Public Key Cryptography in Java

Simon Foley

November 19, 2013

Public Key Crypto

Public Keys in Java: Simple DSA Signatures

```
import java.io.*;
import java.security.*;
  ...
    byte[] data= ”Simon␣Foley”.getBytes();
    /* generate DSA key pair*/
    KeyPairGenerator kpg = KeyPairGenerator.getInstance(”DSA”);
    kpg. initialize (1024);                      /* use default SecureRandom */
    KeyPair pair = kpg.genKeyPair();             /* gen a 1024 bit DSA key */
    /* generate DSA signature generating object and sign data         */
    Signature signature= Signature.getInstance(”DSA”);/* ex factory */
    signature.initSign(pair.getPrivate());       /* use priv key to sign */
    signature.update(data);                      /* feed the data */
    byte[] sig= signature.sign();                /* generate signature */
    /* initialize DSA signature verification object and verify sig  */
    signature.initVerify(pair.getPublic());      /* use pub key to verify */
    signature.update(data);                      /* feed the data */

    if (signature.verify(sig)) ...               /* verify the signature */
```

Java Key Stores

Used to store and manage keys and certificates for a principal.
Includes principal's private keys and public keys (certificates) that it trusts.
keytool provides command-line utility to access and manage user keystore.

```
keytool −genkey −keystore mykeystore −alias SimonsDSA \
          −keyalg DSA −keysize 1024 −storepass spasswd \
          −sigalg DSA −validity 90 −keypass kpasswd \
          −dname ”CN=Simon␣Foley,␣OU=Department␣of␣Computer␣Science,\
O=␣College␣Cork,␣L=Cork,␣C=IE”
```

keytool −genkey generates a new key pair (given parameters)

The key pair is used to create a private key entry in the keystore.

The public key is placed inside a self-signed certificate and is ‘trusted’.

KeyStores: Listing an entry

```
keytool −list −keystore mykeystore −alias simonsDSA −storepass spasswd −v
  Alias name: simonsdsa
  Creation date: Mon Jan 21 11:47:39 GMT+00:00 2002
  Entry type: keyEntry
   Certificate chain length: 1
   Certificate [1]:
  Owner: CN=Simon Foley, OU=Department of Computer Science,
              O=University College Cork, L=Cork, C=IE
  Issuer : CN=Simon Foley, OU=Department of Computer Science,
              O=University College Cork, L=Cork, C=IE
  Serial number: 3c4bffdb
  Valid from: Mon Jan 21 11:47:39 GMT+00:00 2002
           until : Sun Apr 21 11:47:39 GMT+00:00 2002
   Certificate fingerprints :
         MD5: 39:E3:3F:C1:9F:FF:4C:39:72:7B:E7:90:77:C1:7B:36
         SHA1: 78:C7:7C:AE:21:00:A3:9A:36:A2:82:71:5C:4C:EF:EB:A3:8A:E7:49
```

