# Network Programming

[CACS355]

# BCA 6th Sem

Er. Sital Prasad Mandal

(Email : info.sitalmandal@gmail.com)

Bhadrapur, Jhapa, Nepal

https://networkprogam-mmc.blogspot.com/

https://networkprogam-mmc.blogspot.com/

# Unit-8

# Secure Sockets

| | |
|---|---|
| 1 | Secure Communications |
| 2 | Creating Secure Client Sockets |
| 3 | Choosing the Cipher Suites |
| 4 | Event Handlers |
| 5 | Session Management |
| 6 | Client Mode |
| 7 | Creating Secure Server Sockets |
| 8 | Configuring SSLServerSockets |

# Unit-8

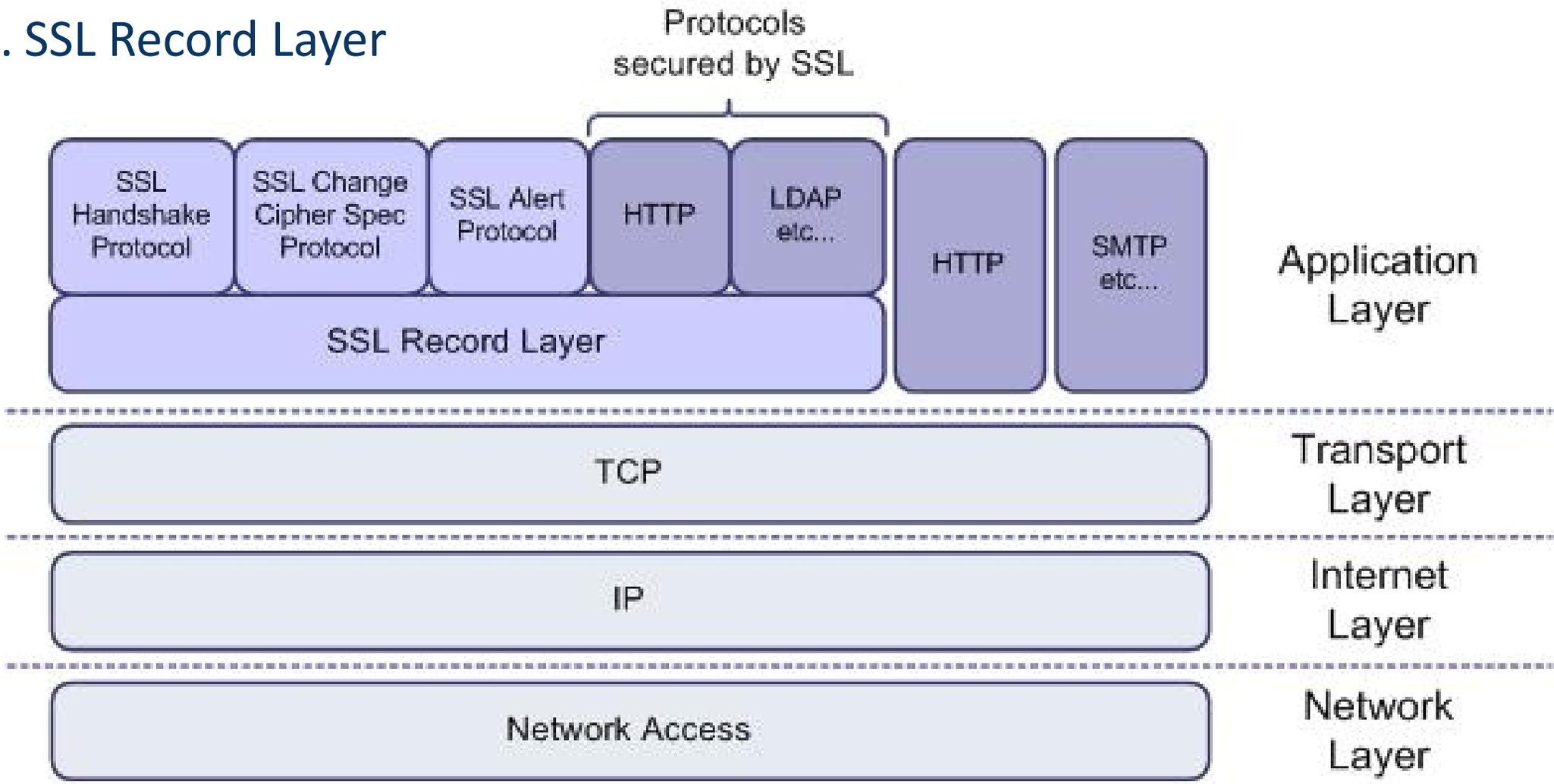
# Secure Sockets

## SSL Protocol Stack

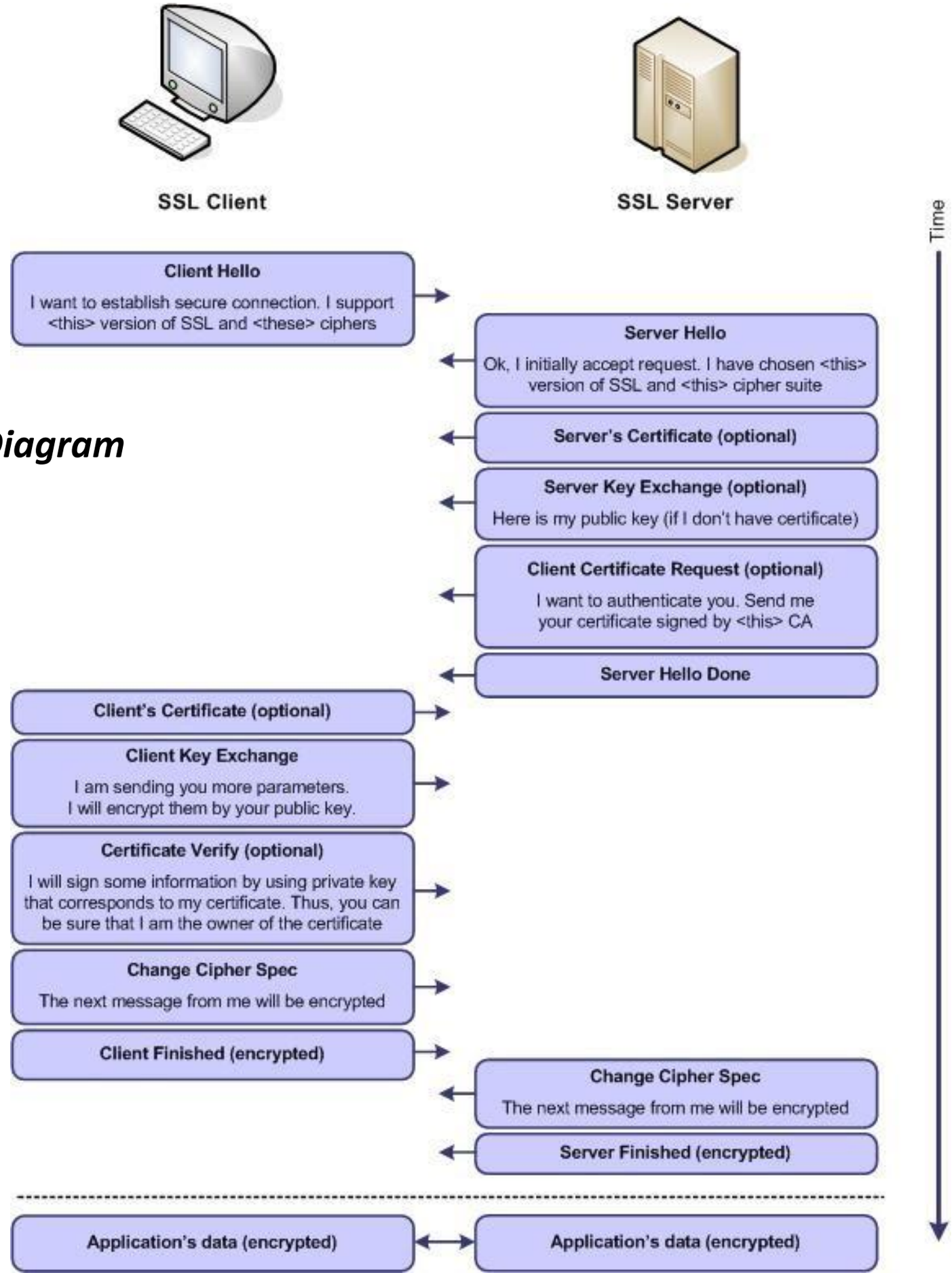
*SSL is a layered protocol and consists of four sub-protocols:*

