95-702 Distributed Systems

Lecture 14: Some Important Cryptographic Protocols
This Week’s Topics

- Secure Voting
- Cryptographic notation
- Four Scenarios from Colouris
- Needham Schroeder
- Kerberos
- SSL
- Using SSL Sockets
Goals Of Secure Voting

• Only Authorized Voters Can Vote
• No one can vote more than once
• No one can determine for whom anyone else voted
• No one can duplicate anyone else’s vote
• No one can change anyone else’s vote without being discovered
• Every voter can make sure that his vote has been taken into account in the final tabulation.
First Attempt

- Each voter encrypts his vote with the public key of a Central Tabulating Facility (CTF)
- Each voter sends his vote to the CTF
- The CTF decrypts the votes, tabulates them, and makes the results public
- What are some problems with this protocol?
Second Attempt

• Each voter signs his vote with his private key
• Each voter encrypts his signed vote with the CTF’s public key
• Each voter send his vote to the CTF
• The CTF decrypts the votes, checks the signature, tabulates the votes and makes the results public
• What are some problems with this protocol?
### Cast of Characters

<table>
<thead>
<tr>
<th>Character</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alice</td>
<td>First participant</td>
</tr>
<tr>
<td>Bob</td>
<td>Second participant</td>
</tr>
<tr>
<td>Carol</td>
<td>Participant in three- and four-party protocols</td>
</tr>
<tr>
<td>Dave</td>
<td>Participant in four-party protocols</td>
</tr>
<tr>
<td>Eve</td>
<td>Eavesdropper</td>
</tr>
<tr>
<td>Mallory</td>
<td>Malicious attacker</td>
</tr>
<tr>
<td>Sara</td>
<td>A server</td>
</tr>
</tbody>
</table>
# Cryptography Notation

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$K_A$</td>
<td>Alice’s key that she keeps secret.</td>
</tr>
<tr>
<td>$K_B$</td>
<td>Bob’s key that he keeps secret.</td>
</tr>
<tr>
<td>$K_{AB}$</td>
<td>Secret key shared between Alice and Bob</td>
</tr>
<tr>
<td>$K_{A priv}$</td>
<td>Alice’s private key (known only to Alice in asymmetric key crypto)</td>
</tr>
<tr>
<td>$K_{A pub}$</td>
<td>Alice’s public key (published by Alice for all to read)</td>
</tr>
<tr>
<td>${M}^K$</td>
<td>Message $M$ encrypted with key $K$</td>
</tr>
<tr>
<td>$[M]^K$</td>
<td>Message $M$ signed with key $K$</td>
</tr>
</tbody>
</table>
Categories of Encryption Algorithms

Symmetric key encryption. Also called secret key crypto.

Alice sends \( \{M\}K_{ab} \) and Bob can read it.
Bob knows \( K_{ab} \).

Asymmetric key encryption. Also called public key crypto.

Alice sends \( \{M\}K_{Bpub} \) and Bob can read it.
Bob knows \( K_{Bpriv} \).

Public key encryption is typically 100 to 1000 times slower than secret key encryption.
Scenario 1

Goal: Alice and Bob want to exchange messages using a shared and secret symmetric key.

Alice and Bob share $K_{AB}$.
Alice computes $E(K_{AB}, M_i)$ for each message $i$.
She sends these to Bob.
Bob uses $D(K_{AB}, \{M_i\} K_{AB})$ and reads each $M_i$.

Problems:
How do Bob and Alice communicate the key $K_{AB}$?
How does Bob know that $\{M_i\} K_{AB}$ isn’t a replay of an old message?
Scenario 2

Goal: Authenticate Alice allowing her to access files held by Bob.

Alice asks Sarah for a ticket to talk to Bob.
Sarah knows Alice’s password so she can compute $K_A$.
Sarah send to Alice $\{\{Ticket\}K_B, K_{AB}\}K_A$. A challenge!
Alice knows her password and is able to compute $K_A$.
Note that the password is never placed on the network.
Alice is able to compute $\{Ticket\}K_B$ and $K_{AB}$. How?
Alice sends a read request to Bob. She sends
$\{Ticket\}K_B,Alice,Read$. Another challenge!
Bob uses $K_B$ to read the content of the Ticket.
The Ticket is $K_{AB},Alice$. Bob and Alice then use this session key to communicate.

