Chapter 16: Authentication in Distributed System

Ajay Kshemkalyani and Mukesh Singhal

Distributed Computing: Principles, Algorithms, and Systems

Cambridge University Press

Introduction

- A distributed system is susceptible to a variety of security threats.
- A principal can impersonate other principal and authentication becomes an important requirement.
- Authentication is a process by which one principal verifies the identity of other principal.
- In one-way authentication, only one principal verifies the identity of the other principal.
- In mutual authentication, both communicating principals verify each other's identity.

Background and definitions

- Authentication is a process of verifying that the principal's identity is as claimed.
- Authentication is based on the possession of some secret information, like password, known only to the entities participating in the authentication.
- When an entity wants to authenticate another entity, the former will verify if the latter possesses the knowledge of the secret.

A simple classification of authentication protocols

- Classified based on the cryptographic technique used.
- There are two basic types of cryptographic techniques: symmetric ("private key") and asymmetric ("public key").
- Symmetric cryptography uses a single private key to both encrypt and decrypt data. (Let $\{X\}_k$ denote the encryption of X using a symmetric key k and $\{Y\}_{k^{-1}}$ denote the decryption of Y using a symmetric key k.)
- Asymmetric cryptography, also called Public-key cryptography, uses a secret key (private key) that must be kept from unauthorized users and a public key that is made public. (For a principal x, K_x and K_x^{-1} denote its public and private keys, respectively.)
- Data encrypted with the public key can be decrypted only by the corresponding private key, and data signed with the private key can only be verified with the corresponding public key.

Authentication protocols with symmetric cryptosystem

- In a symmetric cryptosystem, authentication protocols can be designed using to the following principle:

  “If a principal can correctly encrypt a message using a key that the verifier believes is known only to a principal with the claimed identity (outside of the verifier), this act constitutes sufficient proof of identity.”

Basic Protocol

- Shown in Table below, where a principal P is authenticating itself to principal Q. (‘k’ denotes a secret key that is shared between only P and Q.)

P :	Create a message m = "I am P."
:	Compute $m' = \{m, Q\}_k$
$P \rightarrow Q$:	m, m'
Q :	verify $\{m, Q\}_k = m'$
:	if equal then accept; otherwise the authentication fails

Table: Basic Protocol

- The principal P encrypts a message m identity of Q using the symmetric key k and sends to Q.
- Principal Q encrypts the plaintext message and its identity to get the encrypted message.
- If it is equal to the encrypted message sent by P, then Q has authenticated P, else, the authentication fails.

Weaknesses

- A major weakness of the protocol is its vulnerability to replays. An adversary could masquerade as P by recording the message m, m' and later replaying it to Q.
- Since both plaintext message m and its encrypted version m' are sent together by P to Q, this method is vulnerable to known plaintext attacks.

Modified protocol with nonce

- To prevent replay attacks, we modify the protocol by adding a challenge-and-response step using nonce.
- A nonce ensures that old communications cannot be reused in replay attacks.

$P \rightarrow Q$	:	"I am P."
Q	:	generate nonce n
$Q \rightarrow P$	:	n
P	:	compute $m' = \{P, Q, n\}_k$
$P \rightarrow Q$	:	m'
Q	:	verify $\{P, Q, n\}_k = m'$
	:	if equal then accept; otherwise the authentication fails

Table: Challenge-and-response protocol using a nonce

- Replay is foiled by the freshness of nonce n and because n is drawn from a large space.

Weaknesses

- This protocol has scalability problem because each principal must store the secret key for every other principal it would ever want to authenticate .
- Also vulnerable to known plaintext attacks.

Wide-mouth frog protocol

- Principal A authenticates itself to principal B using a server S as follows:

$$\begin{aligned} A \rightarrow S: &\quad A, \{T_A, K_{AB}, B\}_{K_{AS}} \\ S \rightarrow B: &\quad \{T_S, K_{AB}, A\}_{K_{BS}} \end{aligned}$$

- A sends to S its identity and a packet encrypted with the key, K_{AS}, it shares with S. The packet contains the current timestamp, A's desired communication partner, and a randomly generated key K_{AB}, for communication between A and B.
- S decrypts the packet to obtain K_{AB} and then forwards this key to B in an encrypted packet that also contains the current timestamp and A's identity.
- B decrypts this message with the key it shares with S and retrieves the identity of the other party and the key, K_{AB}.
- Weaknesses: A global clock is required and the protocol will fail if the server S is compromised.

