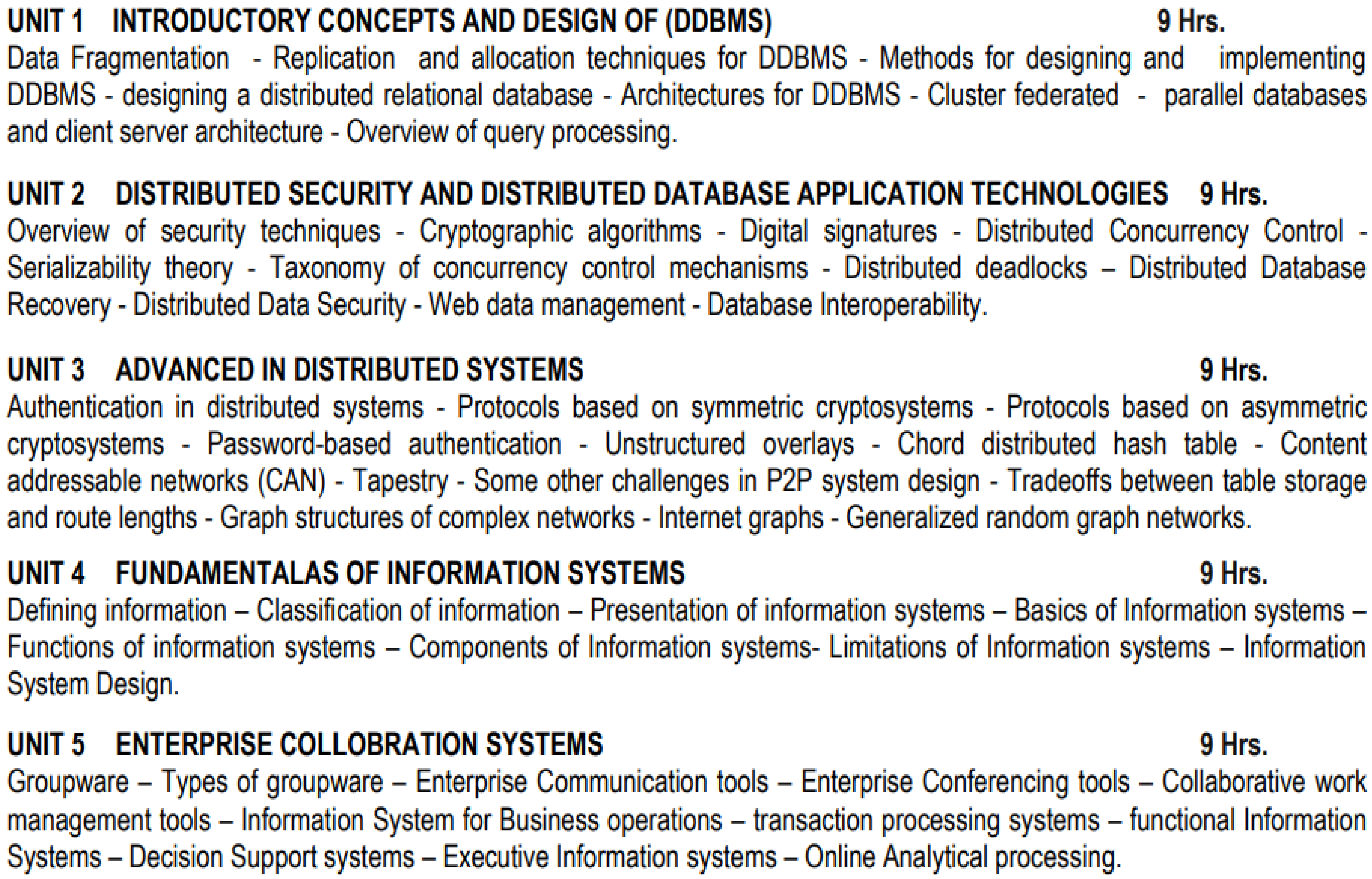
SATHYABAMA
INSTITUTE OF SCIENCE AND TECHNOLOGY
(DEEMED TO BE UNIVERSITY)
Accredited "A" Grade by NAAC | 12B Status by UGC | Approved by AICTE
www.sathyabama.ac.in

SCSA3008

DISTRIBUTED DATABASE AND INFORMATION SYSTEMS

Dr.R.Sathya Bama Krishna,
Associate Professor,
Dept.of CSE

SYLLABUS



UNIT 1 INTRODUCTORY CONCEPTS AND DESIGN OF (DDBMS) 9 Hrs.

Data Fragmentation - Replication and allocation techniques for DDBMS - Methods for designing and implementing DDBMS - designing a distributed relational database - Architectures for DDBMS - Cluster federated - parallel databases and client server architecture - Overview of query processing.