Problems:
Old tickets may be replayed by Mallory. Suppose she has an old session key.
Does not scale well: Sarah must know $K_A, K_B$...
Scenario 3

Goal: Non-repudiation. Alice signs a digital message M.

She computes a digest of M, Digest(M).
If the Digest method is a good one, it is very difficult to find another message M’ so that Digest(M) == Digest(M’).
Alice makes the following available to the intended users:
M,{{Digest(M)}K_{Apriv}}.
Bob obtains the signed document, extracts M and computes Digest(M).
Bob decrypts {{Digest(M)}K_{Apriv}} using K_{Apub} and compares the result with his calculated Digest(M). If they match, the signature is valid.

Problems: Suppose Alice releases her private key to the world. She can now deny that she signed the message.
Scenario 4

Bob and Alice wish to establish a shared secret $K_{AB}$.

Alice uses a key distribution service to get Bob’s public key. This key comes in a certificate. So, Bob’s public key has been signed by a trusted third party, Trent. Alice verifies that Trent signed the public key $K_{Bpub}$.

Alice generates $K_{AB}$ and encrypts it with $K_{Bpub}$.

Bob has many public keys and so Alice sends a key name along as well.

Alice sends key name, $\{K_{AB}\}K_{Bpub}$.
Bob uses the key name to select the correct private key and computes $\{\{K_{AB}\}K_{Bpub}\}K_{Bpriv} = K_{AB}$.

Problem:
The man in the middle attack may be used when Alice first contacts the key distribution service. Mallory may return his own public key (also signed by Trent).
Alice’s Bank Account Certificate

1. Certificate type: Account number
2. Name: Alice
3. Account: 6262626
4. Certifying authority: Bob’s Bank
5. Signature: \{\text{Digest(field 2 + field 3)}\}K_{Bpriv}

Quiz: What is being certified? How would you determine if Bob really signed this certificate?
Public-Key Certificate for Bob’s Bank

1. **Certificate type**: Public key
2. **Name**: Bob’s Bank
3. **Public key**: $K_{Bpub}$
4. **Certifying authority**: Fred – The Bankers Federation
5. **Signature**: ${\text{Digest(field 2 + field 3)}}_{K_{Fpriv}}$

**Quiz**: What is being certified? How would you determine if Fred really signed this public key certificate?
Digital Signatures With Public Keys

Quiz: Is there any attempt at privacy here?
Signatures With a Shared Secret Key

Signing

Verifying

M
K

M
K

H(M+K)

h

h'

h = h'?
# X509 Certificate Format

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Subject</strong></td>
<td>Distinguished Name, Public Key</td>
</tr>
<tr>
<td><strong>Issuer</strong></td>
<td>Distinguished Name, Signature</td>
</tr>
<tr>
<td><strong>Period of validity</strong></td>
<td>Not Before Date, Not After Date</td>
</tr>
<tr>
<td><strong>Administrative information</strong></td>
<td>Version, Serial Number</td>
</tr>
<tr>
<td><strong>Extended Information</strong></td>
<td></td>
</tr>
</tbody>
</table>
The Needham–Schroeder Secret-Key Authentication Protocol

<table>
<thead>
<tr>
<th>Header</th>
<th>Message</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. A-&gt;S:</td>
<td>$A, B, N_A$</td>
<td>A requests S to supply a key for communication with B.</td>
</tr>
<tr>
<td>2. S-&gt;A:</td>
<td>${N_A, B, K_{AB}, {K_{AB}, A}_K_B }_K_A$</td>
<td>S returns a message encrypted in A’s secret key, containing a newly generated key $K_{AB}$ and a ‘ticket’ encrypted in B’s secret key. The nonce $N_A$ demonstrates that the message was sent in response to the preceding one. A believes that S sent the message because only S knows A’s secret key.</td>
</tr>
<tr>
<td>3. A-&gt;B:</td>
<td>${K_{AB}, A}_K_B$</td>
<td>A sends the ‘ticket’ to B.</td>
</tr>
<tr>
<td>4. B-&gt;A:</td>
<td>${N_B}<em>K</em>{AB}$</td>
<td>B decrypts the ticket and uses the new key $K_{AB}$ to encrypt another nonce $N_B$.</td>
</tr>
<tr>
<td>5. A-&gt;B:</td>
<td>${N_B - 1}<em>K</em>{AB}$</td>
<td>A demonstrates to B that it was the sender of the previous message by returning an agreed transformation of $N_B$.</td>
</tr>
</tbody>
</table>
System Architecture of Kerberos

Quiz: Why is this an example of single sign on?

