My Locker, Your Locker, Our Locker – How to Manage Keys and Certificates

Pankaj Kumar

Session Objectives

Learn about secure communication, keys and certificates and how to make them work.

Who am I?

- Not a security expert
- An enterprise application developer
- Worked for VeriFone Payment processing software, HP e-speak, HP Middleware and now HP Management Software
- Co-author of WSMF specifications, a submission to WSDM, a TC of OASIS
- Author of *J2EE Security for Servlets, EJBs and Web Services*

Session Outline

- Warmup
- User perspective of secure communication
- Keys, Certificates and Certificate Stores
- Making things work

Security warm-up Quiz

- To secure your communication, you should
  - A) Use a firewall
  - B) Promptly apply security patches to all your systems
  - C) Use a secure protocol such as SSL or SSH
  - D) Encrypt your data

Answer: (C)

Security warm-up Quiz (Contd.)

- SSL provides security by
  - A) encrypting data packets
  - B) adding Message Authentication Code (MAC)
  - C) mandatory authentication of the server by the client
  - D) mandatory authentication of the client by the server

Answer: A, B and C

Security warm-up Quiz (Contd.)

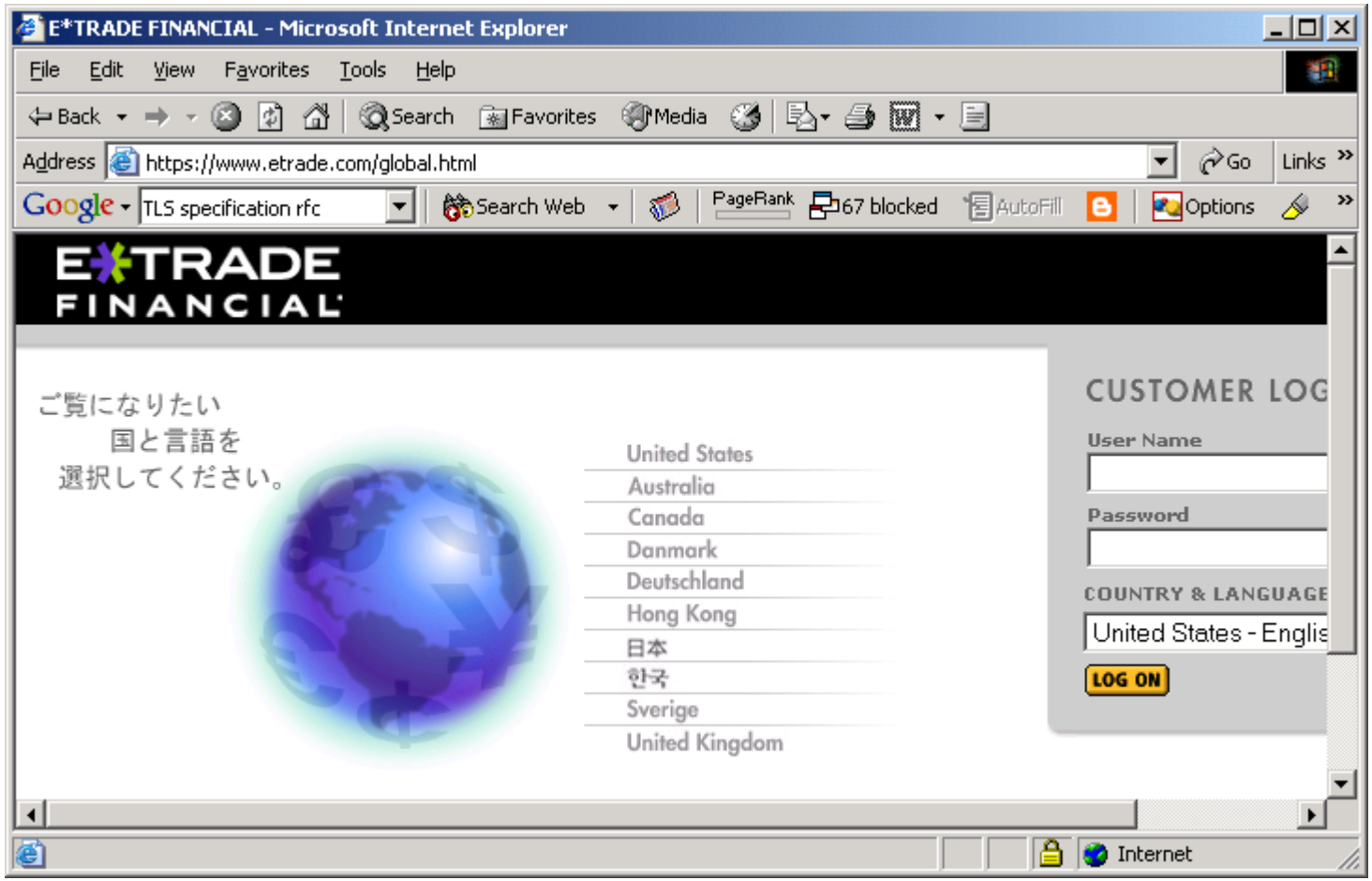
- Alice wants to send a private message to Bob with the assurance that the message was sent by her and no one else. She should
  - A) digitally sign the message with her private key
  - B) encrypt the message with a secret key
  - C) encrypt the message with Bob's public key
  - D) sign the message with her private key and then encrypt it with Bob's public key

