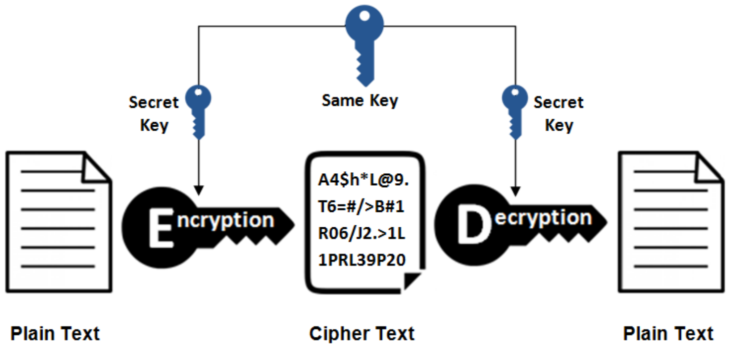
Unit-8

Secure Communication:

Symmetric Encryption



- Symmetric encryption is an old and best-known technique.
- It uses a **secret key** that can either be a number, a word or a string of random letters.
- It is a combined with the **plain text** of a message to change the content in a particular way.
- The **sender** and the **recipient** should know the **secret key** that is used to encrypt and decrypt all the messages. Blowfish, AES, RC4, DES, RC5, and RC6 are examples of symmetric encryption.
- The most widely used symmetric algorithm is **AES-128, AES-192**, and **AES-256**.

How It Works

1. **Key Generation**: A secret key is generated. This key will be used for both encryption and decryption.
2. **Encryption**:
   - The sender uses the secret key to encrypt the message.
   - The encrypted message (ciphertext) is created and sent to the recipient.
3. **Decryption**:
   - The recipient uses the same secret key to decrypt the ciphertext back into the original message (plaintext).

Example

Imagine Person1 wants to send a confidential message to Person2(bob):

1. **Key Generation**:
   - **Person1 and Person2** agree on a secret key: `mySecretKey`.
2. **Encryption**:
   - **Person1** uses the secret key to encrypt her message: "Hello, Person2!".
   - The encrypted message might look like: `A1b2C3d4`.
3. **Decryption**:

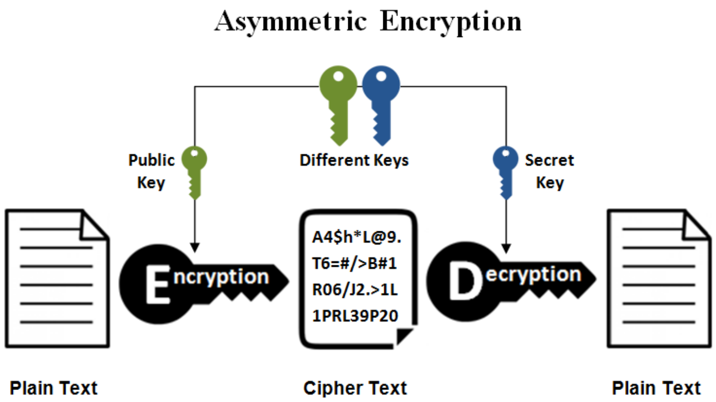
    - **Person2** receives the encrypted message.
    - **Person2** uses the same secret key to decrypt the message back to: "Hello, Person2!".

Advantages:

- **Speed**: Symmetric encryption is generally faster than asymmetric encryption, making it suitable for encrypting large amounts of data.
- **Simplicity**: It is straightforward to implement and use.

Uses

- **Data Storage**: Encrypting files or databases to protect sensitive information.
- **Network Security**: Protecting data transmitted over networks
- **Secure Communications**: Protocols like SSL/TLS use symmetric encryption for the actual data transfer after establishing a secure connection using asymmetric encryption.



- **Asymmetric encryption** is also known as **public key cryptography**, which is a relatively new method, compared to symmetric encryption.

Asymmetric encryption, also known as public-key cryptography, uses a pair of keys to encrypt and decrypt data. These keys are:

1. **Public Key**: This key can be shared with anyone. It is used to encrypt data.
2. **Private Key**: This key is kept secret by the owner. It is used to decrypt data encrypted with the corresponding public key.

How It Works

1. **Key Generation**: A pair of keys (public and private) is generated. These keys are mathematically linked, but it is practically impossible to derive the private key from the public key.
2. **Encryption**:

    - The sender obtains the recipient's public key.
    - The sender uses this public key to encrypt the message.
    - The encrypted message is sent to the recipient.
3. **Decryption**:
    - The recipient receives the encrypted message.
    - The recipient uses their private key to decrypt the message.