A protocol based on an authentication server

- Uses a centralized *authentication server S that* shares a secret key K_{XS} with every principal X in the system .

P→Q :	"I am P."
Q :	generate nonce n
Q→P :	n
P :	compute x = $\{P, Q, n\}_{K_{PS}}$
P→Q :	x
Q :	compute y = $\{P, Q, x\}_{K_{QS}}$
Q→A :	y
A :	recover P, Q, x from y by decrypting y with K_{QS}
:	recover P,Q, n from y by decrypting x with K_{PS}
:	compute m = $\{P, Q, n\}_{K_{QS}}$
A→Q :	m
Q :	independently compute $\{P, Q, n\}_{K_{QS}}$ and verify $\{P, Q, n\}_{K_{QS}} = m$
:	if equal, then accept; otherwise, the authentication fails

Table: A protocol using an authentication server

A protocol based on an authentication server

- The principal P sends its identity to Q. Q generates a nonce and sends this nonce to P.
- P then encrypts P, Q, n with the key K_{PS} and sends this encrypted value x to Q.
- Q then encrypts P, Q, x with K_{QS} and sends this encrypted value y to authentication server S.
- Since S knows both the secret keys, it decrypts y with K_{QS}, recovers x , decrypts x with K_{PS} and recovers P, Q, n.
- Server S then encrypts P, Q, n with key K_{QS} and sends the encrypted value m to Q.
- Q then computes P, Q, $n_{K\,QS}$ and verifies if this value is equal to the value received from S.
- If both values are equal, then authentication succeeds, else it fails.

One-time password scheme

- In the One-Time Password scheme, a password can only be used once.
- The system generates a list of passwords and secretly communicates this list to the client and the server.
- The client uses the passwords in the list to log on to a server.
- The server always expects the next password in the list at the next logon.
- Therefore, even if a password is disclosed, the possibility of replay attacks is eliminated because a password is used only once.
- Protocol consist of two stages:
- Registration stage: where the client registers with the server and gets a list of passwords.
- Login and authentication stage: where the server authenticates the client.

Registration

- Every client shares a pre-shared secret key, represented as SEED with the server.
- The server generates a session key (SK) with the help of a random number D and a timestamp T, i.e., $SK = D||T$.
- The server computes and sends $SEED \oplus SK$ to the client.
- When the client receives $SEED \oplus SK$, it computes the value of SK as follows:

$$SK = SEED \oplus (SEED \oplus SK)$$

- The client then generates an initial key IK with the help of a randomly generated secret key K,

$$IK = K \oplus SEED$$

- The client then decides the number of times (N) it wants to login to the server and sends the generated initial key (IK) to the server.
- The client performs $IK \oplus SK$ and $N \oplus SK$ and sends these values to the server.

Registration

- When the server receives IK $\oplus$ SK and N $\oplus$ SK, it retrieves IK and N from the received values and computes
  $$p_0 = H^N(IK) \text{ for the user where H is a Hash Function}$$
  and performs $p_0 = p_0 \oplus$ SK and stores p_0 and N in its database.
- It also computes p_1 and p_2 as follows:
  $$p_1 = H^{N-1}\ (IK) \text{ and}$$
  $$p_2 = H^{N-2}\ (IK)$$
- The server then sends $p_0 \oplus$ SK, $p_1 \oplus$ SK and $p_2 \oplus$ SK to the client
- On receiving $p_0 \oplus$ SK, $p_1 \oplus$ SK and $p_2 \oplus$ SK from the server, the client performs the XOR operation on SK and $p_0 \oplus$ SK, $p_1 \oplus$ SK and $p_2 \oplus$ SK separately, to obtain $p_{0,}$ $p_1$and $p_{2,}$ respectively.
- The client hashes IK for N times and then compares it with p_0. If both values are equal, the client is sure of the authenticity of the server and that it is not communicating with an intruder.
- It then saves the values of p_0, p_1, p_2 and N for future communication with the server.