Answer: D

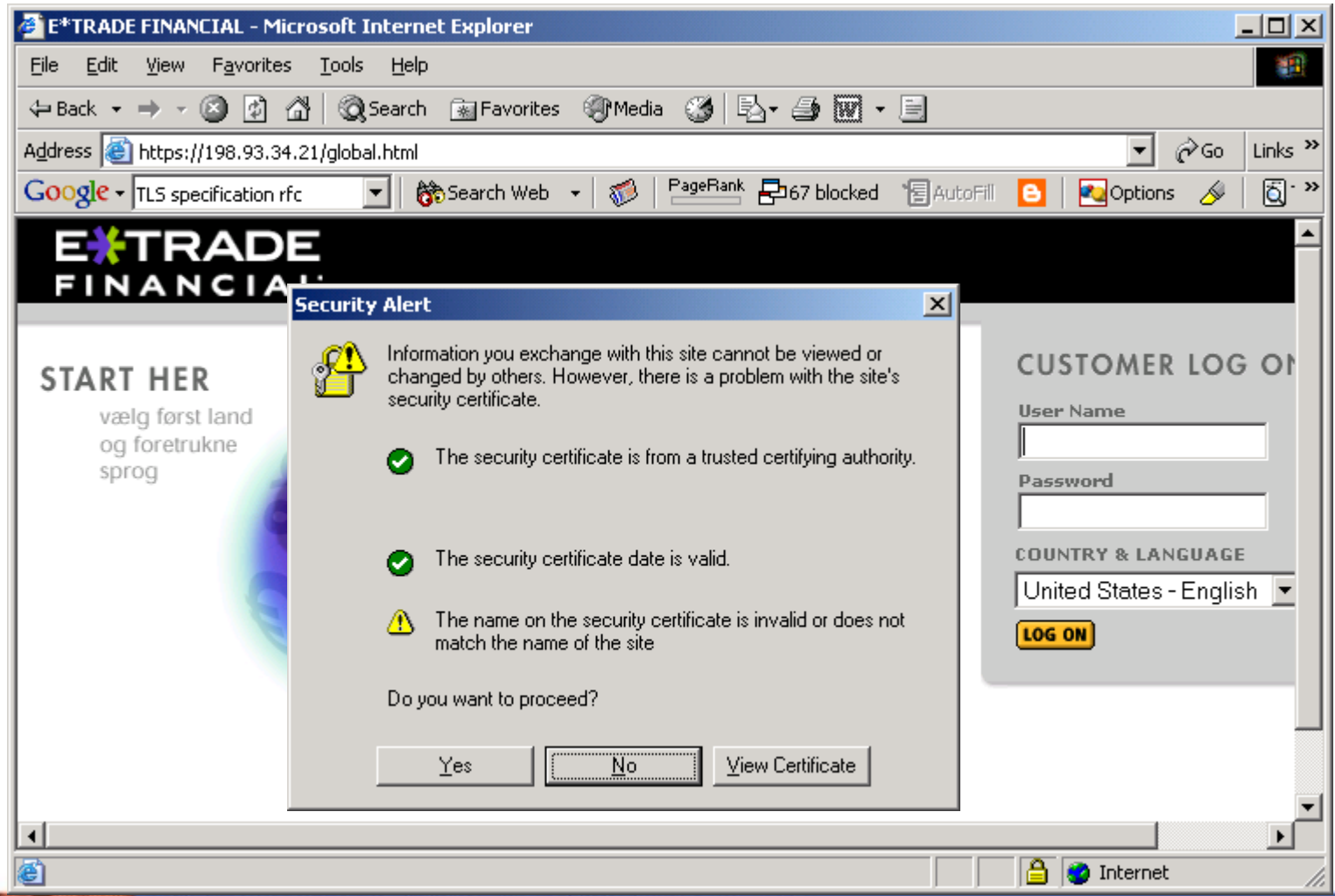
User Perspective of Secure Communication

- Accessing SSL protected website from a Browser
- Establishing SSH session to a remote computer
- Sending signed and encrypted email
- Exchange of WS-Security protected SOAP messages
- ...

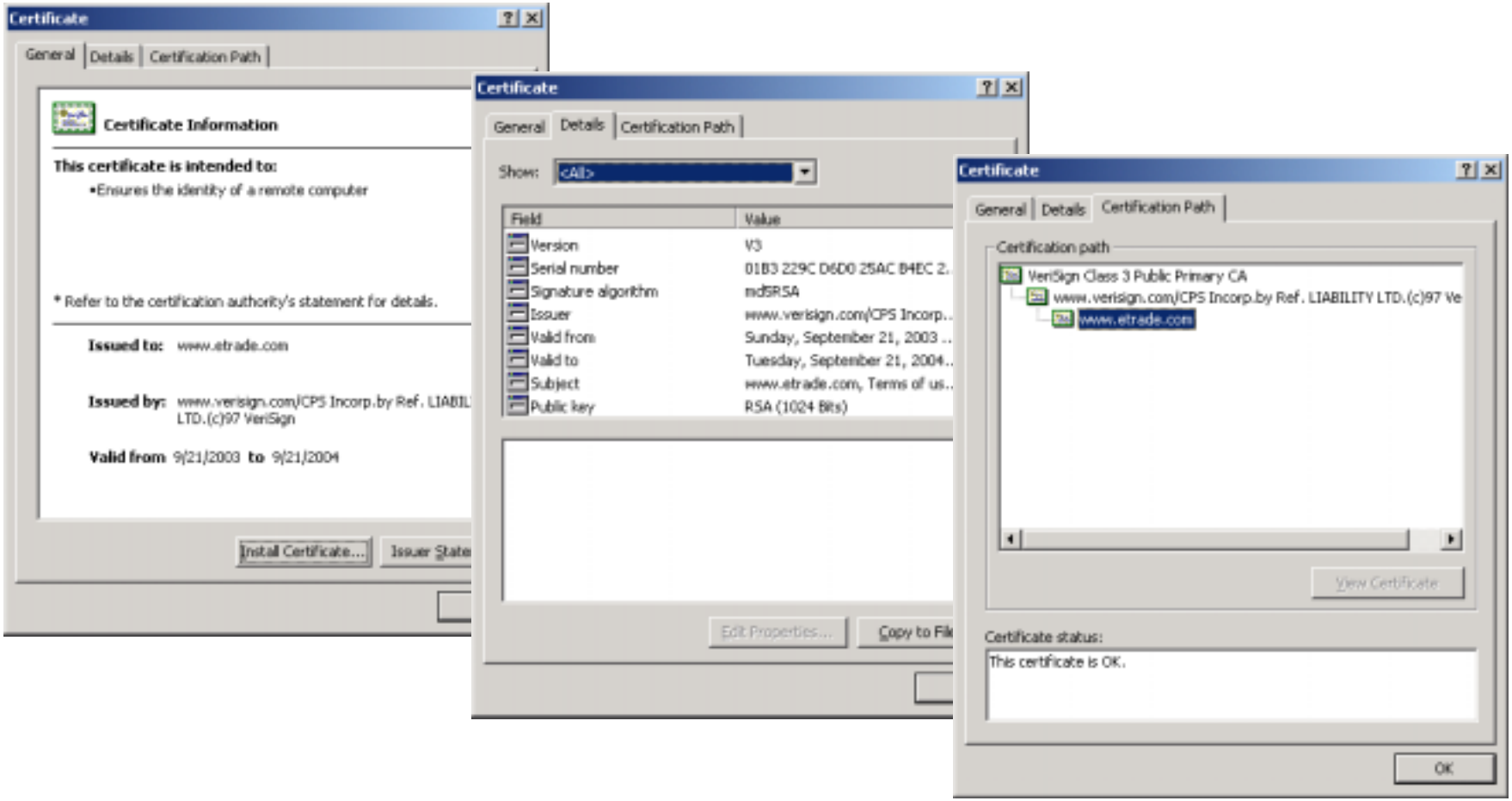
Accessing SSL protected website -- Success