KeyStores: Certificate Signing Request

A CSR is intended to be sent to a CA, who will authenticate the certificate (usually off-line) and will return a certificate or a certificate chain, used to replace the existing certificate chain (usually self signed). CSR is in PKCS10 format. Response in PKCS7 format.

```
keytool −certreq −keystore mykeystore −alias simonsDSA −storepass spasswd
Enter key password for <simonsDSA>: kpasswd
```

```
−−−−−BEGIN NEW CERTIFICATE REQUEST−−−−−
MIICgjCCAkACAQAwfTELMAkGA1UEBhMCSUUxDTALBgNVBAcTBENvcmsxIDAeBgNVBAoTF1VuaXZl
cnNpdHkgQ29sbGVnZSBDb3JrMScwJQYDVQQLEx5EZXBhcnRtZW50IG9mIENvbXB1dGVyIFNjaWVu
Y2UxFDASBgNVBAMTC1NpbW9uIEZvbGV5MIIBuDCCASwGByqGSM44BAEwggEfAoGBAP1/U4EddRIp
Ut9KnC7s5Of2EbdSPO9EAMMeP4C2USZpRV1AIlH7WT2NWPq/xfW6MPbLm1Vs14E7gB00b/JmYLdr
mVClpJ+f6AR7ECLCT7up1/63xhv4O1fnxqimFQ8E+4P208UewwI1VBNaFpEy9nXzrith1yrv8iID
GZ3RSAHHAhUAl2BQjxUjC8yykrmCouuEC/BYHPUCgYEA9+GghdabPd7LvKtcNrhXuXmUr7v6OuqC
+VdMCz0HgmdRWVeOutRZT+ZxBxCBgLRJFnEj6EwoFhO3zwkyjMim4TwWeotUfI0o4KOuHiuzpnWR
bqN/C/ohNWLx+2J6ASQ7zKTxvqhRkImog9/hWuWfBpKLZl6Ae1UlZAFMO/7PSSoDgYUAAoGBALa0
P44E1SwtqouG9cxdtVt91p3A0SrKyby5e6rUFiJliaLtv79Ir0ipBVm6A4VVaaVzq9d0Ts/3hDzK
RKpyBlrtVXkNIKTDKaPFB5JxZGHGy8QjtgKk9XIni7EVPqnIhj6TSfJncq4YjrP5qqHtUCVx1WfO
M2UK/ABKo8gw9zwhoAAwCwYHKoZIzjgEAwUAAy8AMCwCFFs2kXLMLhUOaq1f5vIBTwninIlpAhQK
3tk6NegR8cHZBJLQHj2GaeVxfg==
−−−−−END NEW CERTIFICATE REQUEST−−−−−
```

KeyStores: Importing and Exporting signed Certificates

keytool −export is used to export certificates in PKCS7 form.

keytool −export −keystore alicekeystore −alias AlicesDSA −file alice.cer

keytool −**import** reads the certificate or certificate chain and attempts to verify it by constructing a chain of trust from the certificate to a self-signed certificate (typically belonging to a root CA) and trusted certificates that are already available in the keystore.

keytool −import −keystore bobkeystore −alias alicesDSA −file alice.cer

It's up to Bob to check that this is really Alice's certificate.

See Java JDK Tutorial for further details

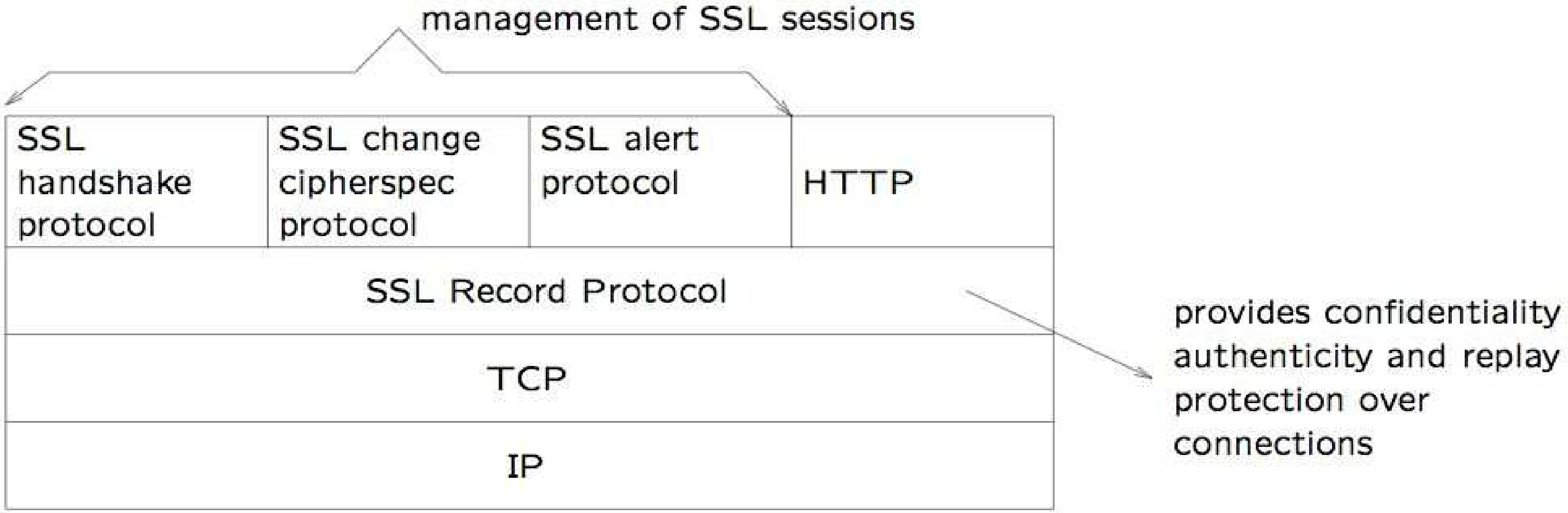
KeyStores: Accessing from Java

```
byte[] data= "Simon␣Foley".getBytes();
String alias = "SimonsDSA";                          /* alias for cert */
String keystorefile = "mykeystore";      /* file containing keystore */
char[] storepass= "spasswd".toCharArray();   /* keystore password */
char[] keypass= "kpasswd".toCharArray();     /* keystore password */
/* load the KeyStore */
KeyStore keystore= KeyStore.getInstance(KeyStore.getDefaultType());
keystore.load(new FileInputStream( keystorefile ), storepass );
/* get DSA signing (private) key cert from the keystore */
PrivateKey priv= (PrivateKey) keystore.getKey( alias ,keypass );
/* generate DSA signature cipher object and sign data              */
Signature signature= Signature.getInstance("DSA");/* ex factory */
signature.initSign ( priv );              /* initialize with signing key */
signature.update(data);                                /* feed the data */
byte[] sig= signature.sign ();                     /* generate signature */
/* initialize DSA signature verification object and verify sig */
signature.initVerify (                   /* re- initialize for verifying */
keystore.getCertificate ( alias ).getPublicKey());
signature.update(data);                                /* feed the data */
```