Login and authentication

- Authentication requires the following steps:
- If the client is logging in for the tth time, the server generates a new session key (SK)

  SK = D||T where T is the timestamp and D is a random number.

  The server also computes $p_{t-1} = H^{C+1}(IK)$ where C=N-t. It then performs $p_{t-1} \oplus SK$ and $SK \oplus SEED$ (SEED is stored in the database) and sends these values to the client.
- On the receipt of the values from the server, the client computes SK as follows:

  $$SK = p_{t-1} \oplus (p_{t-1} \oplus SK)$$

  Then the Client checks the timestamp T of the session key SK. If the timestamp is valid, the client computes $SEED = SK \oplus (SK \oplus SEED)$ and checks the value of SEED with the one saved to make sure of the server's identity. If they match, the server's authenticity is verified.

Login and authentication

- The client proves its identity to the server as follows: It sends SK $\oplus$ p_t to the server. The client uses the p_t saved in the previous login in this EX-OR operation.
- Server calculates p_t from SK $\oplus$ p_t received from the client as follows:

$$p_t = SK \oplus (SK \oplus p_t)$$

- From the received p_t value, it calculates p_{t-1}= H (p_t) and compares it with p_{t-1} obtained in the Step 1. If both match, the identity of the client is verified.
- Finally, the server updates N with C, where C=N-t and computes p_{t+1} using p_0 and sends p_{t+1} $\oplus$ SK to the client.
- The client computes value of p_{t+1} as $p_{t+1=}$SK $\oplus$ (SK $\oplus$ p_{t+1}) and stores it for its next login.

Strength and weaknesses

- Since session key SK is obtained by using the timestamp, replay of previous session does not work and thus the scheme is robust against replay attacks.
- The use of hash function makes the Dictionary attacks impossible.
- One-time passwords that are not time-synchronized are vulnerable to phishing. Phishing usually occurs when a fraudster sends an email that contains a link to a fraudulent website where the users are asked to provide personal account information.

Otway-Rees protocol

- A server-based protocol that provides authenticated key transport without requiring timestamps.
- K_{AB} is a session key that the sever S generates for users A and B to share.
- N_A and N_B are nonces chosen by A and B, respectively.
- M is a nonce chosen by A which serves as a transaction identifier.
- S shares symmetric keys K_{AS} and K_{BS} with A, B, respectively.
- The protocol is shown in the following table.

(1) $A \rightarrow B$:	$M, A, B, (N_A, M, A, B)_{K_{AS}}$
(2) $B \rightarrow S$:	$M, A, B, (N_A, M, A, B)_{K_{AS}}, (N_B, M, A, B)_{K_{BS}}$
(3) $S \rightarrow B$:	$(N_A, K_{AB})_{K_{AS}}, (N_B, K_{AB})_{K_{BS}}$
(4) $B \rightarrow A$:	$M, (N_A, K_{AB})_{K_{AS}}$

Table: Otway Rees protocol

Weaknesses

- A malicious intruder can arrange for A and B to end up with different keys as follows:
  - A and B execute the first three messages; at this point, B has received the key K_{AB}.
  - The intruder intercepts the fourth message.
  - He/She replays step (2), which results in S generating a new key K'_{AB} and sending it to B in step (3).
  - The intruder intercepts this message, too, but sends to A the part of it that B would have sent to A.
  - So A has finally received the expected fourth message, but with K'_{AB} instead of K_{AB}.
- Another problem is that although the server tells B that A used a nonce, B doesn't know if this was a replay of an old message.

Kerberos authentication service

- Authentication in Kerberos is based on the use of a symmetric cryptosystem together with trusted third-party authentication servers.
- The basic components include authentication servers (*Kerberos servers*) and *ticket-granting servers* (TGSs).

Initial Registration

- Every Client/user registers with the Kerberos server by providing its user id, U and a password, password_u.
- The Kerberos server computes a key $k_u = f(\text{password}_u)$ using a one-way function f and stores this key in a database.
- k_u is a secret key that depends on the password of the user and is shared by client U and Kerberos server only.