UNIT 2 DISTRIBUTED SECURITY AND DISTRIBUTED DATABASE APPLICATION TECHNOLOGIES 9 Hrs.

Overview of security techniques - Cryptographic algorithms - Digital signatures - Distributed Concurrency Control - Serializability theory - Taxonomy of concurrency control mechanisms - Distributed deadlocks – Distributed Database Recovery - Distributed Data Security - Web data management - Database Interoperability.

UNIT 3 ADVANCED IN DISTRIBUTED SYSTEMS 9 Hrs.

Authentication in distributed systems - Protocols based on symmetric cryptosystems - Protocols based on asymmetric cryptosystems - Password-based authentication - Unstructured overlays - Chord distributed hash table - Content addressable networks (CAN) - Tapestry - Some other challenges in P2P system design - Tradeoffs between table storage and route lengths - Graph structures of complex networks - Internet graphs - Generalized random graph networks.

UNIT 4 FUNDAMENTALAS OF INFORMATION SYSTEMS 9 Hrs.

Defining information – Classification of information – Presentation of information systems – Basics of Information systems – Functions of information systems – Components of Information systems- Limitations of Information systems – Information System Design.

UNIT 5 ENTERPRISE COLLOBRATION SYSTEMS 9 Hrs.

Groupware – Types of groupware – Enterprise Communication tools – Enterprise Conferencing tools – Collaborative work management tools – Information System for Business operations – transaction processing systems – functional Information Systems – Decision Support systems – Executive Information systems – Online Analytical processing.

Authentication in distributed systems

- A distributed system is susceptible to a variety of threats mounted by intruders as well as legitimate users of the system.
- Authentication is needed for both authorization and accounting functions.
- The entities in a distributed system, such as users, clients, servers and processes, are collectively referred to as principals.
- 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.

Authentication in distributed systems

- 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.
- In a distributed system, authentication is carried out using a protocol involving message exchanges and these protocols are termed **authentication protocols**.
- Authentication is **identification plus verification**.
- Identification is the procedure whereby an entity claims a certain identity, while verification is the procedure whereby that claim is checked.
- Authentication is a process of verifying that the principal's identity is as claimed. The correctness of authentication relies heavily on the verification procedure employed.

Authentication in distributed systems

There are three main types of authentication of interest in a distributed system:

message content authentication — verifying that the content of a message received is the same as when it was sent;

message origin authentication — verifying that the sender of a received message is the same one recorded in the sender field of the message; and

general identity authentication — verifying that the a principal’s identity is as claimed.

Authentication in distributed systems

message content authentication

is commonly handled by tagging a key-dependent *message authentication code* (MAC) onto a message before it is sent. Message integrity can be confirmed upon reception by recomputing the MAC and comparing it with the one attached.

message origin authentication

is a subcase of GIA. A successful general identity authentication results in a belief held by the authenticating principal (the verifier) that the authenticated principal (the claimant) possesses the claimed identity. Hence subsequent claimant actions are attributable to the claimed identity.

general identity authentication

is needed for both authorization and accounting functions.

Types of Principals

Hosts : These are addressable entities at the network level. A host is usually identified by its name (for example, a fully qualified domain name) or its network address (for example, an IP address).

Users : These entities are ultimately responsible for all system activities. Users initiate and are accountable for all system activities. Most access control and accounting functions are based on users. Typical users include humans, as well as accounts maintained in the user database. Users are considered to be outside the system boundary.

Processes : The system creates processes within the system boundary to represent users. A process requests and consumes resources on the behalf of its user.

A simple classification of authentication protocols

Authentication protocols can be categorized based on the type of :

cryptography (symmetric vs. asymmetric),
reciprocity of authentication (mutual vs. one-way),
key exchange,
real-time involvement of a third party (on-line vs. off-line),
nature of trust required from a third party,
nature of security guarantees, and storage of secrets.

Cryptographic Techniques

- Symmetric ("private key") and Asymmetric ("public key").
- Symmetric cryptography uses a single private key to both encrypt and decrypt data. Any party that has the key can use it to encrypt and decrypt data.
- Symmetric cryptography algorithms are typically fast and are suitable for processing large streams of data.
- Asymmetric cryptography, also called public-key cryptography, uses a secret key that must be kept from unauthorized users and a public key that is made public.
- Both the public key and the private key are mathematically linked: 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.
- Both keys are unique to a communication session

Notation

- We and define a system model that characterizes protocol executions.
- We assume a given set of constant symbols, which denote the names of principals, nonces, and keys.
- In symmetric key cryptography, 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.
- In asymmetric key cryptography, for a principal x, K_x and $K^{-1}{}_x$ denote its public and private keys, respectively.
- A communication step whereby P sends a message M to Q is represented as $P \rightarrow Q$: specify authentication protocols with precise syntax and semantics M, whereas a computation step of P is written as P: …, where "…".