Secure Sockets Layer

SSL/TLS Protocol

Secure Sockets Layer/Transport Layer Security protocol originally developed by Netscape Corp. TLSv1.2/SSLv3 (RFC 5246) widely deployed in browsers. Provided by JSSE in Java.

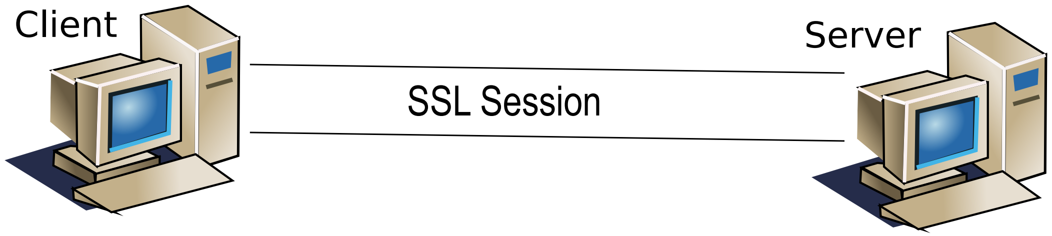


management of SSL sessions

SSL handshake protocol	SSL change cipherspec protocol	SSL alert protocol	HTTP
SSL Record Protocol			
TCP			
IP			

provides confidentiality authenticity and replay protection over connections

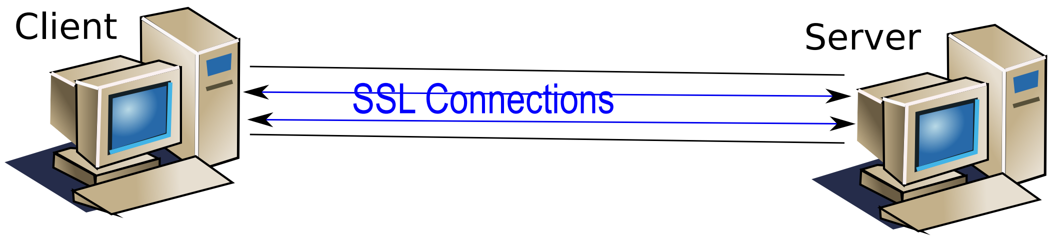
SSL Session



Principals negotiate the ciphers, hash functions and key-exchange, and use security protocol to create security associations between client and server. As a result, the following information is shared between client and server:

- Session Identifier.
- Peer Certificate (X509v3). May be null. SSL supports (optional) server-side and client-side authentication.
- Compression Method. May be null..
- Cipher Specification for bulk encryption and MAC hash.
- Master Secret: establish 48-byte shared secret between the two parties.

