

Authentication in Distributed Systems



Rajiv Misra

Dept. of Computer Science &
Engineering

Indian Institute of Technology Patna

rajivm@iitp.ac.in

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.
- 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.”

Case Studies:

1. **KERBEROS PROTOCOL**
2. **SECURE SOCKET LAYER (SSL) PROTOCOL**

1. Kerberos Protocol

Kerberos Protocol

- **Kerberos primarily addresses client–server authentication** using a symmetric cryptosystem. Kerberos is an authentication system designed for **MIT's Project Athena**.
- The goal of **Project Athena** was to create an educational computing environment based on high-performance workstations, high-speed networking, and servers of various types.
- Researchers envisioned a **large-scale (10,000 workstations to 1,000 servers)** open network computing environment in which individual workstations could be privately owned and operated.
- Therefore, a workstation cannot be trusted to identify its users correctly to network services. Kerberos is not a complete authentication service required for secure distributed computing in general; it only addresses issues of client–server interactions.

Contd...

- Here, we will describe the **Kerberos authentication protocol**.
- Kerberos' design is based on the use of a **symmetric cryptosystem together with trusted third-party authentication servers**.
- The basic components include:
 - (i) Authentication Servers (Kerberos servers) and
 - (ii) Ticket-Granting Servers (TGSs).

Initial registration

- Every client/user registers with the Kerberos server by providing its **user i.d., 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.
- Note that **k_u is a secret key** that depends on the password of the user and is shared by **client U** and the **Kerberos server** only.

The Authentication protocol

Authentication in Kerberos proceeds in three steps:

- 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.

