

**Password Servers**

In networked environments, want user to be able to use any terminal/machine/device. Leads to use of Password Servers.

Two basic protocols ...

**Password verification at the server:**

1. Terminal → Server: Username, Password Info
2. Server → Terminal: Yes/No

(Server verifies password)

(Note: password information need not be equal to password).

**Password verification at the Terminal:**

1. Terminal → Server: Username
2. Terminal verifies password

Server → Terminal: Password Info for Username

(Terminal verifies password)

**Authentication Protocols**

**Basics**

Authentication = provision of high degree of confidence in the identity of an entity

e.g.

1. authentication of an person to a device
2. authentication of a device to another device

(Password/Biometric/Token based mechanisms)

(Cryptographic mechanisms)

**Dictionary Attacks**

Suppose: passwords are hashed using  $h$ , and Eve is able to see the  $h(\text{Password})$  values from the database, or transmitted on the wire:

1. Eve precomputes and saves pairs  $(M, h(M))$  for  $M$  ranging over highly probable passwords, e.g. words in the dictionary, names, dates
2. for each hashed password value  $h(P)$  observed, Eve checks if it is in the database
3. If so, Eve the knows value  $M$  passes the password test.

Experimentally determined hit rate of 30%!

**Preventing Dictionary Attacks with Salt**

Password Database =  $(\text{Username}, R, h(R, \text{Password}))$   
 where for each Username,  $R$  is a distinct random number called the *salt*  
 Eve now has to do more work: needs to recompute the whole dictionary for each  $R$  observed.

**Cryptographic Protection of Passwords**

Storing a password database  $(\text{User}, \text{Password})$  entries in plaintext runs the risk that if the machine/root user password is compromised, *all* user passwords are compromised.

Remedies ..

**Password encryption:**

1. store  $(\text{User}, \{\text{Password}\}_K)$
2. verification test: accept  $(U, P)$  if  $(U, M)$  in database and  $\{M\}_{K^{-1}}=P$
3. need to secure decryption key  $K^{-1}$

**Password Hashing:**

1. store  $(\text{User}, h(\text{Password}))$  where  $h$  is a one-way function
2. verification test: accept  $(U, P)$  if  $(U, M)$  in database and  $h(P)=M$
3. some systems even allow all users to access database

### Key Based Mechanisms

Password mechanisms use “proof of possession of a secret” as the authentication mechanism  
 Can use keys for this also

#### Shared key handshake:

Alice and Bob have shared key  $K_{AB}$

1. A  $\rightarrow$  B: Hi, this is Alice

(B generates a random number N)

2. B  $\rightarrow$  A: N

3. A  $\rightarrow$  B:  $\{N\}_{K_{AB}}$

this authenticates Alice to Bob

#### Public Key handshake

Alice has public signature verification key  $K_A$ , known to Bob

1. A  $\rightarrow$  B: Hi, this is Alice

(B generates a random number N)

2. B  $\rightarrow$  A: N

3. A  $\rightarrow$  B: [N, signed using  $K_A^{-1}$ ]

### Replay Attacks

Suppose Mallory has control of the message channels.

In a *replay attack*, Mallory saves messages observed during previous runs of the protocol, and attempts to use these to trick the principals. In the proposed protocol, message 3 could be a replay, so proves nothing

A proposed 2-way authentication using shared keys:

(A generates a random number N)

1. A  $\rightarrow$  B: Hi, I'm Alice, N

2. B  $\rightarrow$  A:  $\{N\}_{K_{AB}}$

3. A  $\rightarrow$  B:  $\{N\}_{K_{AB}}$

Another two way handshake:

1.  $A \rightarrow B: H, I, \text{Im Alice}, N1$
2.  $B \rightarrow A: N2, \{N1\}_{K_{AB}}$
3.  $A \rightarrow B: \{N2\}_{K_{AB}}$

where  $f$  is some function, e.g.  $f(N)=N+1$

1.  $A \rightarrow B: H, I, \text{Im Alice}, N$
  2.  $B \rightarrow A: \{N\}_{K_{AB}}$
  3.  $A \rightarrow B: \{f(N)\}_{K_{AB}}$
- (A generates a random number  $N$ )

An improvement:

An attack:

- (run1.1)  $I \rightarrow B: H, I, \text{Im Alice}, N1$
- (run1.2)  $B \rightarrow I: N2, \{N1\}_{K_{AB}}$
- (run2.1)  $I \rightarrow B: H, I, \text{Im Alice}, N2$
- (run2.2)  $B \rightarrow I: N3, \{N2\}_{K_{AB}}$
- (run1.3)  $I \rightarrow B: \{N2\}_{K_{AB}}$

### Time-stamp Based Authentication

1.  $A \rightarrow B: H, I, \text{this is Alice}, K_{AB}(\text{timestamp})$
- (Bob verifies  $|\text{clock}_B - \text{timestamp}| > \delta$ )

advantages:

1. only one message needed
2. No need for Alice/Bob to remember values such as  $N$

## Shared Key Authentication Protocols

Assumptions:  
S is an *authentication server* trusted to

1. generate good *session keys* for use by Alice and Bob
2. preserve the secrecy of these keys

S has already established a shared key  $K_{AS}$  with each principal  $A$

Issues/disadvantages:

1.  $K_{AB}(\text{timestamp})$  can be replayed while  $|\text{clock}_B - \text{timestamp}| < \delta$   
solution: Bob remembers timestamp values while this is possible
2. now need to secure Bob's clock values, and ensure that it is highly accurate
3. Authentication breaks down if Bob's clock drifts too far. (Use a different authentication protocol for the user who resets the clock!)