SSL Connection



SSL Connections established within SSL session, providing suitable services peer to peer. The following information is shared (keys/etc derived from the master secret):

- Server and client random values.
- Server write MAC secret.
- Client write MAC secret.
- Server write key.
- Client write key.
- IVs and sequence numbers.

SSL/TLS Protocol Overview

$$
\begin{aligned}
Msg1: &\quad \text{Client} \rightarrow \text{Server} &&: hello\\
Msg2: &\quad \text{Server} \rightarrow \text{Client} &&: hello\\
Msg3: &\quad \text{Server} \rightarrow \text{Client} &&: certificate\ (optional)\\
Msg4: &\quad \text{Server} \rightarrow \text{Client} &&: certificate\ request\ (optional)\\
Msg5: &\quad \text{Server} \rightarrow \text{Client} &&: server\ key\ exchange\ (optional)\\
Msg6: &\quad \text{Server} \rightarrow \text{Client} &&: Server\ hello\ done\\
Msg7: &\quad \text{Client} \rightarrow \text{Server} &&: certificate\ (optional)\\
Msg8: &\quad \text{Client} \rightarrow \text{Server} &&: client\ key\ exchange\\
Msg9: &\quad \text{Client} \rightarrow \text{Server} &&: certificate\ verify\ (optional)\\
Msg10: &\quad \text{Client} \rightarrow \text{Server} &&: change\ cipher\ spec\\
Msg11: &\quad \text{Client} \rightarrow \text{Server} &&: finished\\
Msg12: &\quad \text{Client} \rightarrow \text{Server} &&: change\ cipher\ spec\\
Msg13 &\quad \text{Client} \rightarrow \text{Server} &&: change\ cipher\ spec
\end{aligned}
$$

SSL/TLS Protocol Overview

Msg1. Client sends server information including the highest version of SSL that it supports and a list of the cipher suites it supports. Cipher suites include crypto algorithm and key sizes.

Msg2 Server selects the version, etc.