1. Request for TGS ticket
2. TGS ticket
3. Request for server ticket
4. Server ticket
5. Service request

Request encrypted with session key
Reply encrypted with session key

Based on Needham Schroeder

95-702 Distributed Systems
Master of Information System Management
SSL Overview

• Developed by Netscape Communications. The IETF standard is now called TLS.
• *Authenticates* servers (and optionally clients).
• Performs secret key exchange like Diffie-Hellman.
• Data is *encrypted* with the exchanged key.
• Clients do not need to provide a certificate but may be required to do so by the server.
• Client authentication is typically done in the application layer.
• Servers must provide a certificate.
• Normally uses RSA.
• *Data integrity* provided by Message Authentication Codes.
SSL Handshake Protocol

1. ClientHello
2. ServerHello
3. Certificate
4. Certificate Request
5. ServerHelloDone
6. Certificate
7. Certificate Verify
8. Change Cipher Spec
9. Finished
10. Change Cipher Spec
11. Finished

- **Client**
  - Establish protocol version, session ID, cipher suite, compression method, exchange random values
  - Optionally send server certificate and request client certificate
  - Send client certificate response if requested
  - Change cipher suite and finish handshake

- **Server**
  - The client will then send an encrypted session key to the server. See scenario 4.
SSL Protocol Stack

SSL Handshake protocol
SSL Change Cipher Spec
SSL Alert Protocol
HTTP
Telnet

SSL Record Protocol

Transport layer (usually TCP)

Network layer (usually IP)

SSL protocols: HTTP, Telnet
Other protocols:
## TLS Handshake Configuration Options

<table>
<thead>
<tr>
<th>Component</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Key exchange method</td>
<td>the method to be used for exchange of a session key</td>
<td>RSA with public-key certificates</td>
</tr>
<tr>
<td>Cipher for data transfer</td>
<td>the block or stream cipher to be used for data</td>
<td>IDEA</td>
</tr>
<tr>
<td>Message digest function</td>
<td>for creating message authentication codes (MACs)</td>
<td>SHA</td>
</tr>
</tbody>
</table>
“SOAP is going to open up a whole new avenue for security vulnerabilities”

Bruce Schneier, June 2000

SSL will be part of a solution. We’ll see XMLEncryption and XMLDigitalSignature later.
SSL and Web Applications

- **Server authentication** is the typical case since clients may be authenticated at the application level. E.g. Clients may be challenged for user names and passwords.

- **Client authentication** using certificates is also an option.

**Quiz:** If I give you CMU’s public key certificate does that mean I am CMU?
SSL Detail

- Runs on top of TCP/IP
- Uses session key encryption
- Most commonly used to secure HTTP (HTTPS)
- Is an extension of sockets
- Begins with a handshake
Abbreviated Handshake (1)

1) Client sends to server
   -- SSL versions supported by the client
   -- 32 bytes of random data
   -- a made up session ID
   -- a list of supported ciphers
   -- a list of supported compression methods
Abbreviated Handshake (2)

- The server responds with
  - SSL version selected from client’s list
  - 32 bytes of server generated random data
  - The session ID
  - A cipher chosen from the client list
  - The selected compression method
  - A signed public key (certificate)
  - (Perhaps) a request for the client’s certificate (if client authentication is required)
Abbreviated Handshake (3)

- The client
  - checks the server’s certificate
  - sends a client certificate (if required)
  - sends (RSA encrypted) 48 bytes of random data for the construction of a session key
  - if client authentication is required the client hashes all of this and signs the hash with its private key
Abbreviated Handshake (4)

- The server and client share a session key
- All communication is now handled with symmetric key encryption
- Programmers must make very few changes to their code – just use InputStreams and OutputStreams extracted from SSLSockets rather than regular sockets!
Writing a simple SSL Client

• All SSL clients must have a truststore

• If a client is to be verified by the server then the client needs a keystore as well as a truststore

• The truststore
  - holds trusted certificates (signed public keys of CA’s)
  - is in the same format as a keystore
  - is an instance of Java’s KeyStore class
  - is used by the client to verify the certificate sent by the server
  - may be shared with others
Creating a Truststore

(1) Use keytool –genkey to create an RSA key pair

(2) Use keytool –export to generate a self-signed RSA certificate (holding no private key)