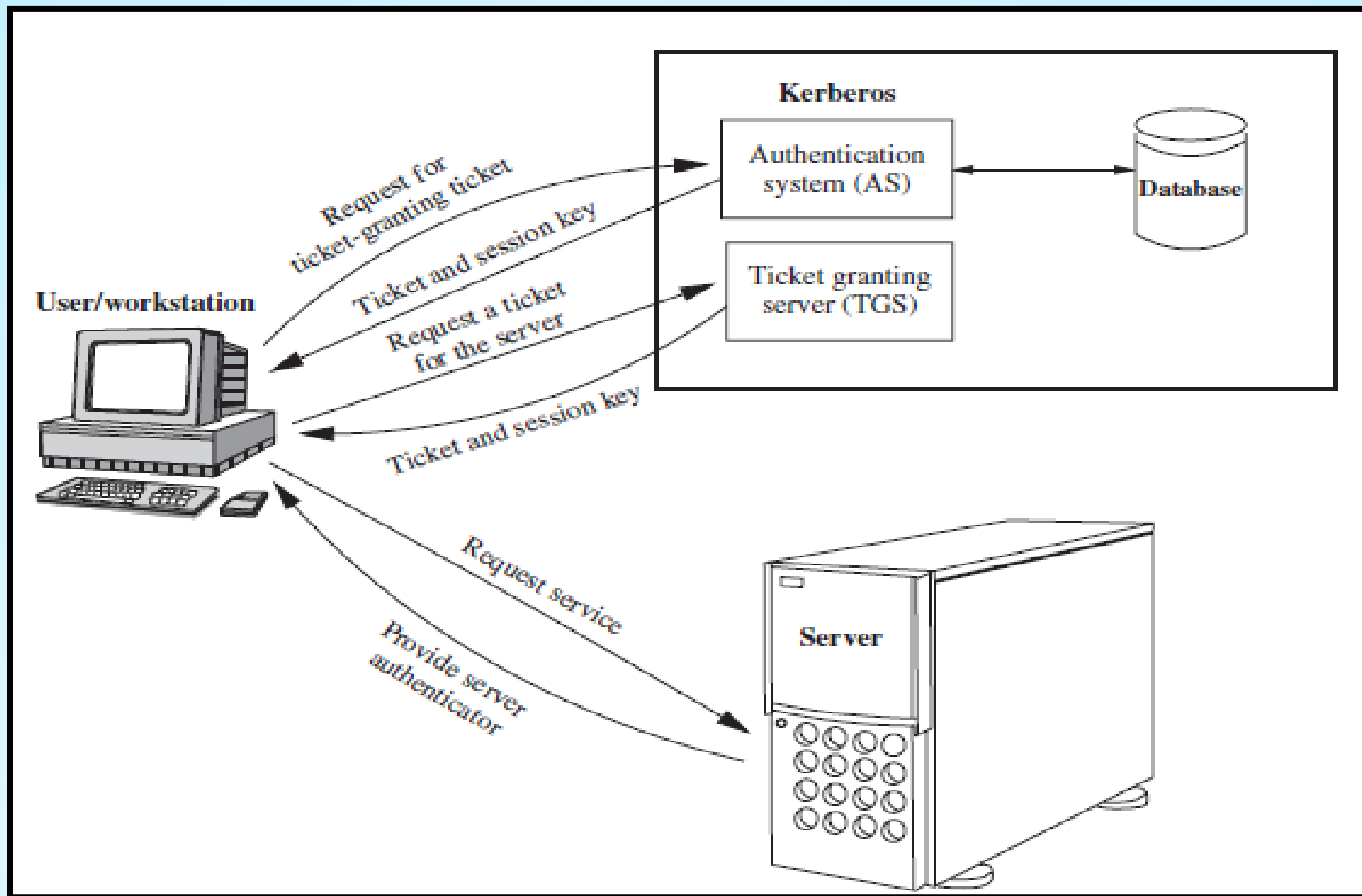


Figure 1. Steps in authentication in Kerberos.

Step 1: Initial authentication at login

- Initial authentication at login uses a Kerberos server and is shown in **Algorithm 1**. Let **U be a user** who is attempting to log into a **host H**.

(1)	$U \rightarrow H$	:	U
(2)	$H \rightarrow \text{Kerberos}$	:	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)	$\text{Kerberos} \rightarrow H$	:	$\{TGS, k, T, L, tick_{TGS}\}_{k_U}$
(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

Algorithm1 :Initial authentication at login

Contd...

- **In step 1, user U** initiates login by entering his/her username.
- **In step 2**, the login **host H** forwards the login request and the i.d. of the **TGS to a Kerberos server**.
- **In step 3**, the Kerberos server retrieves **k_U** and **k_{TGS}** from the database, generates a **new session key k** and creates a ticket-granting ticket **$ticket_{TGS} = \{U, TGS, k, T, L\}_{k_{TGS}}$** , where **U is the identity of the user** who wishes to communicate with the server, **TGS is the identity of the ticketgranting server**, **k is the session key**, **T is a timestamp**, **L is the ticket's lifetime** and **k_{TGS} is the key shared between the TGS and the Kerberos server**.

Contd...

- In step 4, the Kerberos server encrypts the ticket **tickTGS**, the identity of the **TGS**, the session key, the timestamp, and lifetime with **k_U** and sends it to **host H**.
- In step 5, on receiving this message from the Kerberos server, host H prompts the user for his/her password, which the user supplies in step 6. In step 7, host H computes the key, **K'_U** , corresponding to the password using the **one-way function f** .
- The host recovers the session key k by decrypting **$\{TGS, k, T, L, tickTGS\}_{k_U}$** with **$K'_U$** .
- If the password supplied by the user is not the valid password of **U** , **K'_U** would not be identical to **k_U** , and the authentication will fail.

Contd...

- Thus, the user is authenticated if the host is able to decrypt the message for the Kerberos server.
- Upon successful authentication, the host saves the new session **key k** and the ticket-granting ticket, **ticket_{TGS}**, for further use and erases the user password from the memory.
- The ticket-granting ticket is used to request server tickets from the TGS. Note **that ticket_{TGS}** is encrypted **with k_{TGS}**, the key shared between the TGS and the Kerberos server.

Step 2: Obtain a ticket for the server

- The client executes the steps shown in **Algorithm 2** to request a ticket for the server from the TGS. Basically, the client sends the ticket **tick_{TGS}** to the TGS, requesting 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
-

Algorithm2 :Obtain a ticket for the server

Contd...

- Because a ticket is susceptible to interception and replay, it does not by itself constitute sufficient proof of identity.
- For authentication, a principal presenting a ticket must also demonstrate the knowledge of the session key k named in the ticket. An authenticator, $\{C, T\}_k$, where **C is the client identity**, T is the timestamp, and k is the session key, provides the demonstration.

Contd...

- Unlike the ticket, which is reusable, an authenticator can be used only once and has a very short lifetime.
- The ticket proves the client's identity and also distributes the key; however, it is susceptible to replay attacks.
- The authenticator is used to counter this attack. Because an authenticator can be used only once and has a very short lifetime, the threat of an opponent stealing the ticket for a replay attack is countered.

