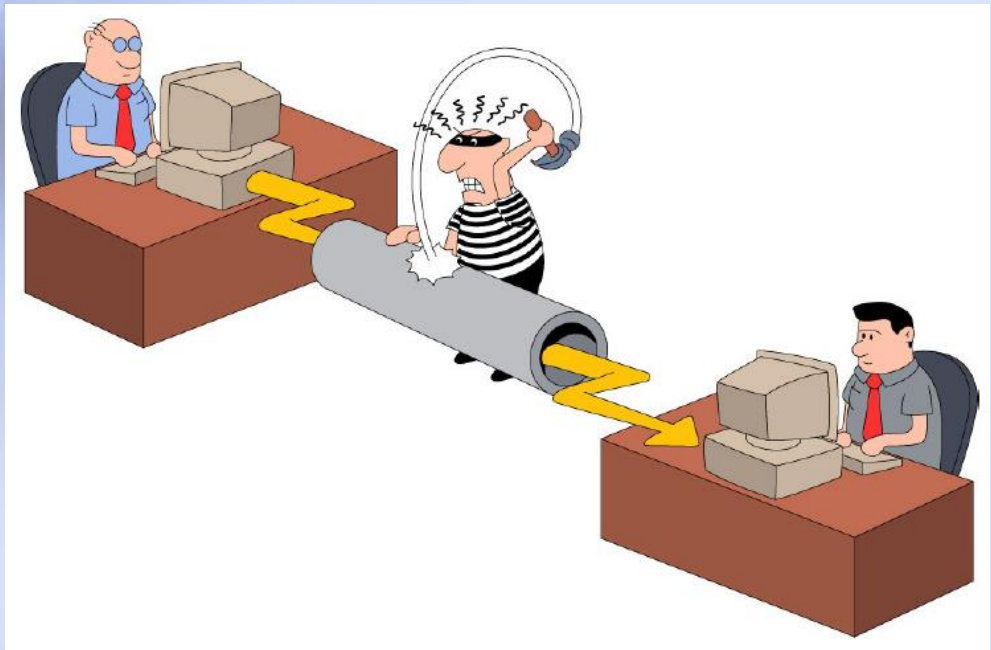
# Sigurna komunikacija

Mrežno i distribuirano programiranje

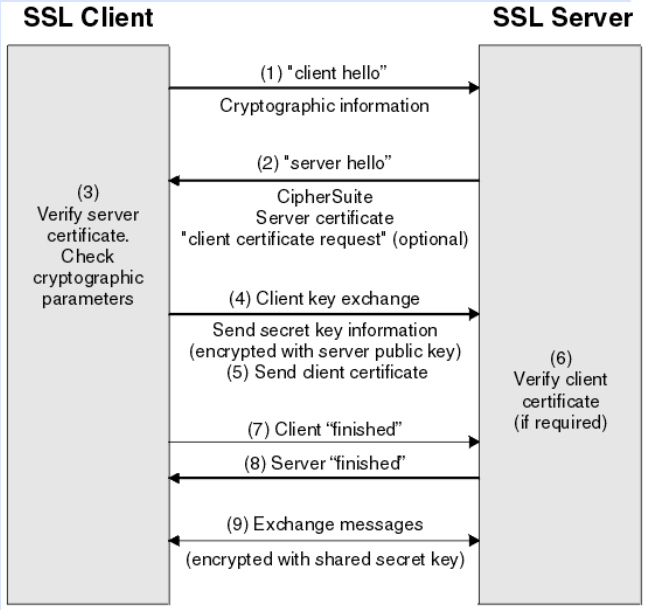
# Uvod

- PKI
- Digitalni sertifikat
- SSL/TLS



# SSL/TLS

- Klijent šalje “client hello” poruku sa kriptografskim informacijama: SSL/TLS verzija, CipherSuite-i koje klijent podržava, opciono informacija o metodama kompresije
  - TLS_RSA_WITH_DES_CBC_SHA
- Server odgovara sa "server hello„ porukom koja sadrži CipherSuite izabran od strane servera, session ID, digitalni sertifikat servera. Ako je potrebna klijentska autentikacija, server šalje "client certificate request" koji uključuje listu tipova sertifikata koje server podržava i DN-ove CA tijela čije sertifikate prihvata
- Klijent verifikuje digitalni sertifikat servera
- Klijent šalje „pre-master secret“ na osnovu kojeg obe strane računaju tajni ključ - „ pre-master secret“ je kriptovan javnim ključem servera
- ako je server ranije poslao "client certificate request", klijent šalje i svoj sertifikat ili "no digital certificate alert„ – ako je klijentska autentikacija obavezna, a klijent ne pošalje sertifikat, handshake se neće završiti uspješno
- server verifikuje klijentski sertifikat
- klijent šalje "finished" poruku, zaštićenu prethodno uspostavljenim algoritmima i ključevima
- server šalje "finished" poruku, zaštićenu prethodno uspostavljenim algoritmima i ključevima
- nakon ovog poruke se mogu kriptovati dijeljenim ključem