Authentication typically not enough: if the session is carried out in plaintext after authentication, an intruder could

1. eavesdrop on the session
2. hijack the session

So typically, authentication protocols also establish a shared key for the session

### Needham-Schroeder Shared Key Protocol

Alice wants to start a session with Bob:

1. A generates  $N$ , a random number)

1. A  $\rightarrow$  S: A,B,N

(S generates session key  $K_{AB}$ )

2. S  $\rightarrow$  A:  $\{K_{AB}, B, N, \{K_{AB}, A\}_{K_{BS}}\}_{K_{AS}}$

3. A  $\rightarrow$  B:  $\{K_{AB}, A\}_{K_{BS}}, \{N2\}_{K_{AB}}$

4. B  $\rightarrow$  A:  $\{N2-1, N3\}_{K_{AB}}$

5. A  $\rightarrow$  B:  $\{N3-1\}_{K_{AB}}$

1. A  $\rightarrow$  S: A,B,N  
 "I'm Alice, I want to talk to Bob", and N is an identifier for this request

2. S  $\rightarrow$  A:  $\{K_{AB}, B, N, \{K_{AB}, A\}_{K_{BS}}\}_{K_{AS}}$   
 Only A and S can read this.

$K_{AB}$  the session key to be used by Alice and Bob

presence of N indicates that this is in response to message 1

$\{K_{AB}, A\}_{K_{BS}}$  is a message from S for forwarding to Bob: "Use  $K_{AB}$  to talk to A"

3. A  $\rightarrow$  B:  $\{K_{AB}, A\}_{K_{BS}}, \dots$

second part of message 3 is part of a two way handshake using  $K_{AB}$ :

3. A  $\rightarrow$  B:  $\dots, \{N2\}_{K_{AB}}$

4. B  $\rightarrow$  A:  $\{N2-1, N3\}_{K_{AB}}$

5. A  $\rightarrow$  B:  $\{N3-1\}_{K_{AB}}$

**A problem with Needham-Schroeder**

Suppose Mallory stores old messages

2.  $S \rightarrow A : \{K_{AB}, B, N, \{K_{AB}, A\}_{K_{BS}}\}_{K_{AS}}$

mounts an attack on Alice's system, and learns  $K_{AS}$ .

Even if Alice discovers the attack and switches to a new key  $K'_{AS}$  by

the time Mallory learns  $K_{AS}$ , Mallory can still run

3.  $M(A) \rightarrow B : \{K_{AB}, A\}_{K_{BS}}, \{N2\}_{K_{AB}}$

4.  $B \rightarrow M(A) : \{N2-1, N3\}_{K_{AB}}$

5.  $M(A) \rightarrow B : \{N3-1\}_{K_{AB}}$

and convince Bob he's talking to Alice.

**A fixed variant of Needham-Schroeder**

1.  $A \rightarrow B : Hi, I'm Alice$

2.  $B \rightarrow A : K_{BS}(NB)$

3.  $A \rightarrow S : A, B, N, K_{BS}(NB)$

(S generates session key  $K_{AB}$ )

2.  $S \rightarrow A : \{K_{AB}, B, N, \{K_{AB}, A, NB\}_{K_{BS}}\}_{K_{AS}}$

3.  $A \rightarrow B : \{K_{AB}, A, NB\}_{K_{BS}}, \{N2\}_{K_{AB}}$

4.  $B \rightarrow A : \{N2-1, N3\}_{K_{AB}}$

5.  $A \rightarrow B : \{N3-1\}_{K_{AB}}$

**Needham Schroeder Public Key Protocol**

(Omitting steps whereby A and B obtain each other's public keys from the trusted server.)

1.  $A \rightarrow B : A, B, \{N_a, A\}_{PK_B}$

2.  $B \rightarrow A : B, A, \{N_b, N_a, N_b\}_{PK_A}$

3.  $A \rightarrow B : A, B, \{N_b\}_{PK_B}$

Main idea: the ticket to Bob now contains the nonce NB,  
 when Bob receives the ticket, this convinces Bob that Alice talked to  
 the key server S since NB was generated  
 Bob can terminate old attempts by Alice to connect by discarding  
 NB.

**Gavin Lowe's Fix**

Add  $B$  to the encrypted part of message 2:

1.  $A \rightarrow B : A, B, \{N_a, A\}_{PK_B}$
2.  $B \rightarrow A : B, A, \{N_a, N_b, B\}_{PK_A}$
3.  $A \rightarrow B : A, B, \{N_b\}_{PK_B}$

now step  $\alpha.2$  in the attack becomes

- $\alpha 2. I \rightarrow A : B, A, \{N_a, N_b, B\}_{PK_A}$

and  $A$  detects that the encrypted part of this message was not created by  $I$ .

(Lowe proves this fix works for any combination of interleaving of runs.)

**Gavin Lowe's attack**

We combine two runs of the protocol,  $\alpha : A \rightarrow I$  and  $\beta : I \rightarrow B$ . In  $\beta$ ,  $I$  poses as  $A$  (denoted  $I(A)$ ).

- $\alpha 1. A \rightarrow I : A, I, \{N_a, A\}_{PK_I}$
- $\beta 1. I(A) \rightarrow B : A, B, \{N_a, A\}_{PK_B}$
- $\beta 2. B \rightarrow I(A) : B, A, \{N_a, N_b\}_{PK_A}$
- $\alpha 2. I \rightarrow A : I, A, \{N_a, N_b\}_{PK_A}$
- $\alpha 3. A \rightarrow I : A, I, \{N_b\}_{PK_I}$
- $\beta 3. I(A) \rightarrow B : A, B, \{N_b\}_{PK_B}$

run  $\beta$  causes  $B$  to believe that it is talking to  $A$ , not  $I$ .