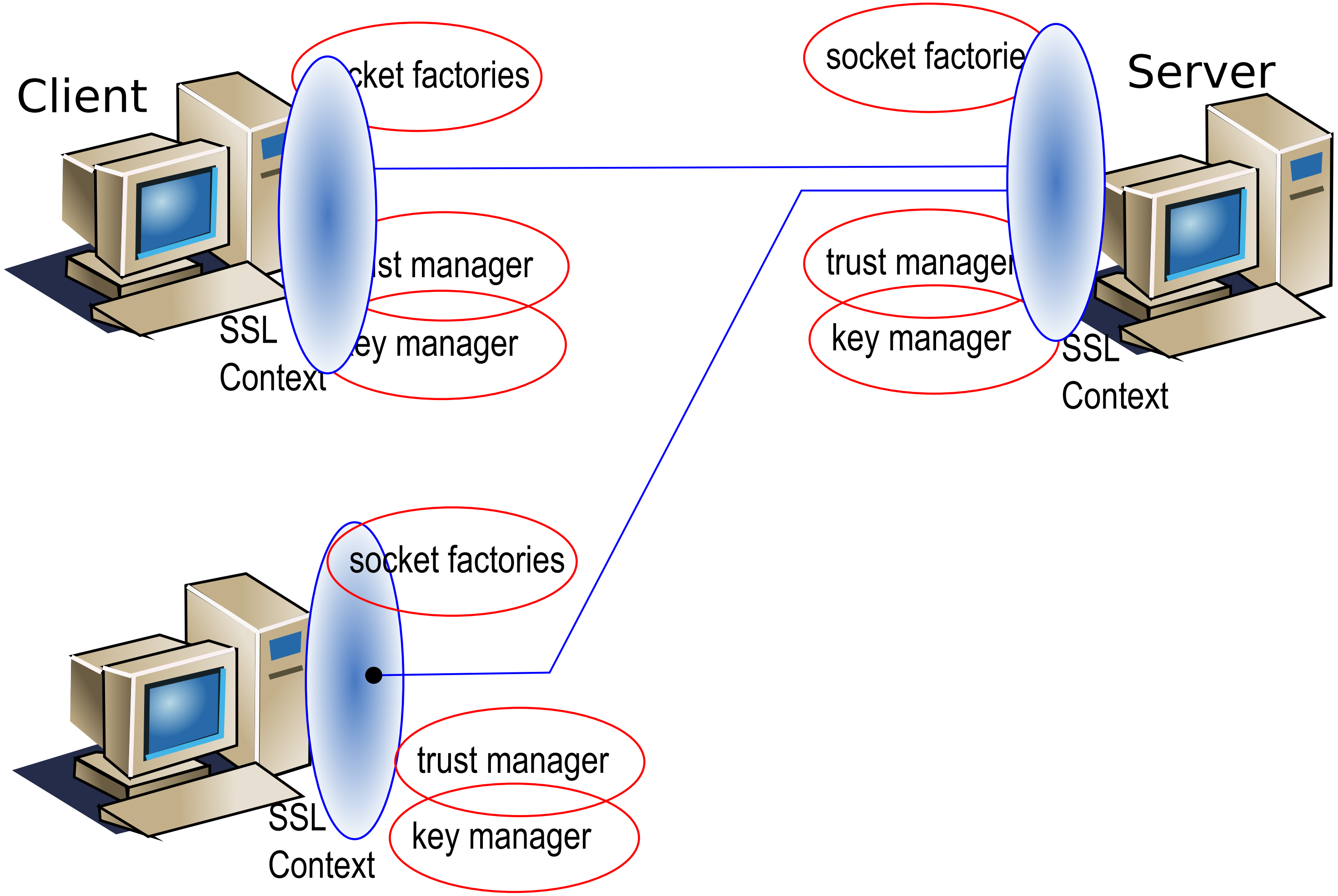
Msg3. Server sends certificate [chain]; used when authentication required.

Msgs 4,5,7 Server may request client authentication.

Msg8. Client generates secret that will be used to derive keys for symmetric encryption. For example, for RSA, the secret (key) is encrypted using Server's public key.

Msg10, Msg12. Principals tell each other to change to encrypted mode.

JSSE (SSL/TLS in Java): Footprint



JSSE Key Managers

A KeyManager object holds the peer's private key and can provide the necessary certificates that allow the peer to be authenticated.

We use a factory to make a default `com.sun.net.ssl.X509KeyManager` object, that comes with Sun's JSSE.

```
/∗ Create the key manager to hold peer's private key ∗/
KeyManagerFactory kmfactory = KeyManagerFactory.getInstance(”SunX509”);
kmfactory. init ( keystore , keypass );
KeyManager[] Peerkm= kmfactory.getKeyManagers();
```

A KeyStore may be passed, which the factory will query for information on which private key and matching public key certificates should be used for authenticating to a remote socket peer. In addition to the KeyStore object, a second parameter gives the password that will be used with the methods for accessing keys from the KeyStore. All keys in the KeyStore must be protected by the same password.

JSSE Trust Managers

A peer uses a TrustManager object to keeps track of the certificates that it can trusts.

We use a factory to make a default com.sun.net.ssl.X509TrustManager object that comes with Sun's JSSE. This provides standard certificate path validation for authorisation.

```
/* Create trust manager to manage certificates Peer trusts */
TrustManagerFactory tmfactory=
                    TrustManagerFactory.getInstance("SunX509");
tmfactory.init(keystore);
TrustManager[] Peertm= tmfactory.getTrustManagers();
```

When generating the TrustManager, the factory will query the keystore to determine which certificates should be trusted when doing authorisation checks.

JSSE SSL Contexts

SSLcontext object is used to manage all the information related to an SSL session. It provides the basis for establishing secure connections, that is, provides factories to generate secure sockets.

```
  /* Create SSL context */
   SSLContext sslcontext= SSLContext.getInstance("SSL");
   sslcontext . init (Peerkm, Peertm, null ); /* use default sec random */
   /* Create server socket factories */
   ServerSocketFactory  ssfactory  = sslcontext . getServerSocketFactory ();
   SocketFactory sfactory = sslcontext .getSocketFactory ();
```

A SSL context is initialised with the Peer's KeyManager, TrustManager and a secure random number generator.