The authentication protocol

Authentication in Kerberos proceeds in three steps:

- Initial Authentication at Login: Kerberos Server authenticates user login at a host and installs a ticket for the ticket granting server, TGS, at the login host.
- Obtain a ticket for the server: Using the ticket for the ticket granting server, the client requests the ticket granting server, TGS, for a ticket for the server.
- Requesting Service from the server: The client uses the server ticket obtained from the TGS to request services from the server.

Initial Authentication at Login

- Initial Authentication at Login uses Kerberos server and is shown in the following Table. Let U be a user who is attempting to log in a host H.

1) U→ H :	U
2) H→Kerberos :	U, TGS
3) Kerberos :	retrieve k_U and k_{TGS} from database
:	generate new session key k
:	create a ticket-granting ticket
:	$tick_{TGS} = \{U, TGS, k, T, L\}_{K_{TGS}}$
4) Kerberos →H :	$\{TGS, k, T, L, tick_{TGS}\}_{k_U}$
5) H→ U :	“Password?”
6) U→ H :	password
7) H :	compute $k'_U = f(password)$
:	recover k, $tick_{TGS}$ by decrypting
:	$\{TGS, k, T, L, tick_{TGS}\}_{k_U}$ with k'_U
:	if decryption fails, abort login, otherwise, retain
:	$tick_{TGS}$ and k.
:	erase password from the memory

Table: Initial Authentication at Login

Obtain a ticket for the server

- The client executes steps shown in the folloing table to request a ticket for the server from TGS.
- The client sends the ticket $tick_{TGS}$ to TGS, requesting it a ticket for the server S. (T_1 and T_2 are timestamps).

1) C $\rightarrow$ TGS :	S, $tick_{TGS}$, $\{C, T_1\}_k$
2) TGS :	recover k from $tick_{TGS}$ by decrypting with k_{TGS},
	recover T_1 from $\{C, T_1\}_k$ by decrypting with k
	check timelines of T_1 with respect to local clock
	generate new session key k.
	Create server ticket $tick_S = \{C, S, k, T, L\}_{k_S}$
3) TGS$\rightarrow$ C :	$\{S, k, T, L, tick_S\}_k$
4) C :	recover k, $tick_S$ by decrypting the message with k

Table: Obtain a ticket for the server

Requesting service from the server

- Client C sends the ticket and the authenticator to server.
- The server decrypts the $tick_S$ and recovers k.
- It then uses k to decrypt the authenticator $\{C, T_2\}_{k'}$ and checks if the timestamp is current and the client identifier matches with that in the $tick_S$ before granting service to the client.
- If mutual authentication is required, the server returns an authenticator.

1) C→S :	$tick_S$, $\{C, T_2\}_{k'}$
2) S :	recover k from $tick_S$ by decrypting it with k_S
	recover T_2 from $\{C, T_2\}_k$ by decrypting with k
	check timeliness of T_2 with respect to the local clock
3) S→C : $\{T_2 + 1\}_k$	

Table: Requesting service from the server

Protocols based on asymmetric cryptosystems

- In an asymmetric cryptosystem, let k_p denote the public key and k_p^{-1} denote the private key of a principal P.
- Only P can generate $\{\mathrm{m}\}_{k_p^{-1}}$ for any message m by signing it using k_p^{-1}.
- The signed message $\{\mathrm{m}\}_{k_p^{-1}}$ can be verified by any principal with the knowledge of k_p
- Authentication protocols can be constructed using the following design principle:

  "If a principal can correctly sign a message using the private key of the claimed identity, this act constitutes a sufficient proof of the identity."

The basic protocol

- A basic protocol is as follows:

P→Q :	"I am P."
Q :	generate nonce n
Q→P :	n
P :	compute m = $\{P, Q, n\}_{k_p^{-1}}$
P→Q :	m
Q :	verify (P, Q, n) = $\{m\}_{k_p}$
:	if equal, then accept; otherwise, the authentication fails

Table: A Basic protocol

A modified protocol with a certification authority

- The basic protocol can be modified as shown in the following table:

$P \rightarrow Q$:	"I am P."
Q :	generate nonce n
$Q \rightarrow P$:	n
P :	compute $m = \{P, Q, n\}_{k_p^{-1}}$
$P \rightarrow Q$:	m
$Q \rightarrow CA$:	"I need P's public key."
CA :	retrieve public key $k_P$of P from key Database Create certificate $c = \{P, k_P\}_{k_{CA}^{-1}}$
$CA \rightarrow Q$:	P, c
Q :	recover P, k_P from c by decrypting with k_{CA} verify $(P, Q, n) = \{m\}_{k_P}$
:	if equal, then accept; otherwise, the authentication fails

Table: A modified protocol with a certification authority, CA

A modified protocol with a certification authority

- A certification authority CA is involved in the authentication process.
- When Q receives a message encrypted with P's private key from P, it requests the authentication server for P's public key.
- CA retrieves public key of P from the key database and provides Q with a certificate for P's public key.
- The certificate, $\{P, k_P\}_{k_{CA}^{-1}}$ contains P's identity and its public key, encrypted with the private key of the certification authority.
- Q retrieves the public key of P by decrypting the certificate with the public key of CA.
- Then it decrypts the message m, it received from P using the public key k_P and checks if $\{m\}_{k_P}$ equals $\{P, Q, n\}$.
- If both are equal, authentication succeeds, else it fails.

Needham and Schroeder protocol

- The Needham-Schroeder protocol uses a trusted key server that issues certificates containing the public key of a user.
- The protocol is described in the following table:

1.	$A \rightarrow S$:	A, B
2.	$S \rightarrow A$:	$\{K_b, B\}_{K_s^{-1}}$
3.	$A \rightarrow B$:	$\{N_a, A\}_{K_b}$
4.	$B \rightarrow S$:	B, A
5.	$S \rightarrow B$:	$\{K_a, A\}_{K_s^{-1}}$
6.	$B \rightarrow A$:	$\{N_a, N_b\}_{K_a}$
7.	$A \rightarrow B$:	$\{N_b\}_{K_b}$

Table: Needham-Schroeder protocol

Needham and Schroeder protocol

- In step 1, A sends a message to the server S, requesting B's public key.
- S responds by returning B's public key K_b along with B's identity encrypted using S's secret key.
- A then seeks to establish a connection with B by selecting a nonce N_a, and sending it along with its identity to B encrypted using B's public key.
- When B receives this message, it decrypts the message to obtain the nonce N_a and to learn that user A is trying to communicate with it. It then requests the public key of A from server S which the server sends to B in message 5.
- B then returns nonce N_a, along with a new nonce N_b, to A, encrypted with A's public key .
- When A receives this message, it decrypts it with its private key and is assured that it is talking to B, since only B could have decrypted message in step 3 to obtain N_a.
- A then returns nonce N_b to B, encrypted with B's key. When B receives this message, it is assured that it is talking to A, since only A could have decrypted message in step 6 to obtain N_b.

Weaknesses

- This protocol provides no guarantee that the public keys obtained are current and not replays of old, possibly compromised keys.
- This problem can be overcome in various ways.
  - First method is that the server S includes timestamps in messages 2 and 5; however, this requires synchronized clocks at processes.
  - Another method is that A sends a nonce in message 1 and S returns the same nonce in message 2.
- The protocol is vulnerable to impersonation attacks (described next).

An impersonation attack on the protocol

- An impersonation attack is shown in the following Table:

1.3 $A \rightarrow I$:	$\{N_a, A\}_{K_i}$
2.3 $I(A) \rightarrow B$:	$\{N_a, A\}_{K_b}$
2.6 $B \rightarrow I(A)$:	$\{N_a, N_b\}_{K_a}$
1.6 $I \rightarrow A$:	$\{N_a, N_b\}_{K_a}$
1.7 $A \rightarrow I$:	$\{N_b\}_{K_i}$
2.7 $I(A) \rightarrow B$:	$\{N_b\}_{K_b}$

Table: An Impersonation attack on Needham-Schroeder Protocol

An impersonation attack on the protocol

- In step 1.3, A starts to establish a session with I, sending it a nonce N_a.
- In step 2.3, the intruder impersonates A to try to establish a false session with B sending it the nonce N_a obtained in the previous message from A.
- B responds in step 2.6 by selecting a new nonce N_b and returning it, along with N_a to A. The intruder intercepts this message, but cannot decrypt it because it is encrypted with A's public key.
- The intruder uses A as an oracle, by forwarding the message to A in step 1.6; note that this message is of the form expected by A in run 1 of the protocol. A decrypts the message to obtain N_b and returns this to I in step 1.7.
- I decrypts this message to obtain N_b and returns it to B in step 2.7, thus completing run 2 of the protocol. After B receives the message in step 2.7, B is led to believe that A has correctly established a session with it.