Example

Imagine Person1 wants to send a confidential message to Person2:

1. **Key Generation**:
    - **Person1** generates a **public key** and a **private key**.
    - **Person1** shares his **public key** with **Person2** but keeps his **private key** secure.
2. **Encryption**:
    - **Person2** uses **Person1's** public key to encrypt her message: "Hello, Person1!".
    - The encrypted message is something like: `7dh34nN&*d8d`
3. **Decryption**:
    - **Person1** receives the encrypted message.
    - **Person1** uses his private key to decrypt the message back to: "Hello, Person1!".

Why Use Asymmetric Encryption?

- **Security**: Even if someone intercepts the encrypted message, they cannot decrypt it without the private key.
- **Key Distribution**: Public keys can be freely distributed, making it easy to securely send encrypted messages to anyone who shares their public key.

Uses

- **Secure Communication**: **Email encryption**.
- **Digital Signatures**: Verifying the authenticity and integrity of a **message or document**.
- **SSL/TLS**: Securing connections between **web browsers** and **servers**.

Secure Socket:

- The Java Secure Socket Extension (JSSE) **can secure network communication** using **SSL (Secure Socket Layer)** and **TLS (Transport Layer Security)** protocol.
- Using JSSE, developers can provide for the secure passage of data between a **client** and a **server** running any application protocol (such as HTTP, Telnet, or FTP) over TCP/IP
- **Secure Sockets Layer (SSL)** is a **standard security technology** for establishing an **encrypted link** between a server and a client—*typically a web server (website) and a browser.*

- **Transport Layer Security (TLS) encrypts data** and sent over the Internet to ensure that **hackers** are unable to see what you transmit which is particularly useful for **private and sensitive information** such as passwords, credit card numbers, and personal correspondence.

The Java Secure Socket Extension is divided into four packages:

javax.net.ssl

The **abstract classes** that define Java's API for secure network communication.

javax.net

The **abstract socket factory classes** used instead of constructors to create secure sockets.

java.security.cert

The classes for handling the **public-key certificates** needed for SSL.

com.sun.net.ssl

The concrete classes that implement the **encryption algorithms and protocols** in Sun's reference implementation of the JSSE

Creating Secure Client Socket:

```
SocketFactory factory = SSLSocketFactory.getDefault();
Socket socket = factory.createSocket("login.ibiblio.org", 7000)
```

Five overloaded **createSocket()** methods to build an **SSLSocket:**

```
1. public abstract Socket createSocket(String host, int port) throws
   IOException, UnknownHostException
2. public abstract Socket createSocket(InetAddress host, int port) throws
   IOException
3. public abstract Socket createSocket(String host, int port,InetAddress
   interface, int localPort)throws IOException, UnknownHostException
4. public abstract Socket createSocket(InetAddress host, int port,InetAddress
   interface, int localPort)throws IOException, UnknownHostException
5. public abstract Socket createSocket(Socket proxy, String host, int
   port,boolean autoClose) throws IOException
```

Step to creating Secure Client Socket

1. Import javax.net.ssl.* to add SSL support:

```
import javax.net.ssl.*;
```

2. Declare a SocketFactory by using SSLSocketFactory to initialize it:

   ```
   SocketFactory newSF = SSLSocketFactory.getDefault();
   ```

3. Use your new SocketFactory to initialize your sockets the same way that you used your old SocketFactory

   ```
   Socket s = newSF.createSocket(("login.ibiblio.org", serverPort);
   ```

Example:

```
import javax.net.ssl.*;
import java.net.*;
public class App {
     public static void main(String[] args) throws Exception {
    SSLSocketFactory factory= (SSLSocketFactory)
SSLSocketFactory.getDefault();
    Socket socket = factory.createSocket("127.0.0.1", 7000);
    System.out.println("Server Connected"+socket);//connected
    //Code to read and write operation with Server
    socket.close();
    }
}
```

creating Secure Server Socket

```
import javax.net.ssl.*;
import java.net.*;
public class App {
    public static void main(String[] args) throws Exception
{
    ServerSocket serverSocket=
((SSLServerSocketFactory)SSLServerSocketFactory.getDefault()).c
reateServerSocket(7000);
    Socket socket = serverSocket.accept();
    System.out.println(socket+"Client Accepted");//connected
    //Code to read and write operation with Server
____socket.close();
    }
}
```

Configuring SSLServerSockets:

SSLSocket and **SSLServerSocket** are both classes in the Java Secure Socket Extension (JSSE) API

SSLSocket:

- Represents a client-side secure socket
- Used by a client to connect to an SSL/TLS server
- Establishes a secure connection to a remote server.