A Login Protocol

For example, a typical login protocol between a host H and a user U is given in Algorithm (f denotes a one-way function, that is, given y, it is computationally infeasible to find an x such that f(x) = y).

$$
\begin{array}{rl}
U \rightarrow H: & U \\
H \rightarrow U: & \text{“Please enter password”} \\
U \rightarrow H: & p \\
H: & \text{compute } y = f(p) \\
: & \text{Retrieve user record } (U, f(\text{password}_U)) \text{ from the database} \\
: & \text{If y } = f(\text{password}_U)\text{, then accept; otherwise reject}
\end{array}
$$

Since authentication protocols for distributed systems directly use cryptosystems, their basic design principles also follow the type of cryptosystem used.

Protocols Based upon Symmetric Cryptosystems

In a symmetric cryptosystem, knowing the shared key lets a principal encrypt and decrypt arbitrary messages. Without such knowledge, a principal cannot create the encrypted version of a message, or decrypt an encrypted message.

Hence, authentication protocols can be designed according to the following principle called SYM:

"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

Using the above principle, we immediately obtain the basic protocol (shown in Algorithm) where principal P is authenticating itself to principal Q. “k” denotes a secret key that is shared between only P and Q.

$$
\begin{array}{rl}
P: & \text{Create a message } m = \text{“I am } P\text{.”} \\
: & \text{Compute } m' = \{m, Q\}_k \\
P \rightarrow Q: & m, m' \\
Q: & \text{verify } \{m, Q\}_k = m' \\
: & \text{if equal then accept; otherwise the authentication fails}
\end{array}
$$

In this protocol, the principal P prepares a message m, and encrypts the message and identity of Q using the symmetric key k and sends to Q both the plain text and encrypted messages. Principal Q, on receiving the message, encrypts the plain text 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

Basic Protocol - Weaknesses

- One 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 plain text message m and its encrypted version m are sent together by P to Q, this method is vulnerable to known plain text attacks.
- Thus the cryptosystem must be able to withstand known plain text attacks.

Modified protocol with nonce

To prevent replay attacks, we modify the protocol by adding a challenge-and response step using a **nonce**.
A nonce is a large random or pseudo-random number that is drawn from a large space so that it is difficult to guess by an intruder.
This property of a nonce helps ensure that old communications cannot be reused in replay attacks.

$$
\begin{array}{rl}
P \to Q: & \text{“I am } P\text{.”} \\
Q: & \text{generate nonce } n \\
Q \to P: & n \\
P: & \text{compute } m' = \{P, Q, n\}_k \\
P \to Q: & m' \\
Q: & \text{verify } \{P, Q, n\}_k = m' \\
: & \text{if equal then accept; otherwise the authentication fails}
\end{array}
$$

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

Modified protocol with nonce - Weaknesses

- This protocol has scalability problems because each principal must store the secret key for every other principal it would ever want to authenticate.
- This presents major initialization (the predistribution of secret keys) and storage problems.
- Moreover, the compromise of one principal can potentially compromise the entire system.
- This protocol is also vulnerable to known plain text attacks.

Wide-Mouth Frog Protocol

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

$$A \rightarrow S: \quad A, \{T_A, K_{AB}, B\}_{K_{AS}}$$

$$S \rightarrow B: \quad \{T_S, K_{AB}, A\}_{K_{BS}}$$

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.

Kerberos Authentication Service

Kerberos primarily addresses client–server authentication using a symmetric cryptosystem, together with trusted third-party authentication servers. (Project Athena)

The basic components include *authentication servers* (*Kerberos servers*) and *ticket-granting servers* (TGSs).