1. SSL Handshake Protocol
2. SSL Change Cipher Spec Protocol
3. SSL Alert Protocol
4. SSL Record Layer

***TCP/IP model has been from below diagram:***



## *Simplified SSL Handshake Diagram*



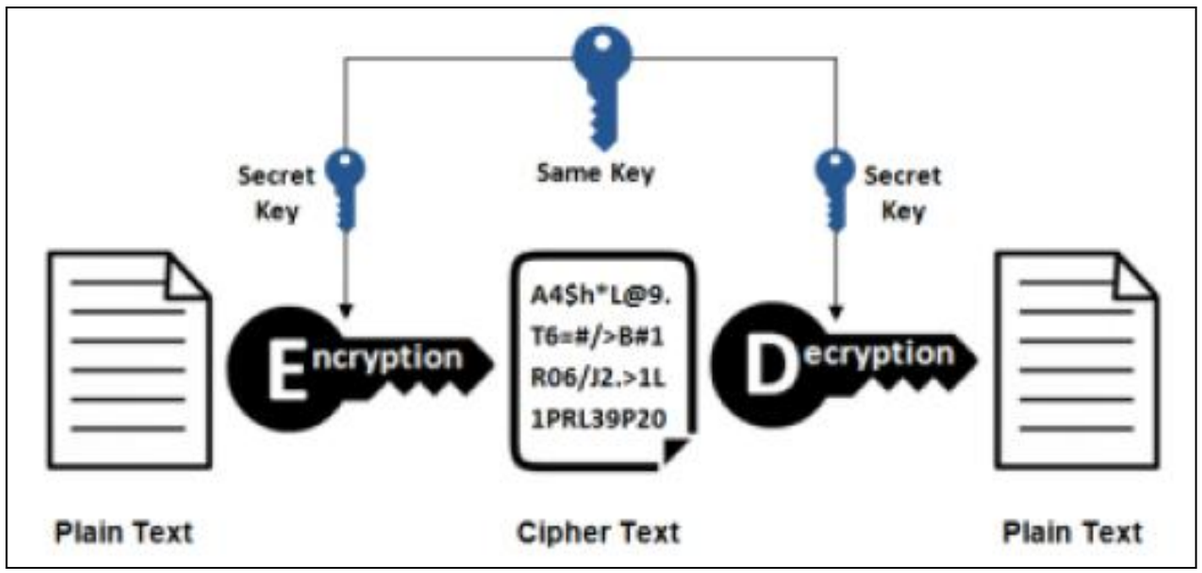
https://networkprogam-mmc.blogspot.com/

# Unit-8

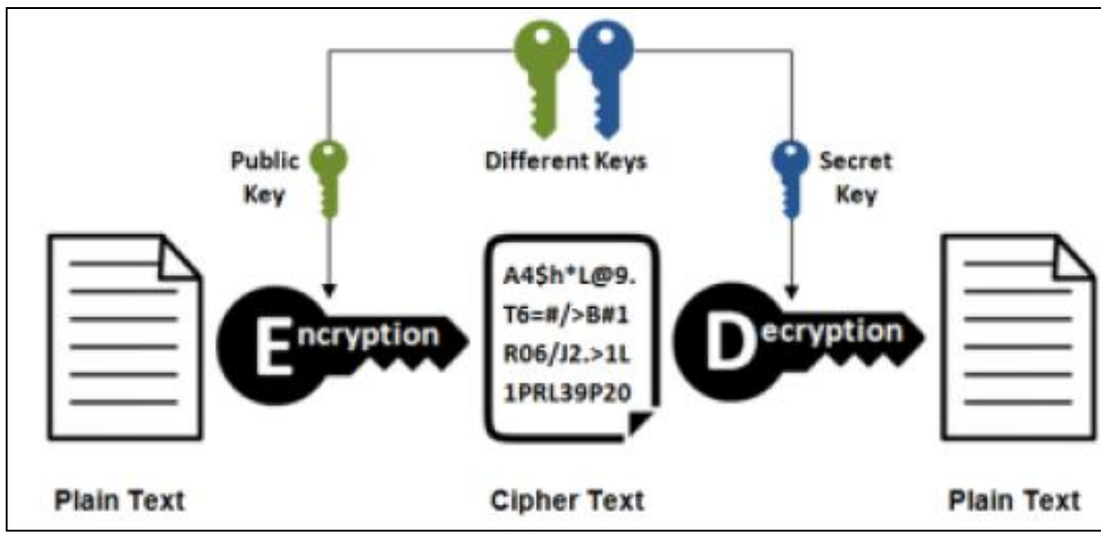
# Secure Sockets

- The Java Secure Sockets Extension (**JSSE**) can secure network communications using the Secure Sockets Layer (**SSL**) Version 3 and Transport Layer Security (**TLS**) protocols and their associated algorithms.
- SSL is a security protocol that enables web browsers and other TCP clients to talk to HTTP and other TCP servers using various levels of confidentiality and authentication.