A Secure Server: Initial Setup

```
import java.io.*;
import java.security.*;
import java.net.*;
import javax.net.*;
import javax.net.ssl.*;
import com.sun.net.ssl.*;
import javax.security.cert.*;

public class ServerAlice {
  public static void main (String[] args) throws Exception {

    String alias = "AliceDSA";                    /* alias for private key */
    String keystorefile = "alicekeystore"; /* file containing keystore */
    char[] storepass = "spasswd".toCharArray();  /* keystore password */
    char[] keypass= "kpasswd".toCharArray();     /* keystore password */
    // [next slide ...]
```

A Secure Server: Building Context

```
// [... contd]
/* load the KeyStore */
KeyStore keystore= KeyStore.getInstance(”JCEKS”);
keystore.load(new FileInputStream( keystorefile ), storepass );

/* Create the key manager to hold server's private key */
KeyManagerFactory kmfactory = KeyManagerFactory.getInstance(”SunX509”);
kmfactory.init(keystore,keypass);
KeyManager[] Serverkm= kmfactory.getKeyManagers();

/* Create SSL context */
SSLContext sslcontext= SSLContext.getInstance(”SSL”);
sslcontext.init(Serverkm, null, null); /* default sec random and tm */

/* Create server socket factories */
ServerSocketFactory ssfactory = sslcontext.getServerSocketFactory();
SocketFactory sfactory= sslcontext.getSocketFactory();
// [next slide ...]
```

A Secure Server: Using

```
    // [... contd]
    /* create a server socket that listens on 5999 */
    SSLServerSocket s= (SSLServerSocket) ssfactory.createServerSocket(5999);

    SSLSocket c= (SSLSocket) s.accept();
    DataInputStream in = new DataInputStream(c.getInputStream());
    String msg= in.readUTF();
    System.out.println(msg);
    }
}
```

A Secure Client: Initial Setup

```
import java.io.*;
import java.security.*;
import java.net.*;
import javax.net.*;
import javax.net.ssl.*;
import com.sun.net.ssl.*;
import javax.security.cert.*;

public class ClientBob {
  public static void main (String[] args) throws Exception {

    String alias = "BobDSA";                            /* alias for private key */
    String keystorefile = "bobkeystore";      /* file containing keystore */
    char[] storepass = "spasswd".toCharArray();     /* keystore password */
    char[] keypass= "kpasswd".toCharArray();             /* key password */
    // [next slide ...]
```

A Client: Building Context

```
// [... contd]
/* load the KeyStore */
KeyStore keystore= KeyStore.getInstance("JCEKS");
keystore.load(new FileInputStream( keystorefile ), storepass );

/* Create trust manager to hold server's presented certificates */
TrustManagerFactory tmfactory= TrustManagerFactory.getInstance("SunX509");
tmfactory.init(keystore);
TrustManager[] clienttm= tmfactory.getTrustManagers();

/* Create SSL context */
SSLContext sslcontext= SSLContext.getInstance("SSL");
sslcontext.init(null, clienttm, null); /* default sec random and km */

/* Create socket factory */
SocketFactory   sfactory  = sslcontext.getSocketFactory();
// [next slide ...]
```

A Client: Using

Public Key Crypto

Secure Sockets Layer

SSL

▷ JSSE

```
     // [... contd]
     /* open an SSL socket on host port 5999 */
     SSLSocket s= (SSLSocket) sfactory.createSocket(”localhost”,5999);
     DataOutputStream out = new DataOutputStream(s.getOutputStream());

     out.writeUTF(”Hi␣there”);
     out.flush();
     }
}
```