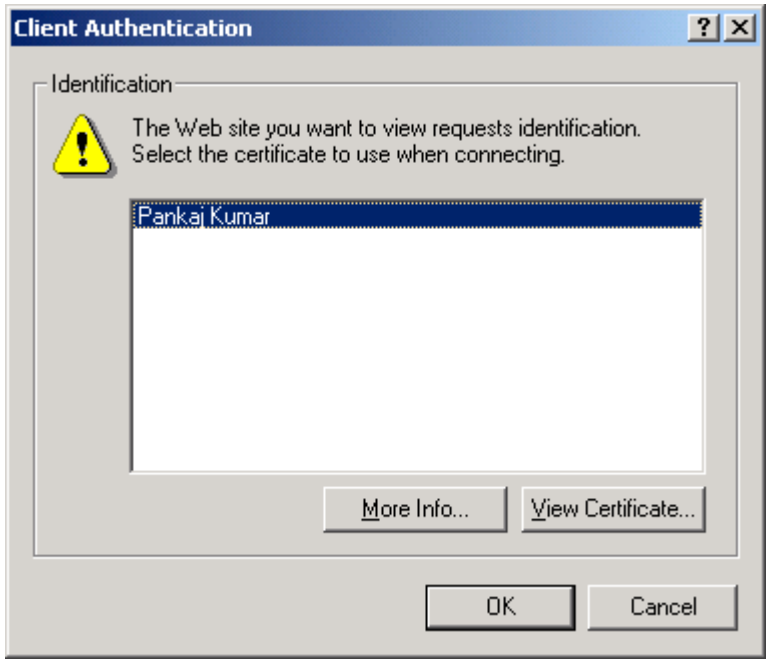
Accessing SSL protected website – Security Alert