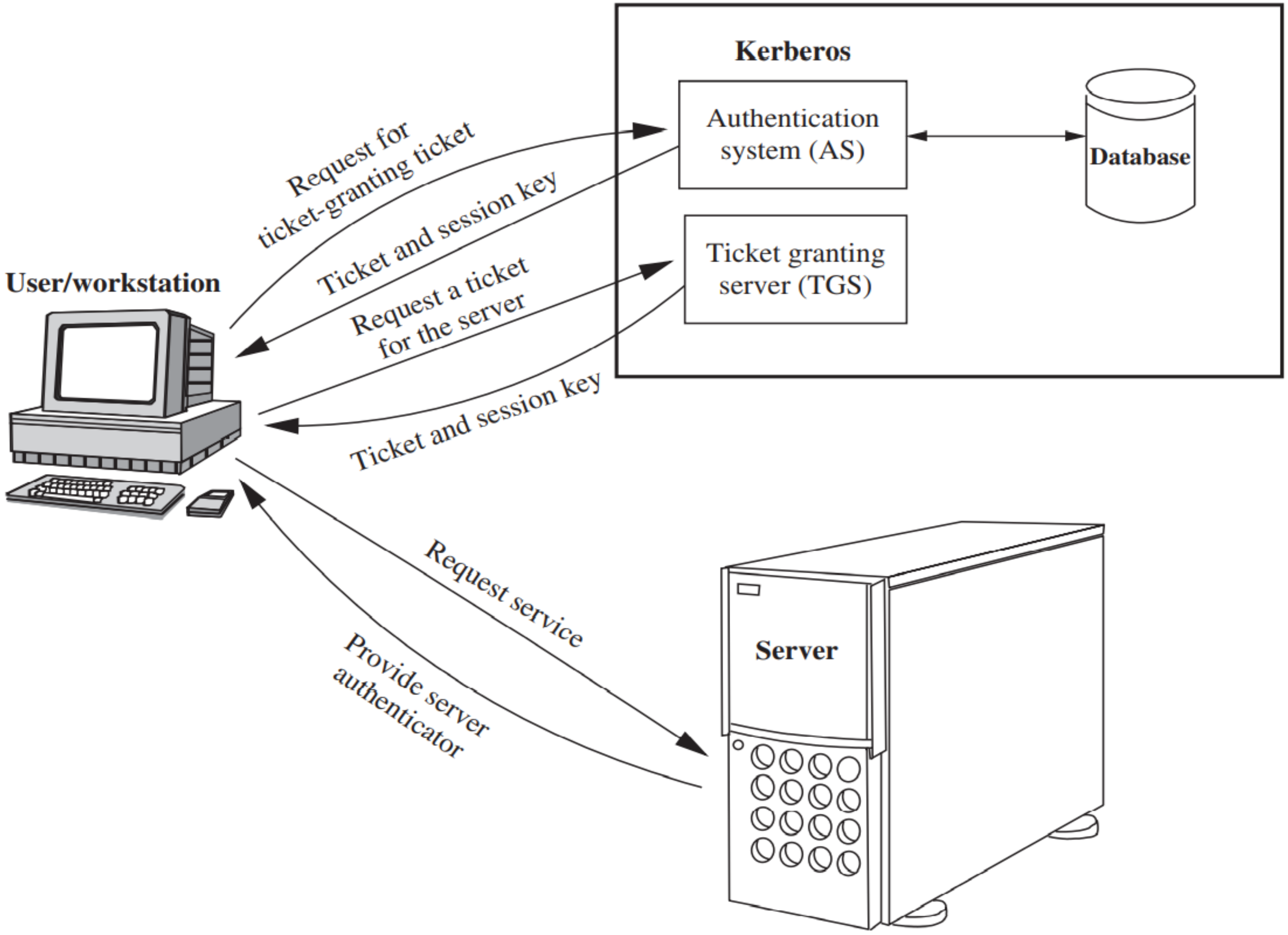
Initial Registration

- 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

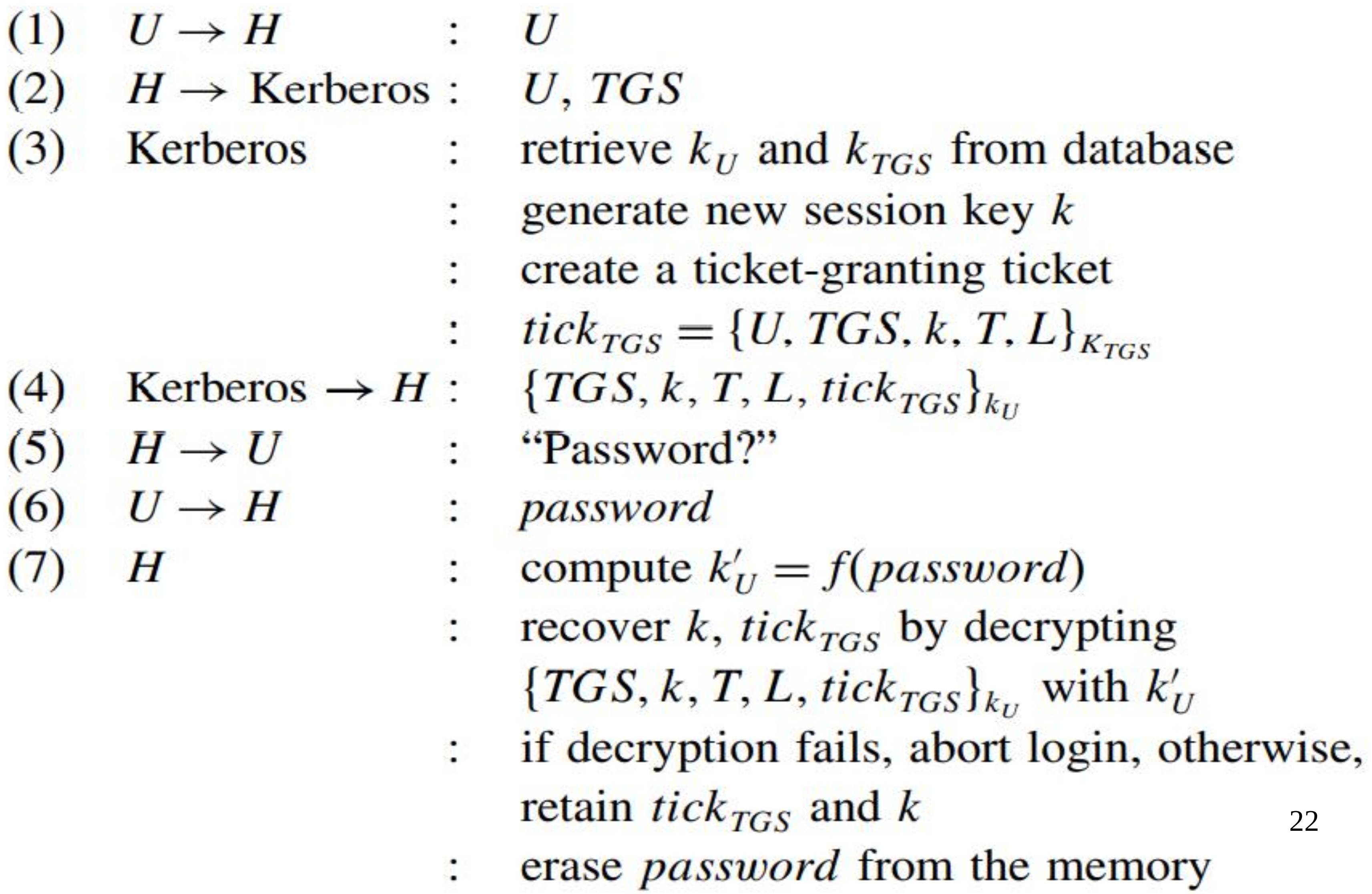
1. **Initial authentication at login** The Kerberos server authenticates user login at a host and installs a ticket for the ticket-granting server, TGS, at the login host.
2. **Obtain a ticket for the server** Using the ticket for the ticket-granting server, the client requests the ticket-granting server, TGS, for a ticket for the server.
3. **Requesting service from the server** The client uses the server ticket obtained from the TGS to request services from the server.

Steps in Authentication in Kerberos


Kerberos
Authentication system (AS)
Database
Ticket granting server (TGS)
User/workstation
Request for ticket-granting ticket
Ticket and session key
Request a ticket for the server
Ticket and session key
Request service
Provide server authenticator
Server

Initial Authentication at Login

Initial authentication at login uses a Kerberos server and is shown below. Let U be a user who is attempting to log into a host H.



(1)	$U \rightarrow H$	:	U
(2)	$H \rightarrow$ 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 $\rightarrow H$	:	$\{TGS, k, T, L, tick_{TGS}\}_{k_U}$
(5)	$H \rightarrow U$	:	“Password?”
(6)	$U \rightarrow H$	:	*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

Obtain a ticket for the server

The client executes the steps shown below to request a ticket for the server from the TGS. Basically, the client sends the ticket tickTGS to the TGS, requesting a ticket for the server S. (T1 and T2 are timestamps.)

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

Requesting service from the server