Running the Example

Since Alice's certificate is not signed by a CA (that we trust), we manually import Alice's self-signed cert into Bob's keystore, where it will be considered trusted by the TrustManager.

```
> keytool -export -keystore alicekeystore -alias AlicesDSA -file alice.cer
Enter keystore password:  spasswd
 Certificate  stored  in  file  <alice.cer>
> keytool -import -keystore bobkeystore -alias alicesDSA -file alice.cer
Enter keystore password:  spasswd
Owner: CN=Alice Smith, OU=Department of Computer Science,
            O=University College Cork, L=Cork, ST="␣", C=IE
 Issuer :  CN=Alice Smith, OU=Department of Computer Science,
            O=University College Cork, L=Cork, ST="", C=IE
 Serial  number: 3c4db510
Valid  from:  Tue Jan 22 18:53:04 GMT+00:00 2002 until: Mon Apr 22 18:53:04 GMT-
 Certificate   fingerprints :
         MD5: 57:E1:20:D3:BF:60:48:3E:59:E3:04:16:73:DD:5F:9E
         SHA1: 6D:31:99:72:35:36:68:6E:29:E4:EC:BA:47:D8:F2:A2:08:DC:48:05
Trust  this   certificate ? [no]:   yes
 Certificate  was added to keystore
```