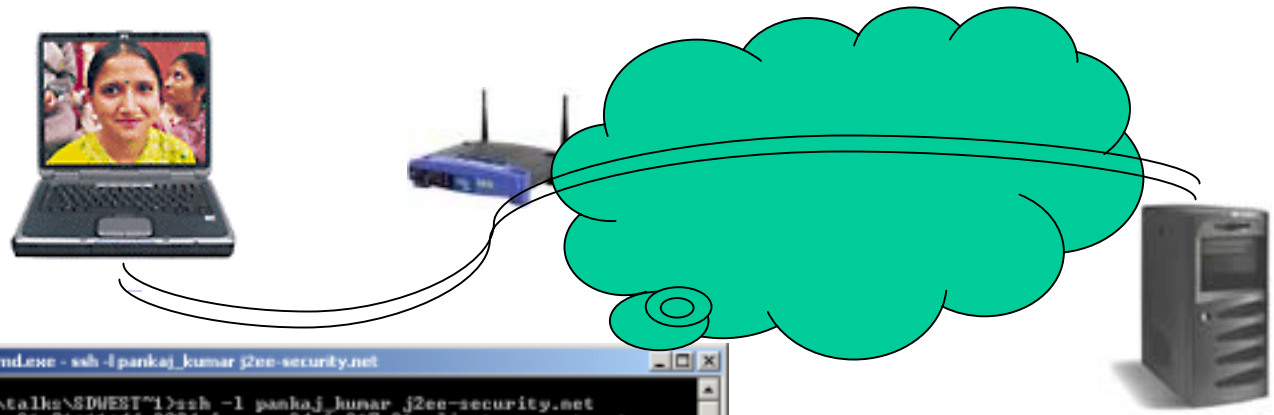
Examine the Server Certificate



Website Requiring Client Certificate

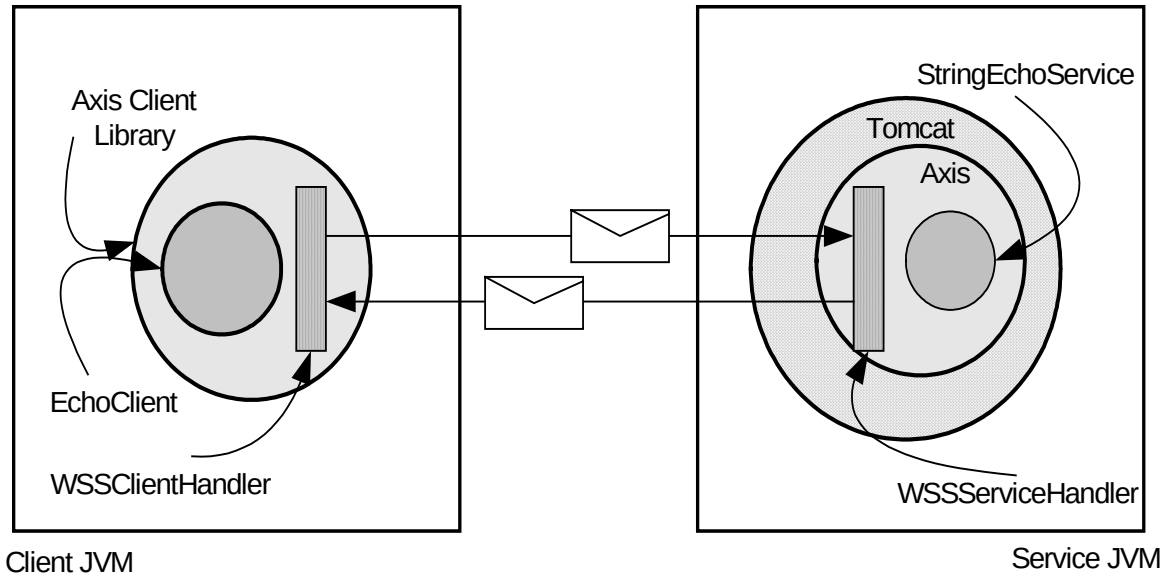


SSH Session between two computers



```
C:\WINNT\system32\cmd.exe - ssh -l pankaj_kumar j2ee-security.net

C:\pankaj\projects\talks\SDWEST"1>ssh -l pankaj_kumar j2ee-security.net
Last login: Wed Jan 21 01:11:44 2004 from c-24-6-217-81.client.comcast.net

This computer system is for official use only.
By accessing this system you consent to monitoring.
Any unauthorized access will not be tolerated.

[pankaj_kumar@www7 pankaj_kumar]$ _
```

Exchange of WS-Security protected SOAP messages

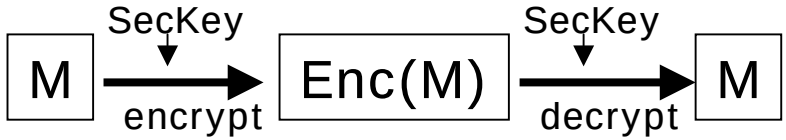


Keys, Certificates and Certificate Stores

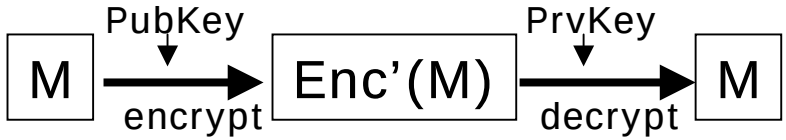
- Recap of cryptographic operations
- Key-pairs
- X.509 certificates, Certificate Signing Request, cert-chains
- Certificate Stores
- PKCS standards: PKCS1, PKCS8, PKCS7, PKCS10, PKCS12

Cryptographic Operations

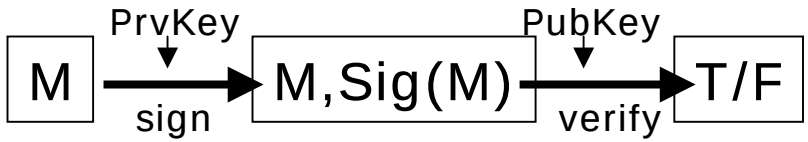
Secret Key Cryptography ----->
(DES, TripleDES, AES, ...)



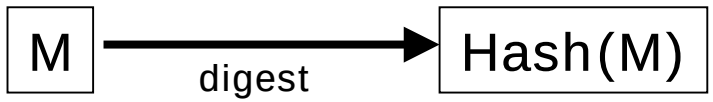
Public Key Cryptography ------>
(RSA, ...)



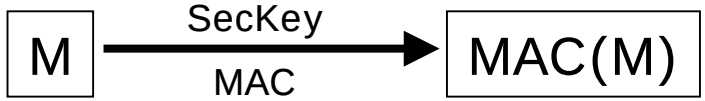
Digital Signature ---------->
(RSA, DSA, ...)



One-way Hash (Digest) ------->
(SHA-1, MD5, SHA-256, ...)



Message Authentication Code ->
(HMAC-SHA1, ...)



Storage format for Cryptographic Content

- Structure of the Cryptographic content is define in PKCS series and a number of IETF RFCs
- Syntax of the format is specified in ASN.1
- The data bytes are typically encoded using DER (Distinguished Encoding Rules)
- Binary files may be converted to BASE64 encoding with suitable header and footer. This format is known as PEM (Privacy Enhanced Mail).