Client C sends the ticket and the authenticator to server. The server decrypts $tick_S$ and recovers k. It then uses k to decrypt the authenticator $\{C,T2\}_k$ 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

$$
\begin{array}{lrl}
(1) & C \rightarrow S: & tick_S, \{C, T_2\}_k \\
(2) & S: & \text{recover } k \text{ from } tick_S \text{ by decrypting it with } k_S \\
 & : & \text{recover } T_2 \text{ from } \{C, T_2\}_k \text{ by decrypting with } k \\
 & : & \text{check timeliness of } T_2 \text{ with respect to the local clock} \\
(3) & S \rightarrow C: & \{T_2 + 1\}_k
\end{array}
$$

Kerberos - Weaknesses

- Kerberos makes no provisions for host security; it assumes that it is running on trusted hosts with an untrusted network. If host security is compromised, then Kerberos is compromised as well.
- Kerberos uses a principal’s password (encryption key) as the fundamental proof of identity. If a user’s Kerberos password is stolen by an attacker, then the attacker can impersonate that user with impunity.
- In Kerberos version 4, authenticators are valid for a particular time. If an attacker sniffs the network for authenticators, they have a small time window in which they can re-use it and gain access to the same service.
- Kerberos version 5 introduced pre-authentication to solve this problem

Protocols based on Asymmetric Cryptosystems

In an asymmetric cryptosystem, each principal P publishes its public key k_p and keeps secret its private key k_p^{-1}. Thus only P can generate $\{m\}_{kp-1}$ for any message m by signing it using k_p^{-1}. The signed message $\{m\}_{kp-1}$ can be verified by any principal with the knowledge of k_p (assuming a commutative asymmetric cryptosystem).