## 1. Traditional Secret Key (Symmetric) Encryption




## 2. Public Key (ASymmetric) Encryption



# Unit-8

# 2. Creating Secure Client Sockets

- createSocket() from javax.net.ssl.SSLSocketFactory

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

```
Socket socket = new Socket("time.nist.gov", 13);
```

- Five createSocket() methods to build an SSLSocket

```
public abstract Socket createSocket(String host, int port)
    throws IOException, UnknownHostException
public abstract Socket createSocket(InetAddress host, int port)
    throws IOException
public abstract Socket createSocket(String host, int port,
    InetAddress interface, int localPort)
    throws IOException, UnknownHostException
public abstract Socket createSocket(InetAddress host, int port,
    InetAddress interface, int localPort)
    throws IOException, UnknownHostException
public abstract Socket createSocket(Socket proxy, String host, int port,
    boolean autoClose) throws IOException
```

  - Actually return javax.net.ssl.SSLSocket, a subclass of java.net.Socket
  - Call getInputStream() and getOutputStream() as usual to get streams
    - Output/Write as usual

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

Writer out = new OutputStreamWriter(socket.getOutputStream(),
    "US-ASCII");
out.write("Name: John Smith\r\n");
out.write("Product-ID: 67X-89\r\n");
out.write("Address: 1280 Deniston Blvd, NY NY 10003\r\n");
out.write("Card number: 4000-1234-5678-9017\r\n");
out.write("Expires: 08/05\r\n");
out.flush();
```

# Unit-8

## 3. Choosing the Cipher Suites

- Different implementations of the JSSE support *different combinations of authentication and encryption algorithms*
  - Oracle bundles with Java 7 only supports 128-bit AES encryption
- getSupportedCipherSuites()

```
public abstract String[] getSupportedCipherSuites()
```

  - tells which combination of algorithms is available on a given socket
- getEnabledCipherSuites()

```
public abstract String[] getEnabledCipherSuites()
```

  - tells which suites this socket is willing to use
- setEnabledCipherSuites()