A brief note on tools

- J2SE SDK tools: primarily keytool
- JSTK (opensource software available from http://www.j2ee-security.net and authored by yours truly) utilities: crypttool, asn1parse and certtool
- OpenSSL tools available from http://www.openssl.org : openssl

Working with Key-pairs

- API to generate, store and load keys (Java)
  - `java.security.KeyPairGenerator`
  - `java.security.KeyPair`
  - `java.security.PublicKey`
  - `java.security.PrivateKey`
  - …
- Format to store and exchange
  - PKCS8 (for private key), X.509
  - PKCS1 (for RSA keys)

key-pair: Java API

Generate RSA key-pair

```
int keysize = 512;
...
KeyPairGenerator kpg = KeyPairGenerator.getInstance(“RSA”);
kpg.initialize(keysize);
KeyPair kp = kpg.generateKeyPair();
PublicKey pubKey = kp.getPublic();
PrivateKey prvKey = kp.getPrivate()
byte[] x509Bytes = pubKey.getEncoded();
byte[] pkcs8Bytes = prvKey.getEncoded();
```

Restore RSA keys from encoded Bytes

```
PKCS8EncodedKeySpec prvKS = new PKCS8EncodedKeySpec(pkcs8Bytes);
X509EncodedKeySpec pubKS = X509EncodedKeySpec(x509Bytes);
KeyFactory kf = KeyFactory.getInstance("RSA");
PrivateKey prvKey = kf.generatePrivate(prvKS);
PublicKey pubKey = kf.generatePublic(prvKS);
```

Private key: PKCS8 Format on disk/wire

PKCS8 Syntax Definition

```
PrivateKeyInfo ::= SEQUENCE {
   version INTEGER,
   privateKeyAlgorithm AlgorithmIdentifier ,
   privateKey OCTET STRING,
   attributes [0] IMPLICIT SET OF Attribute OPTIONAL
}
```

Structure of encoded bytes

```
 0:d=1  hl=4 l= 342 cons: SEQUENCE
 4:d=1  hl=2 l=   1 prim: INTEGER   :0
 7:d=2  hl=2 l=  13 cons: SEQUENCE
 9:d=2  hl=2 l=   9 prim: OBJECT    :1.2.840.113549.1.1.1
20:d=2  hl=2 l=   0 prim: NULL
22:d=1  hl=4 l= 320 prim: OCTET STRING
```

Private Key in PEM (BASE64) format

```
-----BEGIN RSA PRIVATE KEY-----
MIIBPAIBAAJBAOkW3MCzC4gttu82yaTRsIalTrBqElXBLcl2r3QMfDOAN8dqISU3
pa7h2FzlsA8xuTLcD0wViviMi9IbdEZk08sCAwEAAQJBANaaIQceEakAheP6qm6g
TQm62xcboePoZzKMnn+XGycBzLN0uwl0OoNZ3JsAImnCubf7Z9mPob1E1ghap5CL
AEkCIQD97E4XpKSgH2hC6ZNcYGmfAGJq+zwkOXzn2rtM1FUttQIhAOr+7tmumih3
hfj/VxKc61goosVxY37d0XVRK1mPNut/AiEAz1r/UpNi1FeRfsLbUZ5x4iQgOpi7
PiGr/HJEbbMfc9kCIQC0xskXzKnq14fItSVWvECP4TLWkRPpVNr9VSP0Eag+IQIg
WJyLpCXS/BOoJ99IW0Gy5CnfT0JSt1KqruiEkFlJ1JQ=
-----END RSA PRIVATE KEY-----
```

Encrypted private key: PKCS8 Format on disk/wire

PKCS8 Syntax Definition

```
EncryptedPrivateKeyInfo ::= SEQUENCE {
   encryptionAlgorithm AlgorithmIdentifier,
   encryptedData OCTET STRING
}
```

PrivateKeyInfo

Structure of encoded bytes

```
 0:d=0  hl=4 l= 385 cons: SEQUENCE
 4:d=1  hl=2 l=  27 cons: SEQUENCE
 6:d=2  hl=2 l=   9 prim: OBJECT          :pbeWithMD5AndDES-CBC
17:d=2  hl=2 l=  14 cons: SEQUENCE
19:d=3  hl=2 l=   8 prim: OCTET STRING
29:d=3  hl=2 l=   2 prim: INTEGER         :0800
33:d=1  hl=4 l= 352 prim: OCTET STRING
```

Private Key in PEM (BASE64) format

```
-----BEGIN RSA PRIVATE KEY-----
Proc-Type: 4,ENCRYPTED
DEK-Info: DES-CBC,8458815C662CA54A

TEtCNd5FbptsbXqY2Q8TUpXGjYKyMEQKv+nszpiW8y9sIj/rsomqOIQwyIrepnB/
+Ws63MpSZPqj+6jEqqy/m8bhv8SvHfoNOL3qXck/2HZvAlQXYBWd/+I44sQL3By5
7hhH7hi8+Rs1fBN7gSznvw7mqum5tmHQbzhnZeRqm9h4bdBQhRyVYEMstUojFhGF
HeUD6LIWyIVhLd3nh1/zVMA0h7H3XQFCheLAVpUdLHRPHJKMWJH1iN6yeyMq23xe
PDrPfGHmRREddeijruNFbay3uD1d0hL/77xKVv7HTDDbpxFOIGDVmToxvK1xmftw
pJOLcH3DQT/h+Ef4A6acJ3j29kzeckAQrQNBvUZM9oBrCcK5jiNULdtAI2dM8Jb2
hYKxgkg2DNz4xmlKMz48/2V0mBVwD4bX3v/wTxf0clE=
-----END RSA PRIVATE KEY-----
```

RSA Keys: PKCS1 Format on disk/wire

```
RSAPublicKey ::= SEQUENCE {
    modulus           INTEGER,  -- n
    publicExponent    INTEGER   -- e
}
```

PublicKey can be obtained from Private Key

```
RSAPrivateKey ::= SEQUENCE {
    version           Version,
    modulus           INTEGER,  -- n
    publicExponent    INTEGER,  -- e
    privateExponent   INTEGER,  -- d
    prime1            INTEGER,  -- p
    prime2            INTEGER,  -- q
    exponent1         INTEGER,  -- d mod (p-1)
    exponent2         INTEGER,  -- d mod (q-1)
    coefficient       INTEGER,  -- (inverse of q) mod p
    otherPrimeInfos   OtherPrimeInfos OPTIONAL
}
```

Note: You convert PKCS1 formatted RSA keys to PKCS8 format and vice-versa.