Asymmetric authentication protocols can be constructed using a design principle called ASYM, which is as follows:
If a principal can correctly sign a message using the private key of the claimed identity, this act constitutes a sufficient proof of identity.

This ASYM principle follows the proof-by-knowledge principle for authentication, in that a principal's knowledge is indirectly demonstrated through its signing capability

Basic Protocol

Q sends a random number n to P and challenges it to encrypt with its private key. P encrypts (P Q n) with its private key k_p^{-1} and sends it to Q. Q verifies the received message by decrypting it with P's public key k_p and checking with the identity of P, Q, and n

$$
\begin{aligned}
P \rightarrow Q &: \quad \text{“I am } P\text{.”} \\
Q &: \quad \text{generate nonce } n \\
Q \rightarrow P &: \quad n \\
P &: \quad \text{compute } m = \{P, Q, n\}_{k_p^{-1}} \\
P \rightarrow Q &: \quad m \\
Q &: \quad \text{verify } (P, Q, n) = \{m\}_{k_p} \\
&: \quad \text{if equal, then accept; otherwise, the authentication fails}
\end{aligned}
$$

A modified protocol with a Certification Authority

The basic protocol requires that Q has the knowledge of P’s public key. A problem arises if Q does not know P’s public key. This problem is alleviated by postulating a centralized certification authority (CA) that maintains a database of all published public keys. If a user A does not have the public key of another user B, A can request B’s public key from the CA.

$$\begin{array}{rl}
P \rightarrow Q: & \text{“I am } P\text{.”} \\
Q: & \text{generate nonce } n \\
Q \rightarrow P: & n \\
P: & \text{compute } m = \{P, Q, n\}_{k_P^{-1}} \\
P \rightarrow Q: & m \\
Q \rightarrow CA: & \text{“I need } P\text{’s public key.”} \\
CA: & \text{retrieve public key } k_P \text{ of } P \text{ from key database} \\
 & \text{Create certificate } c = \{P, k_P\}_{k_{CA}^{-1}} \\
CA \rightarrow Q: & P, c \\
Q: & \text{recover } P, k_P \text{ from } c \text{ by decrypting with } k_{CA} \\
 & \text{verify } (P, Q, n) = \{m\}_{k_P} \\
: & \text{if equal, then accept; otherwise, the authentication fails}
\end{array}$$

SSL Protocol

- The secure sockets layer (SSL) protocol was developed by Netscape and is the standard Internet protocol for secure communications.
- The secure hypertext transfer protocol (HTTPS) is a communications protocol designed to transfer encrypted information between computers over the World Wide Web. HTTPS is http using a secure socket layer (SSL). SSL resides between TCP/IP and upper-layer applications, requiring no changes to the application layer.
- SSL is used typically between server and client to secure the connection. One advantage of SSL is that it is application protocol independent. A higher-level protocol can layer on top of the SSL protocol transparently.

SSL Protocol

SSL protocol allows client–server applications to communicate in a way so that eavesdropping, tampering, and message forgery are prevented.

The SSL protocol, in general, 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.