A solution to the attack

- The main cause of this attack is that step 6 does not contain the identity of the responder.
- If we include the responder's identity in step 6 of the protocol then the intruder I can not successfully replay this message in step 1.6 because A is expecting a message containing I's identity.

SSL protocol

- SSL, Secure Sockets Layer, protocol developed by Netscape and is the standard Internet protocol for secure communications.
- SSL typically is used between server and client to secure the connection.
- SSL protocol allows client/server applications to communicate so that eavesdropping, tampering, and message_forgery are prevented.
- One advantage of SSL is that it is application protocol independent.

SSL protocol

- The SSL protocol provides the following features:

• End point authentication:	The server is the "real" party that a client wants to talk to, not someone faking the identity.
• Message integrity:	If the data exchanged with the server has been modified along the way, it can be easily detected.
• Confidentiality:	Data is encrypted. A hacker cannot read your information by simply looking at the packets on the network.

SSL record protocol

- The record protocol fragments the data into manageable blocks, optionally compresses the data, applies MAC, encrypts adds a header and transmits the resulting unit into a TCP segment.
- Received data are decrypted, verified, decompressed and reassembled and then delivered into high level users.

SSL handshake protocol

- The SSL Handshake Protocol allows the server and client to authenticate each other and to negotiate an encryption algorithm and cryptographic keys before the application protocol transmits data.
  The following steps are involved in the SSL handshake:
- The SSL client sends a "client hello" message that lists cryptographic information such as the SSL version and, in the client's order of preference, the CipherSuites supported by the client. The message also contains a random byte string.
- The SSL server responds with a "server hello" message that contains the CipherSuite chosen by the server from the list provided by the SSL client, the session ID and another random byte string. The SSL server also sends its digital certificate. If the server requires a digital certificate for client authentication, the server sends a "client certificate request".

SSL handshake protocol

- The SSL client verifies the digital signature on the SSL server's digital certificate and checks that the CipherSuite chosen by the server is acceptable.
- The SSL client, usind all data generated in the handshake so far, creates a premaster secret for the session that enables both the client and the server to compute the secret key to be used for encrypting subsequent message data.
- If the SSL server sent a "client certificate request", the SSL client sends another signed piece of data which is unique to this handshake and known only to the client and server, along with the encrypted premaster secret and the client's digital certificate, or a "no digital certificate alert".
- The SSL server verifies the signature on the client certificate.
- The SSL client sends the SSL server a "finished" message, which is encrypted with the secret key, indicating that the client part of the handshake is complete.

SSL handshake protocol

- The SSL server sends the SSL client a "finished" message, which is encrypted with the secret key, indicating that the server part of the handshake is complete.
- The SSL server and SSL client can now exchange messages that are encrypted with the shared symmetric secret key.
- During both client and server authentication, there is a step that requires data to be encrypted with one of the keys in an asymmetric key pair and is decrypted with the other key of the pair.
- For server authentication, the client uses the server's public key to encrypt the data that is used to compute the secret key. The server can generate the secret key only if it can decrypt that data with the correct private key.

How SSL provides authentication

- For client authentication, the server uses the public key in the client certificate to decrypt the data the client sends during step 5 of the handshake.
- The exchange of finished messages confirms that authentication is complete.
- If any of the authentication steps fails, the handshake fails and the session terminates.
- The exchange of digital certificates during the SSL handshake is a part of the authentication process. (CA X issues the certificate to the SSL client, and CA Y issues the certificate to the SSL server.)

Password-based authentication

- The use of passwords is very popular to achieve authentication because of low cost and convenience.
- However, people tend to pick a password that is convenient, i.e., short and easy to remember. (Vulnerable to a password-guessing attack.)
- *Off-line dictionary attack*: an adversary builds a database of possible passwords, called a dictionary. The adversary picks a password from the dictionary and checks if it works. This may amount to generating a response to a challenge or decrypting a message using the password or a function of the password.After every failed attempt, the adversary picks a different password from the dictionary and repeats the process.
- Preventing Off-line Dictionary Attacks:
  By producing a cryptographically strong shared secret key, called the session key. This session key can be used by both entities to encrypt subsequest messages for a seceret session.