```
public abstract void setEnabledCipherSuites(String[] suites)
```

  - change the suites the client attempts to use
  - Support list of Oracle's JDK 1.7 (the first 28 members are default enabled)
    - Four parts: protocol, key exchange algorithm, encryption algorithm, and checksum

SSL_DH_anon_EXPORT_WITH_DES40_CBC_SHA

- Secure Sockets Layer Version 3;
- Diffie-Hellman method for key agreement;
- no authentication;
- DES encryption with 40-bit keys;
- Cipher Block Chaining, and
- the Secure Hash Algorithm checksum

- TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256
- TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256
- TLS_RSA_WITH_AES_128_CBC_SHA256
- TLS_ECDH_ECDSA_WITH_AES_128_CBC_SHA256
- TLS_ECDH_RSA_WITH_AES_128_CBC_SHA256
- TLS_DHE_RSA_WITH_AES_128_CBC_SHA256
- TLS_DHE_DSS_WITH_AES_128_CBC_SHA256
- TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA
- TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA
- TLS_RSA_WITH_AES_128_CBC_SHA
- TLS_ECDH_ECDSA_WITH_AES_128_CBC_SHA
- TLS_ECDH_RSA_WITH_AES_128_CBC_SHA
- TLS_DHE_RSA_WITH_AES_128_CBC_SHA
- TLS_DHE_DSS_WITH_AES_128_CBC_SHA
- TLS_ECDHE_ECDSA_WITH_RC4_128_SHA
- TLS_ECDHE_RSA_WITH_RC4_128_SHA
- SSL_RSA_WITH_RC4_128_SHA
- TLS_ECDH_ECDSA_WITH_RC4_128_SHA
- TLS_ECDH_RSA_WITH_RC4_128_SHA
- TLS_ECDHE_ECDSA_WITH_3DES_EDE_CBC_SHA
- TLS_ECDHE_RSA_WITH_3DES_EDE_CBC_SHA
- SSL_RSA_WITH_3DES_EDE_CBC_SHA
- TLS_ECDH_ECDSA_WITH_3DES_EDE_CBC_SHA
- TLS_ECDH_RSA_WITH_3DES_EDE_CBC_SHA
- SSL_DHE_RSA_WITH_3DES_EDE_CBC_SHA
- SSL_DHE_DSS_WITH_3DES_EDE_CBC_SHA
- SSL_RSA_WITH_RC4_128_MD5
- TLS_EMPTY_RENEGOTIATION_INFO_SCSV
- TLS_DH_anon_WITH_AES_128_CBC_SHA256
- TLS_ECDH_anon_WITH_AES_128_CBC_SHA
- TLS_DH_anon_WITH_AES_128_CBC_SHA
- TLS_ECDH_anon_WITH_RC4_128_SHA
- SSL_DH_anon_WITH_RC4_128_MD5
- TLS_ECDH_anon_WITH_3DES_EDE_CBC_SHA
- SSL_DH_anon_WITH_3DES_EDE_CBC_SHA
- TLS_RSA_WITH_NULL_SHA256
- TLS_ECDHE_ECDSA_WITH_NULL_SHA
- TLS_ECDHE_RSA_WITH_NULL_SHA
- SSL_RSA_WITH_NULL_SHA
- TLS_ECDH_ECDSA_WITH_NULL_SHA
- TLS_ECDH_RSA_WITH_NULL_SHA
- TLS_ECDH_anon_WITH_NULL_SHA
- SSL_RSA_WITH_NULL_MD5
- SSL_RSA_WITH_DES_CBC_SHA
- SSL_DHE_RSA_WITH_DES_CBC_SHA
- SSL_DHE_DSS_WITH_DES_CBC_SHA
- SSL_DH_anon_WITH_DES_CBC_SHA
- SSL_RSA_EXPORT_WITH_RC4_40_MD5
- SSL_DH_anon_EXPORT_WITH_RC4_40_MD5
- SSL_RSA_EXPORT_WITH_DES40_CBC_SHA
- SSL_DHE_RSA_EXPORT_WITH_DES40_CBC_SHA
- SSL_DHE_DSS_EXPORT_WITH_DES40_CBC_SHA
- SSL_DH_anon_EXPORT_WITH_DES40_CBC_SHA
- TLS_KRB5_WITH_RC4_128_SHA
- TLS_KRB5_WITH_RC4_128_MD5
- TLS_KRB5_WITH_3DES_EDE_CBC_SHA
- TLS_KRB5_WITH_3DES_EDE_CBC_MD5
- TLS_KRB5_WITH_DES_CBC_SHA
- TLS_KRB5_WITH_DES_CBC_MD5
- TLS_KRB5_EXPORT_WITH_RC4_40_SHA
- TLS_KRB5_EXPORT_WITH_RC4_40_MD5
- TLS_KRB5_EXPORT_WITH_DES_CBC_40_SHA
- TLS_KRB5_EXPORT_WITH_DES_CBC_40_MD5

# Unit-8

## 4. Event Handlers

- Authenticated network communications are slower
  - It may take several seconds, so you may deal with the connection asynchronously
- JSSE uses the standard Java event model to notify programs when the handshaking between client and server is complete
  - SSLSocket registers HandshakeCompletedListener objects

```
public abstract void addHandshakeCompletedListener(
    HandshakeCompletedListener listener)
public abstract void removeHandshakeCompletedListener(
    HandshakeCompletedListener listener) throws IllegalArgumentException
```

  - HandshakeCompletedListener interface
    - Declare handshakeCompleted() method to receive an argument HandshakeCompletedEvent

```
public void handshakeCompleted(HandshakeCompletedEvent event)
public class HandshakeCompletedEvent extends java.util.EventObject
```

    - Four methods for getting information about the event

```
public SSLSession getSession()
public String getCipherSuite()
public X509Certificate[] getPeerCertificateChain()
    throws SSLPeerUnverifiedException
public SSLSocket getSocket()
```

# Unit-8

## 5. Session Management

- SSL allows sessions to extend over multiple sockets
  - Multiple sockets within the same session use the same set of public/private keys
- JSSE represents by instances of the SSLSession interface
  - Reuses the session's keys automatically if
    - Multiple secure sockets to one host on one port are opened
    - Within a reasonably short period of time
  - In high security applications, you may want to disallow session-sharing between sockets or force reauthentication of a session
- getSession() method of SSLSocket returns the Session

```
public abstract SSLSession getSession()
```

  - Get various information about the session

```
public byte[] getId()
public SSLSessionContext getSessionContext()
public long getCreationTime()
public long getLastAccessedTime()
public void invalidate()
public void putValue(String name, Object value)
public Object getValue(String name)
public void removeValue(String name)
public String[] getValueNames()
public X509Certificate[] getPeerCertificateChain()
 throws SSLPeerUnverifiedException
public String getCipherSuite()
public String getPeerHost()
```

- setEnableSessionCreation() method
  - To allow/disallow session