# JSSE

- Java Secure Socket Extension (JSSE) obezbjeđuje implementaciju SSL/TLS protokola. Obezbjeđuje mehanizme za enkripciju, serversku autentikaciju, integritet podataka i opciono klijentsku autentikaciju.
- JSSE podržava različite verzije SSL/TLS protokola
- JSSE API je dodatak za java.security i java.net pakete – obezbjeđuje:
  - socket klase
  - socket factory klase
  - trust i key manager-e
- ove klase se nalaze u sljedećim paketima:
  - javax.net
  - javax.net.ssl

# JSSE API

- SSLSocket – Socket koji podržava SSL/TLS protokol
- SocketFactory – *factory* za Socket objekte
- SSLSocketFactory – *factory* za SSLSocket objekte
- SSLServerSocket – serverski Socket koji podržava SSL/TLS protokol
- ServerSocketFactory - *factory* za ServerSocket objekte

# JSSE API

- SSLServerSocketFactory – *factory* za SSLServerSocket objekte
- SSLSession – interfejs ka objektu koji enkapsulira SSL sesiju
- SSLSessionContext – interfejs ka objektu koji enkapsulira kolekciju SSL sesija identifikovanih identifikatorom sesije
- SSLBindingEvent – event klasa koja enkapsulira SSL *session binding* i *unbinding* događaje.

# SSLSocket klasa

- SSLSocket klasa nasljeđuje Socket klasu i obezbjeđuje implementaciju SSL i TLS protokola

```
abstract void addHandshakeCompletedListener
(HandshakeCompletedListener listener)                    // 1
abstract boolean getNeedClientAuth()                     // 2
abstract SSLSession getSession()                         // 3
abstract String[] getSupportedCipherSuites()             // 4
abstract String[] getSupportedProtocols()                // 5
abstract boolean getWantClientAuth()                     // 6
abstract void removeHandshakeCompletedListener
(HandshakeCompletedListener listener)                    // 7
abstract void setEnabledCipherSuites(String[] suites)    // 8
abstract void setEnabledProtocols(String[] protocols)    // 9
abstract void setNeedClientAuth(boolean need)            // 10
abstract void setWantClientAuth(boolean want)            // 11
```

- SSLSocket objekti se kreiraju od strane SSLSocketFactory klase ili prihvatanjem konekcija od strane SSLServerSocket objekta.

# SSLSocketFactory klasa

- SSLSocketFactory klasa zadužena je za kreiranje SSLSocket objekata

```
Socket createSocket(Socket s, InputStream consumed, boolean
autoClose)                                                    // 1
abstract Socket createSocket(Socket s, String host, int port,
boolean autoClose)                                            // 2
static SocketFactory getDefault()                             // 3
abstract String[] getDefaultCipherSuites()                    // 4
abstract String[] getSupportedCipherSuites()                  // 5
```

# SSLSession klasa

- TLS protokol omogućava održavanje sesije preko višestrukih konekcija koje se mogu ostvarivati uzastopno ili paralelno između dva hosta (servera i klijenta)
- na ovaj način više parova *socket*-a mogu koristiti iste parove ključeva
  - ovo će pozitivno uticati na performanse, jer je TLS *handshake* proces relativno spor
  - ovo potencijalno negativno utiče na sigurnost
- na istoj konekciji sesija može biti zamijenjena drugom sesijom
  - pozitivno utiče na sigurnost
  - negativno utiče na performanse

# SSLSession klasa

```
String getCipherSuite()                                    // 1
long getCreationTime()                                     // 2
byte[] getId()                                             // 3
long getLastAccessedTime()                                 // 4
Certificate[] getLocalCertificates()                       // 5
Principal getLocalPrincipal()                              // 6
Certificate[] getPeerCertificates()                        // 7
String getPeerHost()                                       // 8
int getPeerPort()                                          // 9
Principal getPeerPrincipal()                               // 10
String getProtocol()                                       // 11
SSLSessionContext getSessionContext()                      // 12
Object getValue(String name)                               // 13
String[] getValueNames()                                   // 14
void invalidate()                                          // 15
boolean isValid()                                          // 16
void putValue(String name, Object value)                   // 17
void removeValue(String name)                              // 18
```