Contd...

- In **step 1**, to request a ticket for server S, client C presents its ticket-granting ticket **ticket_{TGS}** along with the authenticator to the TGS. C's knowledge of k is demonstrated using the authenticator **$\{C, T_1\}_k$** .
- In **step 2**, the TGS decrypts **ticket_{TGS}** with **k_{TGS}** to recover k, verifies the authenticity of the authenticator by decrypting CT_1k with k, and checks the timeliness of T_1 in the authenticator and T in **ticket_{TGS}**.
- If both decryptations in step 2 are successful and T_1 is timely, the TGS is convinced of the authenticity of the ticket, and creates a ticket **$ticks = \{C, S, k, T, Lk_s\}$** for server S, where C is the identity of the client, S is the server identity, k is the new session key, T is the timestamp of the TGS, L is the lifetime of the ticket, and k_s is the key shared between the TGS and server S.
- This ticket is returned to C in step 3. In step 4, C recovers k and ticks from **$\{S, k, T, L, ticks\}_k$** by decrypting it with k.

Step 3: Requesting service from the server

- Client C sends the ticket and the authenticator to server. The server decrypts ticks 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 ticks before granting service to the client. If mutual authentication is required, the server returns an authenticator (**Algorithm 3**).

(1) $C \rightarrow S : tick_S, \{C, T_2\}_k$
(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) $S \rightarrow C : \{T_2 + 1\}_k$

Algorithm 3 Requesting service from the server.

Contd...

- In **step 1**, C presents S with ticks and a new authenticator. In step 2, S recovers k from ticks by decrypting it with k_s and uses k obtained to decrypt $\{C, T_2\}_k$.
- If both decryptions are successful and T_2 is timely, then S is assured of the authenticity of the Client.
- Finally, **step 3** assures C of the server's identity.

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.
- Since the Kerberos' password database holds all the passwords for all of the principals in a realm, if the host security on the database is compromised, then the entire realm is compromised.

Contd...

- 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 a replay cache** that prevents any authenticator from being used more than once.
- Since anybody can request a ticket-granting ticket for any user, and that ticket is encrypted with the user's secret key (password), it is simple to perform an offline attack on this ticket by trying to decrypt it, say using the dictionary attack.
- **Kerberos version 5 introduced pre-authentication** to solve this problem.

2. Secure Sockets Layer (SSL) Protocol

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.

Features

- 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

- 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 or receives its first byte of data.

Steps for SSL Handshake Protocol

The following steps, shown in **Figure 2**, are involved in the SSL handshake:

1. 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 that is used in subsequent computations.
2. 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” that includes a list of the types of certificates supported and the distinguished names of acceptable **certification authorities (CAs)**.

Contd...

3. 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.
4. The SSL client, using all data generated in the handshake so far, **create 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. The premaster secret itself is encrypted with the server's public key.
5. **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." This alert is only a warning, but with some implementations the handshake fails if client authentication is mandatory.

Contd...

6. The **SSL server verifies the signature** on the client certificate.
7. 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.
8. 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.
9. For the duration of the **SSL session**, the SSL server and SSL client can now exchange messages that are encrypted with the shared symmetric secret key.