Encrypted key exchange (EKE) protocol

- Bellovin and Merritt developed a password-based encrypted key exchange (EKE) protocol using a combination of symmetric and asymmetric cryptography.
-

1. $A : (E_A, D_A)$, K_{pwd}=f(pwd). {* f is a function. *}
2. $A \rightarrow B : A, \{K_{pwd}\}_{E_A}$.
3. B : Compute $E_A = \{\{E_A\}_{K_{pwd}}\}_{K_{pwd}^{-1}}$ and generate a random secret key K_{AB}.
4. $B \rightarrow A : \{\{K_{AB}\}_{E_A}\}_{\{K_{pwd}\}}$.
5. $A : K_{AB} = \{\{\{\{K_{AB}\}_{E_A}\}_{\{K_{pwd}\}}\}_{K_{pwd}^{-1}}\}_{D_A}$. Generate a unique challenge C_A.
6. $A \rightarrow B : \{C_A\}_{K_{AB}}$.
7. B : Compute $C_A = \{\{C_A\}_{K_{AB}}\}_{K_{AB}^{-1}}$ and generate a unique challenge C_B.
8. $B \rightarrow A : \{C_A, C_B\}_{K_{AB}}$.
9. A : Decrypt message sent by B to obtain C_A and C_B.
   Compare the former with his own challenge. If they match,
   go to the next step, else abort.
10. $A \rightarrow B : \{C_B\}_{K_{AB}}$.

Figure: Encrypted Key Exchange Protocol

Encrypted key exchange (EKE) protocol

- Step 1: A generates a public/private key pair (E_A, D_A) and derives a secret key K_{pwd} from his password pwd.
- Step 2: A encrypts his public key E_A with K_{pwd} and sends it to B.
- Steps 3 and 4: B decrypts the message and uses E_A together with K_{pwd} to encrypt a session key K_{AB} and sends it to A.
- Steps 5 and 6: A uses this session key to encrypt a unique challenge C_A and sends the encrypted challenge to B.
- Step 7: B decrypts the message to obtain the challenge and generates a unique challenge C_B.
- Step 8: B encrypts $\{C_A,\ C_B\}$ with the session key K_{AB} and sends it to A.

Encrypted key exchange (EKE) protocol

- Step 9: A decrypts this message to obtain C_A and C_B and compares the former with the challenge it had sent to B. If they match, B is authenticated.
- Step 10: A encrypts B's challenge C_B with the session key K_{AB} and sends it to B. When B receives this message, it decrypts the message to obtain C_B and uses it to authenticate A.
- The resulting session key is stronger than the shared password and can be used to encrypt sensitive data.
- A Drawback: The EKE protocol suffers from the plain-text equivalence (the user and the host have access to the same secret password or hash of the password).

Secure remote password (SRP) protocol

- Wu combined the technique of zero-knowledge proof with asymmetric key exchange protocols to develop a verifier-based protocol, called secure remote password (SRP) protocol.
- SRP protocol eliminates plain-text equivalence.
- All computations in SRP are carried out on the finite field $\mathbb{F}_n$, where n is a large prime. Let g be a generator of $\mathbb{F}_n$.
- Let A be a user and B be a server. Before initiating the SRP protocol, A and B do the following:
  1. A and B agree on the underlying field.
  2. A picks a password pwd, a random salt s and computes the verifier $v = g^x$, where $x = H(s, pwd)$ is the long-term private-key and H is a cryptographic hash function.
  3. B stores the verifier v and the salt s.