SSLServerSocket:

- Represents a server-side secure socket.
- Used by a server to listen for and accept incoming SSL/TLS connections from clients.
- Listens for incoming connection requests on a specified port.
- listens for and accepts connections.
-

Choosing the Cipher Suites

The **SSLServerSocket** class has the same three methods for determining which cipher Suites(A **cipher suite** is a set of algorithms that help secure a network connection) are supported and enabled as SSLSocket does:

public abstract String[] getSupportedCipherSuites()

Retrieve the list of all cipher suites supported by the socket.

public abstract String[] getEnabledCipherSuites()

Choose a secure cipher suites from the supported list based on best practices.

public abstract void setEnabledCipherSuites(String[] suites)

Configure the socket to use only the selected secure cipher suites.

Example:

```
import javax.net.ssl.*;
public class Test {
    public static void main(String[] args) throws Exception{
        SSLSocketFactory sslSocketFactory = (SSLSocketFactory)
SSLSocketFactory.getDefault();
       // Create an SSLSocket with the target host and port
       SSLSocket sslSocket = (SSLSocket)
sslSocketFactory.createSocket("www.tufohss.edu.np", 443);
       // Print enabled and supported cipher suites
       System.out.println("Enabled Cipher Suites: ");
       for (String suite : sslSocket.getEnabledCipherSuites()) {
           System.out.println(suite);
       }
       System.out.println("Supported Cipher Suites: ");
       for (String suite : sslSocket.getSupportedCipherSuites()) {
           System.out.println(suite);
       }
       sslSocket.close();
    }
}
```

Session Management

In a client-server setup, a "session" is a way to keep track of interactions between the client and server. It allows the server to remember things about the client across multiple requests.

- **setEnableSessionCreation(boolean allowSessions)** is a method used by the server to decide whether it will allow sessions to be created or not. If you pass **true**, the server will allow sessions. If you pass **false**, it will not.
- **getEnableSessionCreation()** is a method used to check whether session creation is currently allowed by the server. It returns **true** if sessions can be created, and **false** otherwise

Client Mode

Client Authentication

The S**SLServerSocket** class has **two methods** for determining and specifying whether client sockets are required to authenticate themselves to the server. By passing true to the **setNeedClientAuth()** method.

public abstract void setNeedClientAuth(boolean flag)

This method is used by the server to decide if it requires clients to prove their identity (authenticate themselves) when connecting. If you set **flag** to **true**, clients must authenticate themselves. If you set it to **false**, they don’t need to.

public abstract boolean getNeedClientAuth()

This method checks if the server is set up to require client authentication. It returns **true** if client authentication is required, and **false** if it's not.

Client Mode

The **setUseClientMode()** method allows a program to indicate that even though it has created an **SSLServerSocket**, it is and should be treated as a client in the communication.

public abstract void setUseClientMode(boolean flag)

This method tells the server-side SSL socket to act as a client in communication, even though it's created as a server socket. If you set **flag** to **true**, the socket behaves like a **client**. If **flag** is **false**, it behaves as a **server**.

public abstract boolean getUseClientMode()

This method checks if the server-side SSL socket is currently set to act as a client. It returns **true** if it's in client mode, and **false** if it's not.

setNeedClientAuth and **getNeedClientAuth** control whether clients must prove who they are, while **setUseClientMode** and **getUseClientMode** control whether the server socket should act like a client in the communication.

Event Handlers

- Network communications are slow compared to the speed of most computers.
- Authenticated network communications are even slower.
- The necessary key generation and setup for a secure connection can easily take several second.

In order to get notifications of handshake-complete events, simply implement the

HandshakeCompletedListener Interface

- **HandshakeCompletedListener**: This is an interface that you can use to listen for and respond to SSL/TLS handshake events. An SSL/TLS handshake is the process where the client and server establish a secure connection.
- **handshakeCompleted(HandshakeCompletedEvent event)**: This method is called when the SSL/TLS handshake is finished. You can use this method to handle what happens once the secure connection is established.

HandshakeCompletedEvent Class

- **HandshakeCompletedEvent**: This is an object that contains details about the handshake event. When the handshakeCompleted method is called, it's given a HandshakeCompletedEvent object which contains the following information:

1. **getSession()**: This method returns the SSL session associated with the completed handshake. The SSL session contains information about the secure connection.

2. **getCipherSuite()**: This method returns a string that describes the encryption algorithm and protocol used for the secure connection. It tells you what kind of encryption was used.
3. **getPeerCertificateChain()**: This method returns an array of certificates used by the peer (the other side of the connection). This can help you verify the identity of the peer. Note: This method might throw an exception if the peer's identity isn't verified.
4. **getSocket()**: This method returns the socket through which the handshake event occurred. A socket is like a communication endpoint for the connection.