

UNIT 8

Secure Socket

(4 Hours) — Network Programming (CAC355)

Disclaimer: These notes were generated automatically from the CACS355 syllabus and may contain errors or omissions. Please verify important details (syntax, method names, default values) against your textbook (Java Network Programming, E. R. Harold) and class notes before the exam.

8.1 Secure Communication

Data sent over an ordinary socket travels as plain text and can be read (**eavesdropping**), altered (**tampering**) or redirected to a fake server (**impersonation**). Secure communication protects against all three. In Java this is provided by the **Java Secure Socket Extension (JSSE)** in `javax.net.ssl`, which implements **SSL / TLS**.

Goal	How it is achieved
Confidentiality	Data is encrypted with a symmetric key (e.g., AES) negotiated during the handshake
Integrity	A message authentication code (MAC/AEAD) detects modification
Authentication	The server (and optionally the client) proves identity with an X.509 certificate signed by a trusted CA

How SSL/TLS works (simplified handshake)

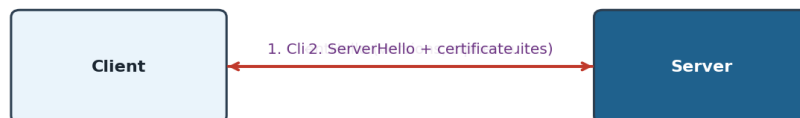


Figure 8.1 — Start of the TLS handshake

- Client sends supported protocol versions and **cipher suites**; server picks one and sends its **certificate**.
- Client verifies the certificate (trusted CA, valid dates, matching host name).
- Both sides agree on **session keys** (using key exchange such as ECDHE).

- Application data is then sent encrypted and integrity-protected.

*Note: A **cipher suite** names the algorithms used, e.g., `TLS_AES_128_GCM_SHA256` (TLS 1.3) or `TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256` (TLS 1.2). “SSL” is the old name; SSLv3 and earlier are insecure — use TLS 1.2 or 1.3.*

8.2 Creating Secure Client Sockets

Use `SSLSocketFactory` instead of the `Socket` constructor. The default factory uses the JVM’s trust store (cacerts), so certificates from well-known CAs are accepted automatically. The resulting `SSLSocket` extends `Socket`, so the streams are used exactly as before.

```
SSLSocketFactory f = (SSLSocketFactory) SSLSocketFactory.getDefault();
try (SSLSocket s = (SSLSocket) f.createSocket("www.example.com", 443)) {
    Writer out = new OutputStreamWriter(s.getOutputStream(), "UTF-8");
    out.write("GET / HTTP/1.1\r\nHost: www.example.com\r\nConnection: close\r\n\r\n");
    out.flush();
    BufferedReader in = new BufferedReader(new InputStreamReader(s.getInputStream(), "UTF-8"));
    String line;
    while ((line = in.readLine()) != null) System.out.println(line);
}
```

Listing 8.1 — A minimal HTTPS client using `SSLSocket`

- **`createSocket(host, port)`** connects and (lazily) performs the handshake on first read/write or when `startHandshake()` is called.
- **`createSocket(Socket existing, host, port, autoClose)`** layers SSL on top of an already connected socket (e.g., after connecting through a proxy).
- For self-signed or private CAs, provide your own trust store through an **`SSLContext`** with a **`TrustManager`** or the property `javax.net.ssl.trustStore`.
- Never disable certificate or host-name checking in production code.

8.3 Event Handlers

The handshake is asynchronous from the program’s point of view, so JSSE lets you register a `HandshakeCompletedListener` with `socket.addHandshakeCompletedListener()`. Its `handshakeCompleted(HandshakeCompletedEvent e)` method is called when a handshake finishes.

HandshakeCompletedEvent method	Returns
<code>getSession()</code>	The <code>SSLSession</code> that was created
<code>getCipherSuite()</code>	Negotiated cipher suite

HandshakeCompletedEvent method	Returns
getPeerCertificates()	Certificate chain of the peer
getSocket()	The SSLSocket involved

8.4 Session Management

Full handshakes are expensive, so SSL supports **sessions**: after the first handshake, later connections between the same two hosts can **resume** the session and skip the costly key exchange. Each session is an `SSLSession` object.