Secure remote password (SRP) protocol

- The protocol is shown in the following table:

1. $A \rightarrow B : A.$
2. $B \rightarrow A : s.$
3. $A : x = H(s, pwd); K_A = g^a.$
4. $A \rightarrow B : K_A.$
5. $B : K_B = v + g^b.$
6. $B \rightarrow A : K_B, r.$
7. $A : S = (K_B - g^x)^{a+rx}$ and $B : S = (K_A v^r)^b.$
8. $A, B : K_{AB} = H(S).$
9. $A \rightarrow B : C_A = H(K_A, K_B, K_{AB}).$
10. B verifies C_A and computes $C_B = H(K_A, C_A, K_{AB}).$
11. $B \rightarrow A : C_B.$
12. A verifies C_B. Accept if verification passes; abort otherwise.

Figure: Secure remote password (SRP) protocol

Secure remote password (SRP) protocol

- Step 1: A sends its username "A" to server B.
- Step 2: B looks-up A's verifier v and salt s and sends A his salt.
- Steps 3 and 4: A computes its long-term private-key $x = H(s, pwd)$, generates an ephemeral public-key $K_A = g^a$ where a is randomly chosen from the interval $1 < a < n$ and sends K_A to B.
- Steps 5 and 6: B computes ephemeral public-key $K_B = v + g^b$ where b is randomly chosen from the interval $1 < a < n$ and sends K_B and a random number r to A.
- Step 7: A computes $S = (K_B - g^x)^{a+rx} = g^{ab+brx}$ and B computes $S = (K_A v^r)^b = g^{ab+brx}$.
- Step 8: Both A and B use a cryptographically strong hash function to compute a session key $K_{AB} = H(S)$.

Secure remote password (SRP) protocol

- Step 9: A computes $C_A = H(K_A, K_B, K_{AB})$ and sends it to B as an evidence that it has the session key. C_A also serves as a challenge.
- Step 10: B computes C_A itself and matches it with A's message. B also computes $C_B = H(K_A, C_A, K_{AB})$.
- Step 11: B sends C_B to A as an evidence that it has the same session key as A.
- Step 12: A verifies C_B, accepts if the verification passes and aborts otherwise.
- None of the protocol are messages encrypted in the SRP protocol. Since neither the user nor the server has access to the same secret password or hash of the password, SRP eliminates plain-text equivalence.

Authentication protocol failures

- Realistic authentication protocols are notoriously difficult to design due the following main reasons:
- First, most realistic cryptosystems satisfy algebraic additional identities which may generate undesirable effects when combined with a protocol logic.
- Second, even after assuming that the underlying cryptosystem is perfect, unexpected interactions among the protocol steps can lead to subtle logical flaws.
- Third, assumptions regarding the environment and the capabilities of an adversary are not well defined.

Authentication protocol failures

- We illustrate the difficulty by showing an authentication protocol with a subtle weakness.
- Consider the following authentication protocol: (k_p and k_q are symmetric keys shared between P and A, and Q and A, respectively, where A is an authentication server. k is a session key.)

$$\begin{array}{ll}
1)\ P \rightarrow A: & P,\ Q,\ n_p \\
2)\ A \rightarrow P: & \{n_p,\ Q,\ k\ ,\ \{k,\ P\}_{k_Q}\}_{k_p} \\
3)\ P \rightarrow Q: & \{k,\ P\}_{k_Q} \\
4)\ Q \rightarrow P: & \{n_Q\}_K \\
5)\ P \rightarrow Q: & \{n_Q+1\}_K
\end{array}$$

Authentication protocol failures

- Message $\{k, P\}_{k_Q}$ in step (3) can only be decrypted by Q and hence can only be understood by Q.
- Step (4) reflects Q's knowledge of k, while step (5) assures Q of P's knowledge of k; hence the authentication handshake is based entirely on the knowledge of k.
- The subtle weakness in the protocol arises from the fact that the message $\{k, P\}_{k_Q}$ sent in step (3) contains no information for Q to verify its freshness. This is the first message sent to Q about P's intention to establish a secure connection.
- An adversary who has compromised an old session key k' can impersonate P by replaying the recorded message $\{k', P\}_{k_Q}$ in step (3) and subsequently executing the steps (4) and (5) using k'.

Authentication protocol failures

Remedies:

- To avoid protocol failures, formal methods may be employed in the design and verification of authentication protocols.
- A formal design method should embody the basic design principles.
- For example, informal reasoning such as "If you believe that only you and Bob know k , then you should believe any message you receive encrypted with k was originally sent by Bob." should be formalized by a verification method.