(3) Use keytool –import to place the certificate into a truststore
(1) Use keytool - genkey to create an RSA key pair

D:\McCarthy\www\95-804\examples\keystoreexamples> keytool -genkey -alias mjm -keyalg RSA -keystore mjmkeystore

Enter keystore password:  sesame

What is your first and last name?
  [Unknown]: Michael McCarthy

What is the name of your organizational unit?
  [Unknown]: Heinz School

What is the name of your organization?
  [Unknown]: CMU
What is the name of your City or Locality?
    [Unknown]: Pittsburgh

What is the name of your State or Province?
    [Unknown]: PA

What is the two-letter country code for this unit?
    [Unknown]: US

Is CN=Michael McCarthy, OU=Heinz School, O=CMU, L=Pittsburgh, ST=PA, C=US correct?
    [no]: yes

Enter key password for <mjm>
    (RETURN if same as keystore password): <RT>
D:\McCarthy\www\95-804\examples\keystoreexamples>dir /w
Volume in drive D has no label.
Volume Serial Number is 486D-D392

Directory of D:\McCarthy\www\95-804\examples\keystoreexamples

[.]     [..]     mjmkeystore
(2) Use keytool –export to generate a self-signed RSA certificate (holding no private key)

D:\McCarthy\www\95-804\examples\keystoreexamples> keytool -export -alias mjm -keystore mjmkeystore -file mjm.cer
Enter keystore password:  sesame
Certificate stored in file <mjm.cer>

D:\McCarthy\www\95-804\examples\keystoreexamples> dir /w
Volume in drive D has no label.
Volume Serial Number is 486D-D392

Directory of D:\McCarthy\www\95-804\examples\keystoreexamples

[.]  [..]  mjm.cer  mjmkeystore
(3) Use keytool –import to place the certificate into a truststore

D:\McCarthy\www\95-804\examples\keystoreexamples>
keytool -import -alias mjm -keystore mjm.truststore -file mjm.cer

Enter keystore password: sesame
Owner:
CN=Michael McCarthy, OU=Heinz School, O=CMU, L=Pittsburgh, ST=PA, C=US

Issuer:
CN=Michael McCarthy, OU=Heinz School, O=CMU, L=Pittsburgh, ST=PA, C=US
Serial number: 3e60f3ce
Valid from:
Certificate fingerprints:

MD5:

SHA1:
Trust this certificate? [no]: yes
Certificate was added to keystore
D:\McCarthy\www\95-804\examples\keystoreexamples>dir /w
Volume in drive D has no label.
Volume Serial Number is 486D-D392

Directory of D:\McCarthy\www\95-804\examples\keystoreexamples

[.]       [..]           mjm.cer   mjm.truststore   mjmkeystore
5 File(s)    2,615 bytes

mjmkeystore will be placed in the server’s directory
SSL will send the associated certificate to the client

mjm.truststore will be placed in the client’s directory
File Organization

D:\McCarthy\www\95-804\examples\keystoreexamples>tree /f
Directory PATH listing
Volume serial number is 0012FC94 486D:D392
D:.
  │  clientcode
  │      mjm.truststore
  │      Client.java
  └─── servercode
        mjmkeystore
        Server.java
import java.io.*;
import javax.net.ssl.*;
import java.net.*;
import javax.net.*;

public class Client {

    public static void main(String args[]) {

        int port = 6502;
        try {
            // tell the system who we trust
            System.setProperty("javax.net.ssl.trustStore","mjm.truststore");
        }
    }
}
// get an SSLSocketFactory
SocketFactory sf = SSLSocketFactory.getDefault();

// an SSLSocket "is a" Socket
Socket s = sf.createSocket("localhost",6502);

PrintWriter out = new PrintWriter(s.getOutputStream());
BufferedReader in = new BufferedReader(new InputStreamReader(s.getInputStream()));

out.write("Hello server\n");
out.flush();
String answer = in.readLine();
System.out.println(answer);
    out.close();
    in.close();
}

} catch(Exception e) {
    System.out.println("Exception thrown " + e);
}
}
public class Server {

    // hold the name of the keystore containing public and private keys
    static String keyStore = "mjmkeystore";

    // password of the keystore (same as the alias)
    static char keyStorePass[] = "sesame".toCharArray();
public static void main(String args[]) {

    int port = 6502;
    SSLServerSocket server;

    try {
        // get the keystore into memory
        KeyStore ks = KeyStore.getInstance("JKS");
        ks.load(new FileInputStream(keyStore), keyStorePass);