```
public abstract void setEnableSessionCreation(boolean allowSessions)
public abstract boolean getEnableSessionCreation()
```

- startHandshake() method
  - To reauthenticate a connection

```
public abstract void startHandshake() throws IOException
```

# Unit-8

## 6. Client Mode

- Usually the server is required to authenticate itself, but the client doesn't
  - i.e. Amazon proves to the browsers that it is indeed Amazon
- setUseClientMode() method determines whether the socket needs to use authentication in its first handshake
  - When true is passed in, the socket is in client mode (won't offer to authenticate itself)

```
public abstract void setUseClientMode(boolean mode)
    throws IllegalArgumentException
public abstract boolean getUseClientMode()
```

  - Can be set only once for any given socket
- setNeedClientAuth() method requires that all clients also need to authenticate themselves

```
public abstract void setNeedClientAuth(boolean needsAuthentication)
    throws IllegalArgumentException
public abstract boolean getNeedClientAuth()
```

# Unit-8

# 7. Creating Secure Server Sockets

- javax.net.SSLServerSocket (SSL-enabled server sockets) created by abstract factory class javax.net.SSLServerSocketFactory with createServerSocket()

```
public abstract class SSLServerSocketFactory
    extends ServerSocketFactory
public static ServerSocketFactory getDefault()

public abstract class SSLServerSocket extends ServerSocket
```

```
public abstract ServerSocket createServerSocket(int port)
    throws IOException
public abstract ServerSocket createServerSocket(int port,
    int queueLength) throws IOException
public abstract ServerSocket createServerSocket(int port,
    int queueLength, InetAddress interface) throws IOException
```

  - SSLServerSocketFactory.getDefault() generally only supports server authentication. It doesn't support encryption. More initialization and setup are required
    - Sun's reference implementation, a com.sun.net.ssl.SSLContext object is responsible for creating fully configured and initialized secure server sockets
    - Steps to create a secure server socket in the reference implementation
      1. Generate public keys and certificates using *keytool*
      2. Pay money to have your certificates authenticated by a trusted third party such as Comodo
      3. Create an SSLContext for the algorithm you'll use
      4. Create a TrustManagerFactory for the source of certificate material you'll be using (Oracle/Sun default)
      5. Create a KeyManagerFactory for the type of key material you'll be using
      6. Create a KeyStore object for the key and certificate database (Oracle's default is JKS)
      7. Fill the KeyStore object with keys and certificates; for instance, by loading them from the filesystem using the passphrase they're encrypted with
      8. Initialize the KeyManagerFactory with the KeyStore and its passphrase
      9. Initialize the context with the necessary key managers from the KeyManagerFactory, trust managers from the TrustManagerFactory, and a source of randomness

      (The last two can be null if you're willing to accept the defaults.)

# Unit-8

# 8. Configuring SSLServerSockets

- Like SSLSocket, SSLServerSocket provides methods to choose cipher suites, manage sessions, and establish whether clients are required to authenticate themselves
  - Most of these methods have the similar names in SSLSocket
- Choosing the Cipher Suites

```
public abstract String[] getSupportedCipherSuites()
public abstract String[] getEnabledCipherSuites()
public abstract void     setEnabledCipherSuites(String[] suites)
```

  - newEnabled[]
    = anonCipherSuitesSupported[]
    + oldEnabled[]

```
// add anonymous (non-authenticated) cipher suites
String[] supported = server.getSupportedCipherSuites();
String[] anonCipherSuitesSupported = new String[supported.length];
int numAnonCipherSuitesSupported = 0;
for (int i = 0; i < supported.length; i++) {
  if (supported[i].indexOf("_anon_") > 0) {
    anonCipherSuitesSupported[numAnonCipherSuitesSupported++] =
                                                    supported[i];
} }
String[] oldEnabled = server.getEnabledCipherSuites();
String[] newEnabled = new String[oldEnabled.length
    + numAnonCipherSuitesSupported];
Systen.arraycopy(oldEnabled, 0, newEnabled, 0, oldEnabled.length);
Systen.arraycopy(anonCipherSuitesSupported, 0, newEnabled,
    oldEnabled.length, numAnonCipherSuitesSupported);
server.setEnabledCipherSuites(newEnabled);
```

- Session Management
  - Session creation is enabled by default

```
public abstract void setEnableSessionCreation(boolean allowSessions)
public abstract boolean getEnableSessionCreation()
```

- Client Mode
  - setNeedClientAuth() method (default is false)