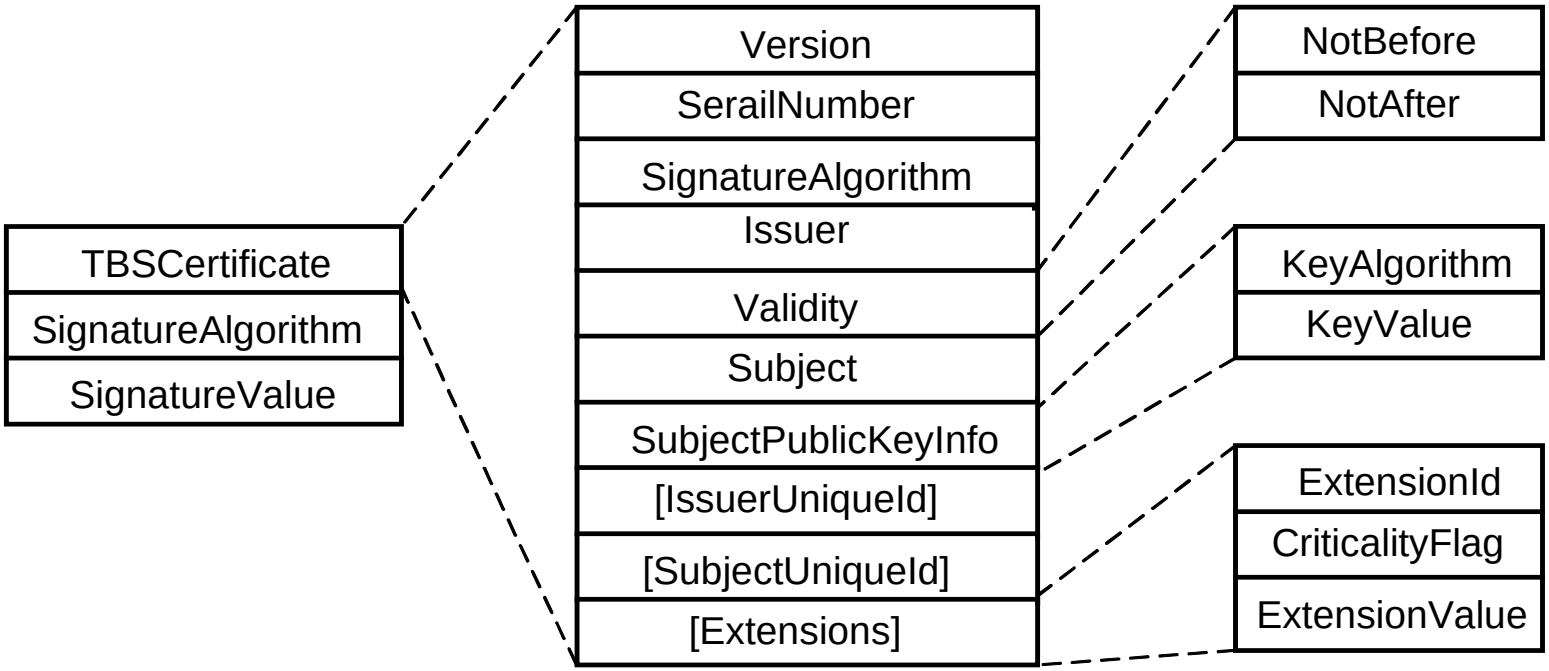
Working with Key-pairs

```
C:\jstk>bin\crypttool genkp -algorithm RSA -action save
KeyPair written to files pubkey.x509b(public) and prvkey.p8b(private).

C:\jstk>bin\crypttool printk -algorithm RSA -file pubkey.x509b -format x509
PublicKey::
Modulus        : 12157350124775005647478041733747643619010134972705559351385884
7
23933656361057507268735414407860625330152695721284483638067076097105791581892097
011783941957
PublicExponent : 65537

C:\jstk>bin\asn1parse prvkey.p8b
    0:d=1  hl=4 l= 341 cons: SEQUENCE
    4:d=1  hl=2 l=   1 prim: INTEGER           :0
    7:d=2  hl=2 l=  13 cons: SEQUENCE
    9:d=2  hl=2 l=   9 prim: OBJECT            :1.2.840.113549.1.1.1
   20:d=2  hl=2 l=   0 prim: NULL
   22:d=1  hl=4 l= 319 prim: OCTET STRING

C:\jstk>openssl pkcs8 -in prvkey.p8b -inform DER -out prvkey.p1 \
-outform PEM –nocrypt

C:\jstk>openssl rsa -in prvkey.p1 -inform PEM -modulus
Modulus=E81FE3BAB9B88C8ED0073BBABE7C5E3B8200ADF3C99C1461B26B17FBE4815A49112BEDDA
7AF0F52170D0F1EE1E792624CBEC672389C3B26D594938B32BE5BF45
writing RSA key
-----BEGIN RSA PRIVATE KEY-----
MIIBOwIBAAJBAOgf47q5uIyO0Ac7ur58XjuCAK3zyZwUYbJrF/vkgVpJESvt2nrw
... Skipped ...
Km8G4uw+URUF9PHFawP/X1FyZcktQ6OrsDB/up/qqw==
-----END RSA PRIVATE KEY-----
```