        // initialize the key manager factory with the keystore data
        KeyManagerFactory kmf =
            KeyManagerFactory.getInstance("SunX509");
        kmf.init(ks, keyStorePass);
    }
}
// initialize the SSLContext engine
// may throw NoSuchProvider or NoSuchAlgorithmException exception
// TLS - Transport Layer Security most generic

SSLContext sslContext = SSLContext.getInstance("TLS");

// Initialize context with given KeyManagers, TrustManagers,
// SecureRandom defaults taken if null

sslContext.init(kmf.getKeyManagers(), null, null);

// Get ServerSocketFactory from the context object
ServerSocketFactory ssf = sslContext.getServerSocketFactory();
// Now like programming with normal server sockets
ServerSocket serverSocket = ssf.createServerSocket(port);

System.out.println("Accepting secure connections");

Socket client = serverSocket.accept();
System.out.println("Got connection");

BufferedWriter out = new BufferedWriter(
    new OutputStreamWriter(
        client.getOutputStream()));
BufferedReader in = new BufferedReader(
    new InputStreamReader(
        client.getInputStream()));
String msg = in.readLine();
System.out.println("Got message " + msg);
out.write("Hello client\n");
out.flush();
in.close();
out.close();

} catch(Exception e) {
    System.out.println("Exception thrown " + e);
}
}
On the server

D:\McCarthy\www\95-804\examples\keystoreexamples\servercode> java Server
Accepting secure connections
Got connection
Got message Hello server
On the client

```
D:\McCarthy\www\95-804\examples\keystoreexamples\clientcode>
java Client
Hello client
```
## What we have so far...

<table>
<thead>
<tr>
<th>The Client</th>
<th>The Server</th>
</tr>
</thead>
<tbody>
<tr>
<td>Has a list of public keys it trusts in the file mjm.truststore</td>
<td>Has no list of trusted public keys in a truststore</td>
</tr>
<tr>
<td>Has no public/private key pair of its own</td>
<td>Has a public/private key pair of its own</td>
</tr>
</tbody>
</table>
SSL Server Authentication

Should the client work with this server? Yes, if and only if the server’s public key has been signed by the Company Issued Public Key.
For client authentication we need

1. To generate a key pair for the client
2. Extract a client certificate from the key pair
3. Copy the certificate to the server
4. Import this certificate into the server's truststore
5. Have the server code trust the truststore
6. Have the client code know about its own keys

Quiz: Is there another way?
(1) Generate a key pair for the client

D:\McCarthy\www\95-804\examples\keystoreexamples3\client>
keytool -genkey -alias mjmclient
-keyalg RSA -keystore mjmclientkeystore

Enter keystore password:  sesame
What is your first and last name?
  [Unknown]:  Michael J. McCarthy
What is the name of your organizational unit?
  [Unknown]:  Heinz School
What is the name of your organization?
  [Unknown]:  CMU
What is the name of your City or Locality?  
[Unknown]: Pittsburgh
What is the name of your State or Province?  
[Unknown]: PA
What is the two-letter country code for this unit?  
[Unknown]: US
Is CN=Michael J. McCarthy, OU=Heinz School,  
O=CMU, L=Pittsburgh, ST=PA, C=US correct?  
[no]: yes

Enter key password for <mjmclient>  
(RETURN if same as keystore password):<RT>

Created mjmclientkeystore
(2) Extract a client certificate from the key pair

D:\McCarthy\www\95-804\examples\keystoreexamples3\client>
keytool -export -alias mjmclient -keystore mjmclientkeystore
-file mjmclient.cer

Enter keystore password:  sesame
Certificate stored in file <mjmclient.cer>

Created mjmclient.cer
(3) Copy the certificate to the server

D:\McCarthy\www\95-804\examples\keystoreexamples3\server>dir

03/05/03  12:25p                   602 mjmclient.cer
03/01/03  12:54p                 1,363 mjmkeystore
03/05/03  01:49p                 2,670 Server.class
03/05/03  01:48p                 2,740 Server.java
(4) Import the certificate into the server's truststore

D:\McCarthy\www\95-804\examples\keystoreexamples3\server>

keytool -import -alias mjmcclient -keystore mjmcclient.trustore
-file mjmcclient.cer