Server Alice Execution Highlights

```
> java -Djavax.net.debug=ssl ServerAlice > Alice.trace
found key for : alicesdsa
 trustStore is: /usr/local/IBMJava2-13/jre/lib/security/cacerts
 trustStore type is : jks
 init truststore
adding as trusted cert: [
 [... a series of well known CA certs ...]
 trigger seeding of SecureRandom
done seeding SecureRandom
matching server alias : alicesdsa
 Finalizer, SEND SSL v3.1 ALERT: warning, description = close_notify
 Finalizer, WRITE: SSL v3.1 Alert, length = 2
 [...]
Cipher Suites: { 0, 5, 0, 4, 0, 9, 0, 10, 0, 18, 0, 19, 0, 3, 0, 17 }
 [...]
*** ServerHello, v3.1
 [...]
Cipher suite: SSL_DHE_DSS_WITH_DES_CBC_SHA
 [...]
```

```
*** Certificate chain
chain [0] = [
[
  Version: V1
  Subject: CN=Alice Smith, OU=Department of Computer Science, O=University College Cork, L=Cork, S
  Signature Algorithm: SHA1withDSA, OID = 1.2.840.10040.4.3

  Key:  Sun DSA Public Key
    Parameters:DSA
          [...]
  Validity : [From: Tue Jan 22 18:53:04 GMT+00:00 2002,
               To: Mon Apr 22 18:53:04 GMT+00:00 2002]
  Issuer : CN=Alice Smith, OU=Department of Computer Science, O=University College Cork, L=Cork, ST
  SerialNumber: [    3c4db510 ]
]
  Algorithm: [SHA1withDSA]
  Signature:  [...]
]
***
*** Diffie-Hellman ServerKeyExchange
DH Modulus = { 0, 244, 136, 253, 88, 78, 73, 219, 205, 32, 180, 157, 228, 145, 7, 54,
 [...]
*** ClientDiffieHellmanPublic
DH Public key = { 0, 151, 17, 70, 10, 183, 30, 255,
 [...]
```

```
SESSION KEYGEN:
PreMaster Secret:
0000: 6B 73 A6 93 39 91 DA E5  85 E0 3A 76 55 5F 0C 05 ks  ..9.....:  vU␣..
 [...]
CONNECTION KEYGEN:
 Client  Nonce:
0000: 3C 4D B9 3B 98 5E 47 D5 42 7B 62 42 13 98 03 41  <M.;.^G.B.bB...A
0010: 6B 3A FF 1C F8 D8 07 9B 6E D8 49 86 07 22 BC AA k :...... n.I ..” ..
Server␣Nonce:
0000:␣3C␣4D␣B9␣3B␣9C␣6D␣0C␣89␣␣␣9D␣40␣5A␣8D␣DA␣97␣73␣8B␣␣<M.;.m...@Z...s.
0010:␣16␣21␣F2␣32␣F2␣9D␣CB␣47␣␣␣7A␣62␣E3␣B3␣6A␣D6␣BB␣F4␣␣.!.2...Gzb..j...
Master␣Secret:
0000:␣8A␣2E␣4A␣51␣77␣CA␣12␣73␣␣␣76␣4C␣C1␣87␣E6␣BD␣A1␣B1␣␣..JQw..svL......
0010:␣36␣EF␣84␣6A␣95␣65␣D2␣8D␣␣␣44␣0C␣C1␣6B␣80␣0E␣2E␣6D␣␣6..j.e..D..k...m
0020:␣25␣8A␣68␣44␣54␣8E␣1D␣D7␣␣␣D8␣B3␣46␣21␣65␣84␣0C␣3D␣␣%.hDT.....F!e..=
 Client␣MAC␣write␣Secret:
0000:␣AE␣1E␣05␣7C␣A5␣C7␣6C␣C8␣␣␣B3␣1C␣BF␣70␣38␣1E␣8B␣AF␣␣......l....p8...
0010:␣90␣5B␣9B␣D6␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣.[..
Server␣MAC␣write␣Secret:
0000:␣63␣AC␣61␣1A␣EC␣C4␣E1␣A1␣␣␣E9␣17␣34␣53␣44␣F8␣C0␣63␣␣c.a.......4SD..c
0010:␣00␣2F␣6F␣36␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣./o6
 Client␣write␣key:
0000:␣8F␣B1␣91␣EF␣B8␣5D␣FC␣62␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣....].b
Server␣write␣key:
0000:␣5A␣5B␣EB␣2C␣8C␣88␣1A␣60␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣Z[.,...‘
```

```
Client  write  IV:
0000: C9 A3 2E A5 D7 53 81 1B                                  ..... S..
Server  write  IV:
0000: E0 15 F9 4E B5 9F 9B C3                                  ... N ....
[read]  MD5 and SHA1 hashes: len = 135
0000: 10 00 00 83 00 81 00 97    11 46 0A B7 1E FF E2 6D   .........  F ..... m
0010: DA 42 BE 80 9C 92 78 EC F6 9D 7B 37 94 CB E9 90 .B .... x  ....7....
0020: 61 16 F4 5A F2 E7 01 FC  F9 91 A2 70 C3 2D F5 6C a..Z  .......  p.−.l
0030: E7 E2 2F 72 93 4B 08 3B  F9 22 5B FF 00 45 C7 1E ../ r.K.;." [.. E..
0040:␣BB␣25␣A6␣8B␣76␣D9␣56␣00␣␣␣D7␣EE␣7F␣8B␣96␣5C␣52␣AF␣␣.%..v.V......\R.
0050:␣2B␣23␣40␣60␣D4␣C0␣AC␣30␣␣␣0A␣A6␣C9␣F7␣BE␣15␣27␣30␣␣+#@‘...0......’0
0060:␣12␣1A␣09␣46␣37␣C5␣57␣26␣␣␣07␣9C␣9A␣16␣45␣F7␣98␣8C␣␣...F7.W&....E...
 [...]
Padded␣plaintext␣before␣ENCRYPTION:␣␣len␣=␣40
0000:␣14␣00␣00␣0C␣0B␣28␣B1␣99␣␣␣D0␣B4␣93␣CD␣55␣3A␣24␣56␣␣.....(......U:$V
0010:␣58␣AC␣72␣AD␣61␣3A␣4F␣BA␣␣␣37␣04␣34␣FF␣63␣5F␣72␣D4␣␣X.r.a:O.7.4.c_r.
0020:␣62␣89␣87␣DA␣03␣03␣03␣03␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣b.......
main,␣WRITE:␣␣SSL␣v3.1␣Handshake,␣length␣=␣40
%%␣Cached␣server␣session:␣[Session−1,␣SSL_DHE_DSS_WITH_DES_CBC_SHA]
main,␣READ:␣␣SSL␣v3.1␣Application␣Data,␣length␣=␣32
Padded␣plaintext␣after␣DECRYPTION:␣␣len␣=␣32
0000:␣00␣08␣48␣69␣20␣74␣68␣65␣␣␣72␣65␣2C␣58␣88␣E6␣AD␣5F␣␣..Hi␣there,X..._
0010:␣EE␣5E␣CD␣BC␣40␣91␣3A␣EE␣␣␣11␣5B␣4D␣1A␣EB␣C8␣01␣01␣␣.^..@.:..[M.....
```