```
public abstract void setNeedClientAuth(boolean flag)
public abstract boolean getNeedClientAuth()
```

    - True: only connections in which the client is able to authenticate itself will be accepted
  - setUseClientMode() method (default is false for SSLServerSocket)

```
public abstract void setUseClientMode(boolean flag)
public abstract boolean getUseClientMode()
```

    - True if the SSLSeverSocket should be treated as a client in the communication wrt authentication and other negotiations

# Unit-8

# 8. Configuring SSLServerSockets

- Like SSLSocket, SSLServerSocket provides methods to choose cipher suites, manage sessions, and establish whether clients are required to authenticate themselves
  - Most of these methods have the similar names in SSLSocket
- Choosing the Cipher Suites

```
public abstract String[] getSupportedCipherSuites()
public abstract String[] getEnabledCipherSuites()
public abstract void     setEnabledCipherSuites(String[] suites)
```

  - newEnabled[]
    = anonCipherSuitesSupported[]
    + oldEnabled[]

```
// add anonymous (non-authenticated) cipher suites
String[] supported = server.getSupportedCipherSuites();
String[] anonCipherSuitesSupported = new String[supported.length];
int numAnonCipherSuitesSupported = 0;
for (int i = 0; i < supported.length; i++) {
  if (supported[i].indexOf("_anon_") > 0) {
    anonCipherSuitesSupported[numAnonCipherSuitesSupported++] =
                                                    supported[i];
} }
String[] oldEnabled = server.getEnabledCipherSuites();
String[] newEnabled = new String[oldEnabled.length
    + numAnonCipherSuitesSupported];
Systen.arraycopy(oldEnabled, 0, newEnabled, 0, oldEnabled.length);
Systen.arraycopy(anonCipherSuitesSupported, 0, newEnabled,
    oldEnabled.length, numAnonCipherSuitesSupported);
server.setEnabledCipherSuites(newEnabled);
```

- Session Management
  - Session creation is enabled by default

```
public abstract void setEnableSessionCreation(boolean allowSessions)
public abstract boolean getEnableSessionCreation()
```

- Client Mode
  - setNeedClientAuth() method (default is false)

```
public abstract void setNeedClientAuth(boolean flag)
public abstract boolean getNeedClientAuth()
```

    - True: only connections in which the client is able to authenticate itself will be accepted
  - setUseClientMode() method (default is false for SSLServerSocket)

```
public abstract void setUseClientMode(boolean flag)
public abstract boolean getUseClientMode()
```

    - True if the SSLSeverSocket should be treated as a client in the communication wrt authentication and other negotiations

# Unit-8 Write a Program for Creating Secure Sockets with tufoshss.edu.np

```
HTTP/1.1 301 Moved Permanently
Date: Tue, 05 Jul 2022 00:38:04 GMT
Server: Apache
Expires: Wed, 11 Jan 1984 05:00:00 GMT
Cache-Control: no-cache, must-revalidate, max-age=0
X-Redirect-By: WordPress
Location: https://www.tufohss.edu.np
Content-Length: 0
Content-Type: text/html; charset=UTF-8
```

```java
import javax.net.ssl.SSLSocket;
import javax.net.ssl.SSLSocketFactory;
import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.io.OutputStreamWriter;
import java.io.Writer;
public class SecureScoketEx {
    public static void main ( String[] args ) {
        try {
            SSLSocketFactory factory = (SSLSocketFactory)SSLSocketFactory.getDefault();
            //System.out.println(factory);
            SSLSocket socket = (SSLSocket) factory.createSocket("tufohss.edu.np", 443);
            String[] supported = socket.getSupportedCipherSuites();
            socket.setEnabledCipherSuites(supported);
            Writer out = new OutputStreamWriter(socket.getOutputStream(),"US-ASCII");
            out.write("GET http://tufohss.edu.np/ HTTP/1.1\r\n");
            out.write("Host:tufohss.edu.np\r\n");
            out.write("\r\n");
            out.flush();
            //Read all header fields
            BufferedReader in = new BufferedReader(new InputStreamReader(socket.getInputStream()));
            String s;
            while (!(s=in.readLine()).equals("")){
                System.out.println(s);
            }
        } catch (Exception e) {
        }
    }
}
```

# Unit-8 Write a Program for Creating Secure ServerSockets and Client Sockets.