Enter keystore password:  sesame
Owner: CN=Michael J. McCarthy, OU=Heinz School, O=CMU, L=Pittsburgh, ST=PA, C=US

Issuer: CN=Michael J. McCarthy, OU=Heinz School, O=CMU, L=Pittsburgh, ST=PA, C=US
Serial number: 3e663114
Valid from: Wed Mar 05 12:17:08 EST 2003 until:
Tue Jun 03 13:17:08 EDT 2003

Certificate fingerprints:
9B:C8
Trust this certificate? [no]: yes
Certificate was added to keystore
D:\McCarthy\www\95-804\examples\keystoreexamples3\server>dir
Volume in drive D has no label.
Volume Serial Number is 486D-D392

Directory of server

03/05/03  12:25p                   602 mjmclient.cer
03/05/03  12:35p                   668 mjmclient.trustore
03/01/03  12:54p                 1,363 mjmkeystore
03/01/03  10:40p                 2,942 Server.class
03/01/03  10:40p                 3,798 Server.java
  9 File(s)         18,184 bytes
(5) Have the server code trust the truststore

// Server side SSL
import java.io.*;
import java.net.*;
import javax.net.*;
import javax.net.ssl.*;
import java.security.*;

public class Server {

    // hold the name of the keystore containing public and private keys
    static String keyStore = "mjmkeystore";

    // password of the keystore (same as the alias)
    static char keyStorePass[] = "sesame".toCharArray();
public static void main(String args[]) {

    int port = 6502;
    SSLServerSocket server;

    try {
        // get the keystore into memory
        KeyStore ks = KeyStore.getInstance("JKS");
        ks.load(new FileInputStream(keyStore), keyStorePass);

        // initialize the key manager factory with the keystore data
        KeyManagerFactory kmf =
            KeyManagerFactory.getInstance("SunX509");
        kmf.init(ks, keyStorePass);
    }
}
// tell the system who we trust, we trust the client's certificate
// in mjmclient.truststore

System.setProperty("javax.net.ssl.trustStore",
    "mjmclient.truststore");

// initialize the SSLContext engine

// may throw NoSuchProvider or NoSuchAlgorithmException exception
// TLS - Transport Layer Security most generic

SSLContext sslContext = SSLContext.getInstance("TLS");
// Initialize context with given KeyManagers, TrustManagers,
// SecureRandom
// defaults taken if null
sslContext.init(kmf.getKeyManagers(), null, null);
// Get ServerSocketFactory from the context object
ServerSocketFactory ssf = sslContext.getServerSocketFactory();

// Now almost like programming with normal server sockets
ServerSocket serverSocket = ssf.createServerSocket(port);
((SSLServerSocket)serverSocket).setNeedClientAuth(true);
System.out.println("Accepting secure connections");
Socket client = serverSocket.accept();
System.out.println("Got connection");
PrintWriter out = new PrintWriter(client.getOutputStream(),true);

BufferedReader in = new BufferedReader(new InputStreamReader(client.getInputStream()));
String fromClient = in.readLine();
System.out.println(fromClient);
out.println("Hello client\n");
out.flush();
in.close();
out.close();
System.out.println("Data sent");
}
catch(Exception e) {
    System.out.println("Exception thrown " + e);
}
}
(6) Have the client code know about its own keys

```java
import java.net.*;
import java.io.*;
import javax.net.ssl.*;
import javax.security.cert.X509Certificate;
import java.security.KeyStore;

public class Client {

    public static void main(String args[]) {

        int port = 6502;
        // tell the system who we trust
        System.setProperty("javax.net.ssl.trustStore","mjm.truststore");
```
try {
    SSLSocketFactory factory = null;
    try {
        SSLContext ctx;
        KeyManagerFactory kmf;
        KeyStore ks;
        char[] passphrase = "sesame".toCharArray();
        ctx = SSLContext.getInstance("TLS");
        kmf = KeyManagerFactory.getInstance("SunX509");
        ks = KeyStore.getInstance("JKS");
        ks.load(new FileInputStream("mjmclientkeystore"),
                passphrase);
        kmf.init(ks, passphrase);
        ctx.init(kmf.getKeyManagers(), null, null);
        factory = ctx.getSocketFactory();
    } catch (Exception e) { throw new IOException(e.getMessage()); }
} catch (Exception e) { throw new IOException(e.getMessage()); }
SSLSocket s = (SSLSocket)factory.createSocket("localhost", port);
s.startHandshake();
PrintWriter out = new PrintWriter(s.getOutputStream());
BufferedReader in = new BufferedReader(
    new InputStreamReader(
        s.getInputStream()));
out.write("Hello server\n");
out.flush();
String answer = in.readLine();
System.out.println(answer);
out.close();
in.close();
} catch(Exception e) {
    System.out.println("Exception thrown " + e); }
Testing