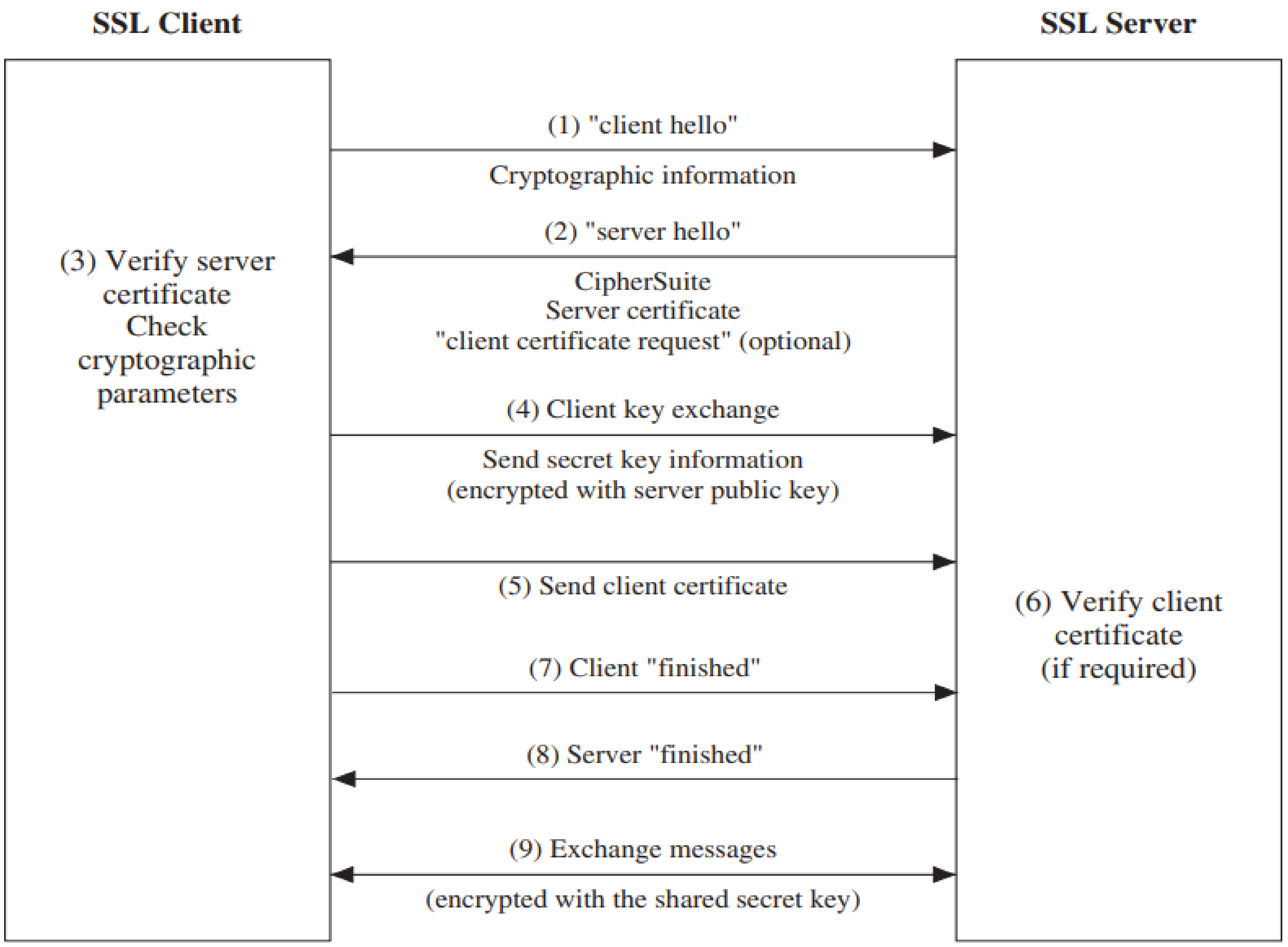
Confidentiality : Data is encrypted. A hacker cannot read your information by simply looking at the packets on the network.

SSL record Protocol

- The record protocol takes an application message to be transmitted, 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 to high-level users.

SSL handshake Protocol


SSL Client
SSL Server
(1) "client hello"
Cryptographic information
(2) "server hello"
CipherSuite
Server certificate
"client certificate request" (optional)
(3) Verify server certificate
Check cryptographic parameters
(4) Client key exchange
Send secret key information
(encrypted with server public key)
(5) Send client certificate
(6) Verify client certificate
(if required)
(7) Client "finished"
(8) Server "finished"
(9) Exchange messages
(encrypted with the shared secret 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 $\mathcal{X}$ issues the certificate to the SSL client, and CA $\mathcal{Y}$ 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.

Password-based Authentication

Preventing Off-line Dictionary Attacks:

A password-based authentication protocol aims at 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 subsequent messages for a secret session.

we present two password based authentication protocols :

- Encrypted key exchange (EKE) protocol.
- Secure remote password (SRP) protocol

Password-based Authentication : 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) $A \rightarrow B: \{C_A\}_{K_{AB}}$.