Working with Key-pairs (Contd.)

```
C:\jstk>openssl genrsa -out prv.p1
Generating RSA private key, 512 bit long modulus
...++++++++++++
.........++++++++++++
e is 65537 (0x10001)

C:\jstk>openssl asn1parse -in prv.p1
    0:d=0  hl=4 l= 315 cons: SEQUENCE
    4:d=1  hl=2 l=   1 prim: INTEGER           :00
    7:d=1  hl=2 l=  65 prim: INTEGER
:BF638B16AC256DA4313D23CCCE6DEAE9B2B4B63F61224EFFD54C25F7DB5388B003214AB7EFAAD2DE0F630DE2B1
768ECC76BEF43DB10D98095295A5952D2D8E55
   74:d=1  hl=2 l=   3 prim: INTEGER           :010001
... Skipped ...

C:\jstk>openssl pkcs8 -in prv.p1 -inform PEM -topk8 -out prv.p8b \
-outform DER –nocrypt

C:\jstk>bin\crypttool printk -algorithm RSA -format pkcs8 –file prv.p8b

PrivateKey::
Modulus         : 10023846957195057153890586429447325605539500628011697687878912l
261915445370866957697042961910793864293371060015020284049613321304663990724933603087224377l7
PrivateExponent: 57827121436384137338342507549101192684168857247641220407300595O
32889805126595076303635572358650402749107015556125342708173068996901499696927637370187628l7
PublicExponent : 65537
CRTCoefficient : 10587002017550265734373854127125829266168482628376245511534575467451235102769l
PrimeExponentP : 10878419758742436130315282666976143778521318550263045250719793528445167071171l
PrimeExponentQ : 36899381609860423230981742929419947471258965250351522095076650968072423196019
PrimeP          : 11485492818595895746499648480750334844665984467136906490687144305474375563342l
PrimeQ          : 87273982192263255883967320594947313054383009333151239797359647569185549785179
```

X.509 Certificates: Tying additional info. with a public key



Self-signed certificates ==> Issuer is same as the Subject

Certificate Stores

- A Certificate Store contains
  - Certificates of trusted CAs
  - Own certificates (along with private keys)
  - Certificates of others
- Java supports JKS and JCEKS certificate stores. Also has read only support for PKCS12. Includes Java APIs and CLI utility keytool.
- Windows and Netscape browser have their own proprietary format with visual I/f.

Self-signed Certificate with keytool

```
C:\jstk>keytool -genkey -keystore test.ks -alias testcert -keyalg rsa
Enter keystore password:  changeit
What is your first and last name?
  [Unknown]:  Test User
What is the name of your organizational unit?
  [Unknown]:  Test Unit
What is the name of your organization?
  [Unknown]:  Test Org
What is the name of your City or Locality?
  [Unknown]:  Test Land
What is the name of your State or Province?
  [Unknown]:  Test State
What is the two-letter country code for this unit?
  [Unknown]:  Test Country
Is CN=Test User, OU=Test Unit, O=Test Org, L=Test Land, ST=Test State, C=Test Co
untry correct?
  [no]:  yes

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

C:\jstk>keytool -export -keystore test.ks -alias testcert -storepass changeit \
-file testcert.pem -rfc
Certificate stored in file <testcert.pem>
```

Looking at the Certificate

```
C:\jstk>type testcert.pem
-----BEGIN CERTIFICATE-----
MIICaDCCAdECBEAUCmEwDQYJKoZIhvcNAQEEBQAwezEVMBMGA1UEBhMMVGVzdCBDb3VudHJ5MRMw
EQYDVQQIEwpUZXN0IFN0YXRlMRIwEAYDVQQHEwlUZXN0IExhbmQxETAPBgNVBAoTCFRlc3QgT3Jn
MRIwEAYDVQQLEwlUZXN0IFVuaXQxEjAQBgNVBAMTCVRlc3QgVXNlcjAeFw0wNDAxMjUxODI2NDFa
Fw0wNDA0MjQxODI2NDFaMHsxFTATBgNVBAYTDFRlc3QgQ291bnRyeTETMBEGA1UECBMKVGVzdCBT
dGF0ZTESMBAGA1UEBxMJVGVzdCBMYW5kMREwDwYDVQQKEwhUZXN0IE9yZzESMBAGA1UECxMJVGVz
dCBVbml0MRIwEAYDVQQDEwlUZXN0IFVzZXIwgZ8wDQYJKoZIhvcNAQEBBQADgY0AMIGJAoGBAOKl
IJDRg/QdOhlOJ/mI0PE54KogK2i+INnSGWgcdLzTpM8suXSQBN9UnCrxuCunkwqeo+QtgHYBWZ2d
in7XxNczSPgErLYA+QRNLZ2VyJ3wS/E6nOvGwSQTeKo0dt+7TuMdqAoNjUHERhkkzCd/fxsqYBUB
KLQ/Ds+pSS9XiNtJAgMBAAEwDQYJKoZIhvcNAQEEBQADgYEAyGN7X6K0MuYyWf9osxcjnIummopg
jiX/jncSghqxZ27tFJTXj1CznXZeZvbhfvIPTElif+ZjH/TVUa8cbDvv7ms1HchgRyuXOz4F4OYs
RHT92LsdQbnlbMoUrtgYoI2RMmSd+He18r5iscc2A6DtOx23uYf2qhJBTjnzk8iPRRU=
-----END CERTIFICATE-----

C:\jstk>keytool -printcert -v -file testcert.pem
Owner: CN=Test User, OU=Test Unit, O=Test Org, L=Test Land, ST=Test State, C=Tes
t Country
Issuer: CN=Test User, OU=Test Unit, O=Test Org, L=Test Land, ST=Test State, C=Te
st Country
Serial number: 40140a61
Valid from: Sun Jan 25 10:26:41 PST 2004 until: Sat Apr 24 11:26:41 PDT 2004
Certificate fingerprints:
         MD5:  45:D0:8B:FB:8A:4E:28:7E:A7:66:75:67:5B:26:7E:F7
         SHA1: C1:DE:CC:AA:57:36:B8:20:63:5F:33:54:EB:5D:E4:8F:3B:C8:EF:96
```

Exercise: Produce the ASN.1 parse of a self-signed certificate.

