E(K_b, [ID_A \| K_s \| T_b]) \| N_b$
4. A → B: $E(K_b, [ID_A \| K_s \| T_b]) \| E(K_s, N_b)$

T is a timestamp that assures A and B that the session key has only just been generated.

Thus, both A and B know that the key distribution is a fresh exchange. A and B can verify timeliness by checking that | Clock – *T* | < *delta t*1 + delta *t*2



The Denning protocol seems to provide an increased degree of security compared to the Needham/Schroeder protocol. However, a new concern is raised: namely, that this new scheme requires reliance on clocks that are synchronized throughout the network. [GONG92] points out a risk involved. The risk is based on the fact that the distributed clocks can become unsynchronized as a result of sabotage on or faults in the clocks or the synchronization mechanism.2 The problem occurs when a sender's clock is ahead of the intended recipient's clock. In this case, an opponent can intercept a message from the sender and replay it later when the timestamp in the message becomes current at the recipient's site. This replay could cause unexpected results. Gong refers to such attacks as suppress-replay attacks .

One way to counter suppress-replay attacks is to enforce the requirement that parties regularly check their clocks against the KDC's clock. The other alternative, which avoids the need for clock synchronization, is to rely on handshaking protocols using nonces. This latter alternative is not vulnerable to a suppress-replay attack, because the nonces the recipient will choose in the future are unpredictable to the sender. The Needham/Schroeder protocol relies on nonces only but, as we have seen, has other vulnerabilities.

Main Application

- Kerberos: See next set of notes.

Authentication using Public Key Approach

- Many varieties of protocols exist.
- We saw before a scheme based on public key encryption.
- Important issue is each party should have correct public key of the other
- A method using Authentication Server is a possibility.
- Various protocols exist making use of time stamps and nonces.