```
// ServerSocketEx.java
import java.net.ServerSocket;
import java.net.Socket;
import javax.net.ssl.SSLServerSocketFactory;
public class ServerSocketEx {
    public static void main ( String[] args ) {
 try{
    SSLServerSocketFactory factory = (SSLServerSocketFactory)SSLServerSocketFactory.getDefault();
    ServerSocket serverSocket = factory.createServerSocket(1422);
    Socket s = serverSocket.accept();
    System.out.println(s + "Client Accepted and connected....");
    s.close();
    }catch (Exception e){
        }
    }
}
```

```
//ClientSocketEx.java
import javax.net.ssl.SSLSocketFactory;
import java.net.Socket;
public class ClientSocketEx {
    public static void main ( String[] args ) {
        try{
            SSLSocketFactory factory = (SSLSocketFactory)SSLSocketFactory.getDefault();
            Socket socket = factory.createSocket("localhost",1422);
            System.out.println("Server Connected: " + socket);
            socket.close();
        }catch (Exception e){
        }
    }
}
```

# Unit-8 This example loads the necessary keys and certificates from a file name.



**Goto "CMD"** >java -version

**C:\Users\sital>**keytool -genkey -alias zastore -keystore za.store

Enter keystore password: **password**
Re-enter new password: **password**
What is your first and last name?
[Unknown]: sital
What is the name of your organizational unit?
[Unknown]: sital
What is the name of your organization?
[Unknown]: computer
What is the name of your City or Locality?
[Unknown]: jhapa
What is the name of your State or Province?
[Unknown]: 1
What is the two-letter country code for this unit?
[Unknown]: NP
Is CN=sital, OU=sital, O=computer, L=jhapa, ST=1, C=NP correct?
[no]: **yes**
Enter key password for <zastore>
(RETURN if same as keystore password): **password**
Re-enter new password: **password**

**Warning:**
The JKS keystore uses a proprietary format. It is recommended to migrate to PKCS 12 which is an industry standard format using "keytool -importkeystore -srckeyst ore za.store -destkeystore za.store -deststoretype pkcs12".

**Goto same location--→ C:\Users\sital>**

# Unit-8 This example loads the necessary keys and certificates from a file name.

```
$ keytool -genkey -alias ourstore -keystore jnp4e.keys
Enter keystore password:
Re-enter new password:
What is your first and last name?
  [Unknown]:  Elliotte Harold
What is the name of your organizational unit?
  [Unknown]:  Me, Myself, and I
What is the name of your organization?
  [Unknown]:  Cafe au Lait
What is the name of your City or Locality?
  [Unknown]:  Brooklyn
What is the name of your State or Province?
  [Unknown]:  New York
What is the two-letter country code for this unit?
  [Unknown]:  NY
Is <CN=Elliotte Harold, OU="Me, Myself, and I", O=Cafe au Lait, L=Brooklyn,
ST=New York, C=NY> correct?
  [no]:  y

Enter key password for <ourstore>
        (RETURN if same as keystore password):
```

**password “2andnotafnord”**

When this is finished, you’ll have a file named *jnp4e.keys, which contains your public* keys. This example loads the necessary keys and certificates from a file named ***jnp4e.keys*** *in* the **current working directory**.

Default Oracle includes a verified keystore file called ***testkeys****, protected* with the password “**passphrase**,”.

1 - First Generate the server Certificate and public/private key and store it in keystore file

***keytool -genkey -keyalg RSA -keysize 2048 -validity 360 -alias mykey -keystore myKeyStore.jks***

2 - Export the certficate and the public key that should be send to the client.

***keytool -export -alias mykey -keystore myKeyStore.jks -file mykey.cert***

3 - Add the key at the client side to a TrustedStore to trust the server

***keytool -import -file mykey.cert -alias mykey -keystore myTrustStore.jts***

# Unit-8

# Summary

10.1 Secure Communications
- javax.net.ssl, javax.net, java.security.cert, com.sun.net.ssl

10.2 Creating Secure Client Sockets
- javax.net.ssl.SSLSocket
- HTTPSClient (Example 10-1)

10.3 Choosing the Cipher Suites
- getSupportedCipherSuites(), getEnabledCipherSuites(), setEnabledCipherSuites()

10.4 Event Handlers
- HandshakeCompletedListener

10.5 Session Management
- SSLSession
- getSession(), setEnableSessionCreation(), startHandshake()

10.6 Client Mode
- setUseClientMode(), getUseClientMode(), setNeedClientAuth(), getNeedClientAuth()

10.7 Creating Secure Server Sockets
- javax.net.ssl.SSLServerSocket
- SecureOrderTaker (Example 10-2)

10.8 Configuring SSL Server Sockets
- Similar to SSL client sockets