SSLSession method	Meaning
getId()	Session identifier (byte array)
getCipherSuite() / getProtocol()	Negotiated cipher suite and protocol version
getPeerHost() / getPeerPort()	The other party
getPeerCertificates()	Peer's certificates
getCreationTime() / getLastAccessedTime()	Timestamps
isValid() / invalidate()	Check or end the session so it can't be resumed

`socket.setEnabledSessionCreation(false)` allows only existing sessions to be resumed and refuses new ones. `socket.getSession()` returns the current session (forcing a handshake if needed).

8.5 Client Mode

SSL sockets have a **mode**: in **client mode** the socket starts the handshake as the client (initiates it) and, by default, authenticates the *server* only. In **server mode** it answers the handshake. Which mode is used does not depend on who opened the TCP connection, and can be changed before the handshake with `setUseClientMode(boolean)` and read with `getUseClientMode()`. This is occasionally useful, e.g., for protocols such as FTP where the TCP roles and SSL roles are reversed for the data channel.

8.6 Creating Secure Server Sockets

A secure server uses `SSLServerSocketFactory` and produces `SSLServerSocket` objects; `accept()` returns an `SSLSocket`. Unlike clients, a secure server **must have a certificate and private key**, kept in a **keystore**.

Steps

- Create a keystore with a key pair and certificate: `keytool -genkeypair -alias server -keyalg RSA -keystore server.jks`.
- Load it into a **KeyStore**; initialise a **KeyManagerFactory** with the keystore and password.
- Create an **SSLContext** (`SSLContext.getInstance("TLS")`) and call `init(keyManagers, null, null)`.
- Get the **SSLServerSocketFactory** from the context and create the **SSLServerSocket**.
- Call `accept()` as usual; each accepted **SSLSocket** is a secure connection.

```
KeyStore ks = KeyStore.getInstance("JKS");
ks.load(new FileInputStream("server.jks"), pw);
KeyManagerFactory kmf = KeyManagerFactory.getInstance("SunX509");
kmf.init(ks, pw);
SSLContext ctx = SSLContext.getInstance("TLS");
ctx.init(kmf.getKeyManagers(), null, null);
SSLServerSocket ss = (SSLServerSocket) ctx.getServerSocketFactory().createServerSocket(8443);
SSLSocket c = (SSLSocket) ss.accept();
```

Listing 8.2 — Secure server setup

The JVM can also load the keystore using system properties `javax.net.ssl.keyStore` and `javax.net.ssl.keyStorePassword`.

8.7 Configuring SSLServerSockets

Choosing the cipher suites

Not all supported suites are enabled or safe. `getSupportedCipherSuites()` lists everything the implementation knows; `getEnabledCipherSuites()` lists those in use; `setEnabledCipherSuites(String[])` chooses the ones you allow. Remove weak suites (NULL, EXPORT, RC4, DES, anonymous) and prefer forward-secrecy suites (ECDHE). Protocols are selected similarly with `setEnabledProtocols()` (e.g., TLSv1.2 and TLSv1.3 only).

Session management

`setEnabledSessionCreation(boolean)` decides whether accepted sockets may create new sessions (false = resume only). Other session settings such as timeouts are set through the `SSLSessionContext` of the `SSLContext` (`setSessionTimeout`, `setSessionCacheSize`).

Client mode (client authentication)

By default a server does not ask for a client certificate. **setNeedClientAuth(true)** makes it *require* one (handshake fails without it); **setWantClientAuth(true)** requests it but continues without. **setUseClientMode(false)** is the default for server sockets. Mutual (two-way) authentication is used for high-security services.

Method	Purpose
setEnabledCipherSuites(String[])	Choose allowed cipher suites
setEnabledProtocols(String[])	Choose allowed protocol versions
setEnabledSessionCreation(boolean)	Allow or forbid new sessions
setNeedClientAuth(boolean) / setWantClientAuth(boolean)	Require / request a client certificate
setUseClientMode(boolean)	Set client or server role in handshake

Key Takeaways

- ✓ SSL/TLS gives confidentiality, integrity and authentication; JSSE (javax.net.ssl) implements it.
- ✓ Clients: `SSLContext.getDefault().createSocket(host, port)` gives an `SSLSocket`.
- ✓ `HandshakeCompletedListener` reports finished handshakes; `SSLSession` stores session details and allows resumption.
- ✓ Client mode vs server mode decides who acts as the handshake initiator.
- ✓ Secure servers need a keystore (certificate + private key), `SSLContext`, and `SSLServerSocketFactory`.
- ✓ Configure enabled cipher suites, protocols, session creation and client authentication (`needClientAuth/wantClientAuth`).

— End of Unit 8 —