D:\...\server>
java Server
Accepting secure connections
Got connection
Hello server
Data sent

D:\...\client>java Client
Hello client
Testing after deleting the server’s truststore

D:\...\server>java Server
Accepting secure connections
Got connection

Exception thrown javax.net.ssl.SSLHandshakeException:
Couldn't find trusted certificate

D:\...\client>java Client
Exception thrown javax.net.ssl.SSLHandshakeException:
Received fatal alert: certificate_unknown
Testing after deleting the client’s truststore

D:\server\java Server
Accepting secure connections
Got connection
Exception thrown javax.net.ssl.SSLHandshakeException:
Received fatal alert: certificate_unknown

D:\...\client>java Client
Exception thrown javax.net.ssl.SSLHandshakeException:
Couldn't find trusted certificate
Configuring Tomcat for SSL

The web server needs a certificate so that the client can identify the server.

The certificate may be signed by a Certificate Authority or it may be self-signed.

The web server needs a private key as well.
keytool -genkey -keyalg RSA -alias tomcat -keystore .keystore

Enter keystore password:  sesame

What is your first and last name?
[Unknown]:  localhost
What is the name of your organizational unit?
[Unknown]:  Heinz School
What is the name of your organization?
[Unknown]:  CMU
What is the name of your City or Locality?
[Unknown]:  Pgh.
What is the name of your State or Province?
[Unknown]:  PA

Generate public and private keys for Tomcat

The keystore file is called .keystore
What is the two-letter country code for this unit?
[Unknown]: US
Is CN=localhost, OU=Heinz School, O=CMU, L=Pgh., ST=PA, C=US correct?
[no]: yes

Enter key password for <tomcat>
(RETURN if same as keystore password):<RT>

D:\McCarthy\www\95-804\examples\SSLAndTomcat>
Use Admin Tool to tell a Web Container about SSL

(1) Startup Tomcat  
(2) Run the admin server with [http://localhost:8080/admin](http://localhost:8080/admin)  
(3) Log in with your user name and password  
(4) Select Service (Java Web Service Developer Pack)  
(5) Select Create New Connector from the drop down list in the right pane  
(6) In the type field enter HTTPS  
(7) In the port field enter 8443  
(8) Enter complete path to your .keystore file  
(9) Enter keystore password  
(10) Select SAVE and then Commit Changes
Testing

Shutdown Tomcat.

Visit Tomcat from a browser.

Use https://localhost:8443/

You can also visit your other installed web apps through https.
Information you exchange with this site cannot be viewed or changed by others. However, there is a problem with the site's security certificate.

The security certificate was issued by a company you have not chosen to trust. View the certificate to determine whether you want to trust the certifying authority.

The security certificate date is valid.

The security certificate matches the name of the page you are trying to view.

Do you want to proceed?

Yes  No  View Certificate
Root Certificate Store

Do you want to ADD the following certificate to the Root Store?

Subject: localhost, Heinz School, CMU, Pgh., PA, US
Issuer: Self Issued
Time Validity: Thursday, March 06, 2003 through Wednesday, June 04, 2003
Serial Number: 3E677F3C
Thumbprint (sha1): 16800FA7 6B27DF31 A027B360 B1708A5E 4044F69C
Thumbprint (md5): 8BA184C3 8EDFB6AD 6320283F 26DC53FD

Yes  No
Java™ Web Services Developer Pack 1.0_01

The Java Web Services Developer Pack (Java WSDP) is an all-in-one download containing key technologies to simplify building of Web services using the Java™ 2 Platform. The technologies comprising the Java Web Services Developer Pack are:

- Java XML Pack, which includes:
  - Java API for XML Messaging (JAXM)
  - SOAP with Attachments API for Java (SAAJ)
  - Java API for XML Processing (JAXP)
  - Java API for XML Registries (JAXR)
  - Java API for XML-based RPC (JAX-RPC)
- JSP Standard Tag Library (JSTL)
- Tomcat (Java Servlet and JSP container and tools)