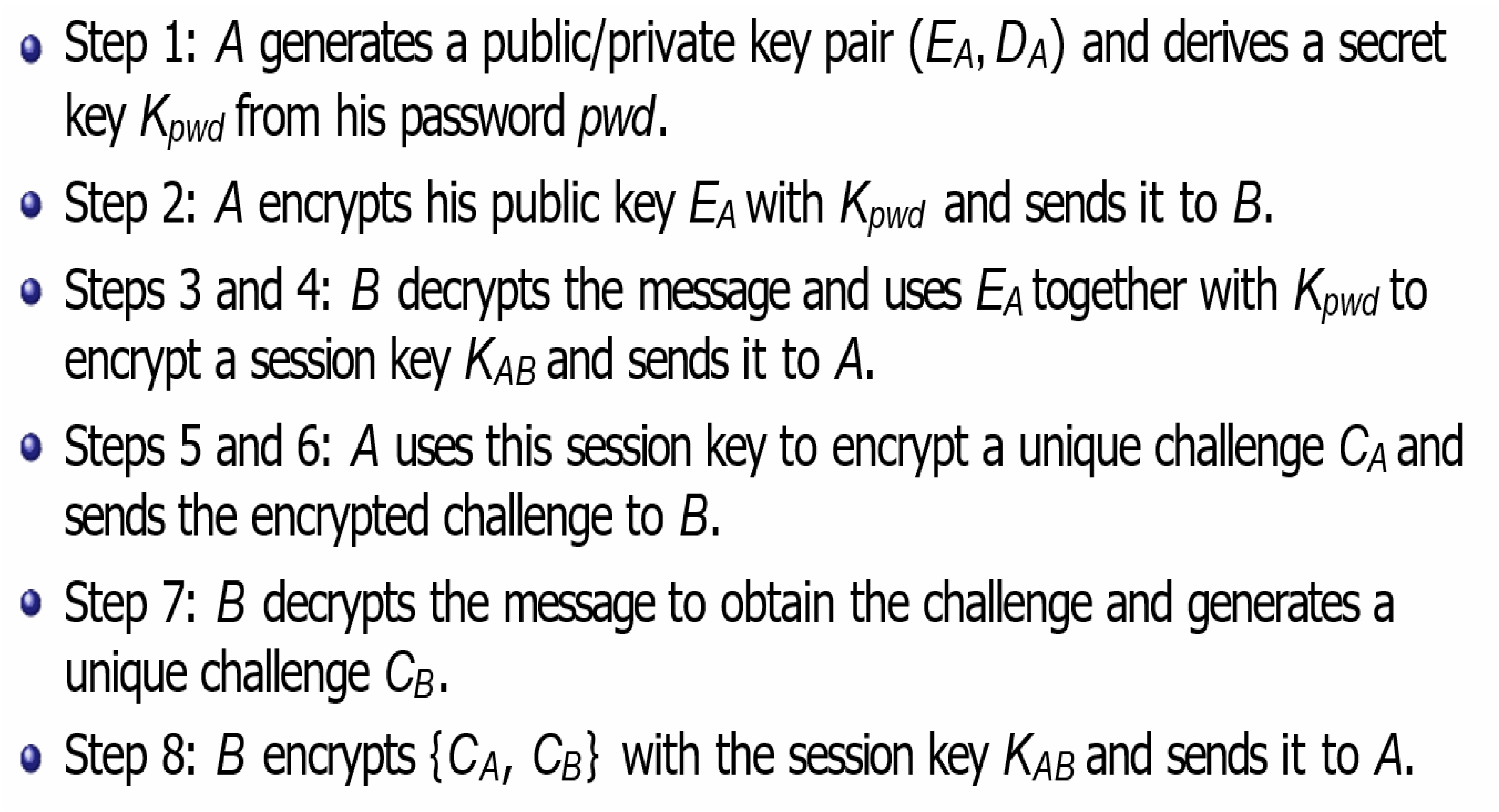
(7) $B:$ Compute $C_A = \{\{C_A\}_{K_{AB}}\}_{K_{AB}^{-1}}$ and generate a unique challenge C_B.

(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 own challenge. If they match, go to the next step, else abort.

(10) $A \rightarrow B:$ $\{C_B\}_{K_{AB}}$.

Password-based Authentication : 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*.

Password-based Authentication : 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).

Password-based Authentication : 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 F_n, where n is a large prime. Let g be a generator of F_n. Let A be a user and B be a server. Before initiating the SRP protocol, A and B do the following:

- A and B agree on the underlying field.
- 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.
- B stores the verifier v and the salt s.

Password-based Authentication : Secure remote password (SRP) protocol

(1) $A \rightarrow B: \quad A.$

(2) $B \rightarrow A: \quad s.$

(3) $A: \quad x := H(s, pwd); K_A := g^a.$

(4) $A \rightarrow B: \quad K_A.$

(5) $B: \quad K_B := v + g^b.$

(6) $B \rightarrow A: \quad K_B, r.$

(7) $A: \quad S := (K_B - g^x)^{a+rx}$ and $B: \quad S := (K_A v^r)^b.$

(8) $A, B: \quad K_{AB} := H(S).$

(9) $A \rightarrow B: \quad C_A := H(K_A, K_B, K_{AB}).$

(10) B verifies C_A and computes $C_B := H(K_A, C_A, K_{AB}).$

(11) $B \rightarrow A: \quad C_B.$

(12) A verifies C_B. Accept if verification passes; abort otherwise.

Password-based Authentication : 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 $K_{AB} = H(S)$.

Password-based Authentication : 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.