SSL Client

SSL Server

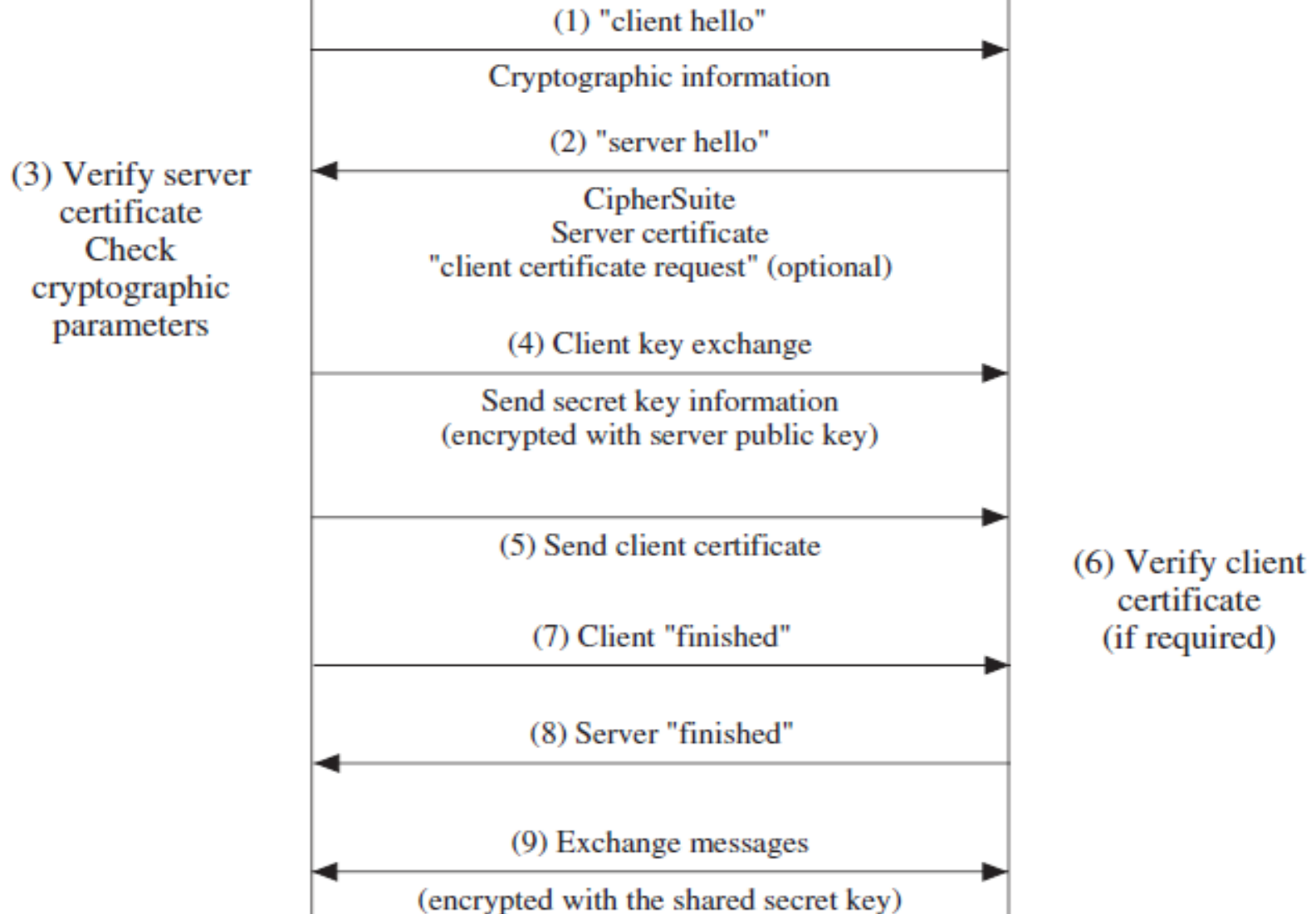


Figure 2: SSL Handshake Protocol and Data Exchange

How SSL provides authentication

- **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.
- **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 that are encrypted with the secret key (steps 7 and 8 in the overview) confirms that authentication is complete.

Contd...

- 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. The certificates required are as follows, where CA X issues the certificate to the SSL client, and CA Y issues the certificate to the SSL server:
 - For server authentication only, the SSL server needs the following:
 - The personal certificate issued to the server by CA Y .
 - The server's private key.

Contd...

- **The SSL client needs:**

- The CA certificate for CA Y or the personal certificate issued to the server by CA Y .
- If the SSL server requires client authentication, the server verifies the client's identity by verifying the client's digital certificate with the public key for the CA that issued the personal certificate to the client, in this case CA X. For both server and client authentication, the SSL server needs:
 - The personal certificate issued to the server by CA Y .
 - The server's private key.
 - The CA certificate for CA X or the personal certificate issued to the client by CA X.

Contd...

- **The SSL client needs:**

- The personal certificate issued to the client by CA X.
- The client's private key.
- The CA certificate for CA Y or the personal certificate issued to the server by CA Y .
- Both the SSL server and the SSL client might need other CA certificates to form a certificate chain to the root CA certificate.

Conclusion

- **Authentication is a process by which one principal verifies the identity of other principal.** For example, in a client–server system, the client and the server may need to verify each other's identity to assure that each is talking to the right entity.
- Generally, authentication is based on the possession of a secret information, like password, that is known only to the entities participating in the authentication. For a successful authentication, the entity must demonstrate the knowledge of the right secret information.
- In this lecture, we have described the authentication protocols such as **Kerberos and Secure Socket Layer (SSL) protocol.**