# SSLServerSocket klasa

- SSLServerSocket klasa nasljeđuje ServerSocket klasu i obezbjeđuje implementaciju SSL i TLS protokola

```
abstract String[] getEnabledCipherSuites()                  // 1
abstract String[] getEnabledProtocols()                     // 2
abstract boolean getEnableSessionCreation()                 // 3
abstract boolean getNeedClientAuth()                        // 4
SSLParameters getSSLParameters()                            // 5
abstract String[] getSupportedCipherSuites()                // 6
abstract String[] getSupportedProtocols()                   // 7
abstract boolean getUseClientMode()                         // 8
abstract boolean getWantClientAuth()                        // 9
abstract void setEnabledCipherSuites(String[] suites)       // 10
abstract void setEnabledProtocols(String[] protocols)       // 11
abstract void setEnableSessionCreation(boolean flag)        // 12
abstract void setNeedClientAuth(boolean need)               // 13
void setSSLParameters(SSLParameters params)                 // 14
abstract void setUseClientMode(boolean mode)                // 15
abstract void setWantClientAuth(boolean want)               // 16
```

- SSLServerSocket objekti se kreiraju od strane SSLServerSocketFactory klase
- osnovni zadatak SSLServerSocket objekta jeste uspostavljanje SSL konekcija i kreiranje SSLSocket objekata

# SSLServerSocketFactory klasa

- SSLServerSocketFactory klasa zadužena je za kreiranje SSLServerSocket objekata

```
static ServerSocketFactory getDefault()                    // 1
abstract String[] getDefaultCipherSuites()                 // 2
abstract String[] getSupportedCipherSuites()               // 3
```

# JKS

- digitalni sertifikati i ključevi čuvaju se u odgovarajućim skladištima (eng. *keystores*)
- JKS (*Java keystore*) je tip skladišta digitalnih sertifikata i ključeva koje koriste Java aplikacije
- pristup ovom skladištu zaštićen je lozinkom
- Java keytool je alat koji omogućava kreiranje JKS skladišta, kreiranje ključeva i digitalnih sertifikata i njihovo smještanje u JKS skladišta, kao i uvoz i izvoz istih u/iz JKS skladišta

# Keytool

- omogućava korisnicima da kreiraju i administriraju njihove privatne i javne ključeve i digitalne sertifikate
- dio JDK
- keystore – datoteka u kojoj se čuvaju ključevi i digitalni sertifikati
- truststore – keystore u kojem se čuvaju javni ključevi / digitalni sertifikati druge strane u komunikaciji

# Keytool

- generisanje keystore-a koji sadrži par ključeva sa sertifikatom:
  - keytool -genkey -alias SecureServer -keyalg RSA –keystore Server_Keystore
- pregled sadržaja keystore-a
  - keytool -list -v -keystore Server_Keystore
- izvoz sertifikata
  - keytool -export -alias SecureServer -keystore Server_Keystore -rfc –file Server.cer
- uvoz sertifikata u trustsore koji koristi klijent
  - keytool -import -alias SecureServer -file Server.cer -keystore Client_Truststore
- pregled sadržaja truststore-a
  - keytool -list -v -keystore Client_Truststore

# Server

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

public class Server {
        int port = portNumber;
        SSLServerSocket server;
        try {
                SSLServerSocketFactory factory =
                (SSLServerSocketFactory)
SSLServerSocketFactory.getDefault();
                server = (SSLServerSocket)
factory.createServerSocket(portNumber);
                SSLSocket client = (SSLSocket) server.accept();
                // rad sa ulaznim/izlaznim stream-ovima
                //
        } catch(Exception e) {
                // obrada izuzetka
        }
}
```

# Klijent

```
import java.io.*;
import javax.net.ssl.*;
public class Client {
        try {
                SSLSocketFactory factory =
(SSLSocketFactory)SSLSocketFactory.getDefault();
                SSLSocket client =
(SSLSOcket)factory.createSocket(serverHost, port);
                // rad sa ulaznim/izlaznim stream-ovima
                //
        } catch(Exception e) {
                // obrada izuzetka
        }
}
```

# JSSE provajderi

- statička konfiguracija:
  - dodati provajder u [JRE_HOME]\lib\security\java.security fajl
- dinamička konfiguracija:
  - java.security.Provider provider = new BouncyCastleProvider();
  - java.security.Security.addProvider(provider);

# RMI sa SSL/TLS podrškom

- Server:
  - System.setProperty("javax.net.ssl.keyStore", "keystore.jks");
  - System.setProperty("javax.net.ssl.keyStorePassword", "test123");
  - RMIClientSocketFactory rmiClientSocketFactory = new SslRMIClientSocketFactory();
  - RMIServerSocketFactory rmiServerSockeyFactory = new SslRMIServerSocketFactory();
  - CalculatorI calculatorServer = new CalculatorServer();
  - CalculatorI stub = (CalculatorI) UnicastRemoteObject.exportObject(calculatorServer, 0,
  - rmiClientSocketFactory, rmiServerSockeyFactory);

- Klijent:
  - System.setProperty("javax.net.ssl.trustStore", "keystore.jks");
  - System.setProperty("javax.net.ssl.keyStorePassword", "test123");