Establishing Trust

- Bob doesn't know Alice. He cannot trust any information she gives him.
- Many people, including Bob, trust Charlie to verify identity and other info. and issue unforgeable document.
- Alice shows a document issued by Charlie and *proves* that the document was issued to her.
- Bob can now trust Alice to be Alice and all the information contained in the document.

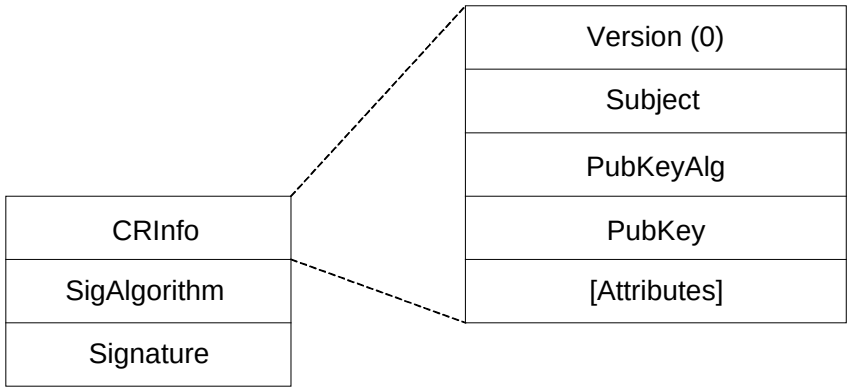
Establishing Trust through PKI

- In PKI terminology, Charlie is the **CA**, Alice is the **Subject**, and Bob is the **Relying Party**.
- The unforgeable document is a CA signed **X.509 certificate**.
- The mechanism to prove that the certificate belongs to someone is to perform an operation that requires possession of the private key.

PKI Processes

- CA gets its self-signed cert to interested parties (distribution as part of OS, browser, JRE, ...)
- The subject forms a Certificate Signing Request (CSR) and sends it to the CA
- The CA creates a signed certificate and makes it available to the subject
- The subject stores the certificate along with its private key for use by software applications
- The CA makes available a list of revoked certificates in Certificate Revocation List (CRL)

Certificate Signing Request



- PKCS10 defines the syntax for Certificate Signing Request
- Designed to minimize request size
- Use of signature provides security against impersonation
- This differs from the format for CSR defined in RFC 1424

CSR from self-signed cert. Using keytool

C:\jstk>**keytool -certreq -keystore test.ks -alias testcert -storepass changeit \ -file testcert.csr**

C:\jstk>**type testcert.csr**

```
-----BEGIN NEW CERTIFICATE REQUEST-----
MIIBuzCCASQCAQAwezEVMBMGA1UEBhMMVGVzdCBDb3VudHJ5MRMwEQYDVQQIEwpUZXN0IFN0YXRl
... Skipped ...
aSTt+TvFGBSOJRJ92WchHY9fvu0cwxDow7WdB3luz6UqLzN4vg6I/bzsgnIw60F1
-----END NEW CERTIFICATE REQUEST-----
```

C:\jstk>**bin\asn1parse testcert.csr**

```
   0:d=1  hl=4 l= 443 cons: SEQUENCE
   4:d=2  hl=4 l= 292 cons: SEQUENCE
   8:d=2  hl=2 l=   1 prim: INTEGER           :0
  11:d=3  hl=2 l= 123 cons: SEQUENCE
  13:d=4  hl=2 l=  21 cons: SET
  15:d=5  hl=2 l=  19 cons: SEQUENCE
  17:d=5  hl=2 l=   3 prim: OBJECT            :COUNTRYNAME
  22:d=5  hl=2 l=  12 prim: PRINTABLESTRING   :Test Country
... Skipped ...
 136:d=3  hl=3 l= 159 cons: SEQUENCE
 139:d=4  hl=2 l=  13 cons: SEQUENCE
 141:d=4  hl=2 l=   9 prim: OBJECT            :1.2.840.113549.1.1.1
 152:d=4  hl=2 l=   0 prim: NULL
 154:d=3  hl=3 l= 141 prim: BIT STRING
 298:d=2  hl=2 l=   0 cons: [0]
 300:d=2  hl=2 l=  13 cons: SEQUENCE
 302:d=2  hl=2 l=   9 prim: OBJECT            :1.2.840.113549.1.1.4
 313:d=2  hl=2 l=   0 prim: NULL
 315:d=1  hl=3 l= 129 prim: BIT STRING
```

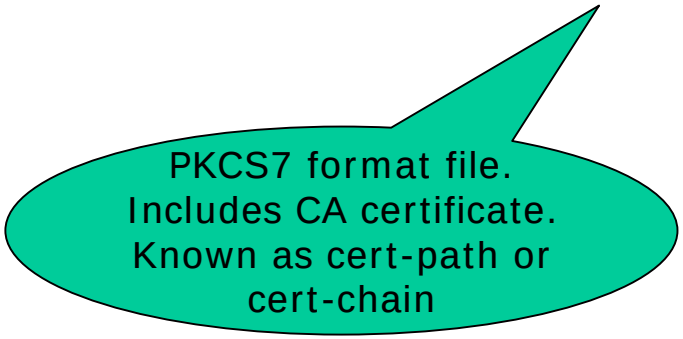
Signing cert with JSTK utility certtool

Setting up a minimal CA (one time operation):

C:\jstk>**bin\certtool setupca -password changeit**
```
CA setup successful: cadir
```

Issuing Signed Certificate:

C:\jstk>**bin\certtool issue -csrfile testcert.csr -cerfile testcert.cer **
-password changeit
```
Issued Certificate written to file: testcert.cer
```



PKCS7 defines ASN.1 syntax and processing steps for cryptographically protected (signed, encrypted, digested, ...) content. Also defines syntax for SET of Certificates and SET of CRLs.

Looking at Cert-Path

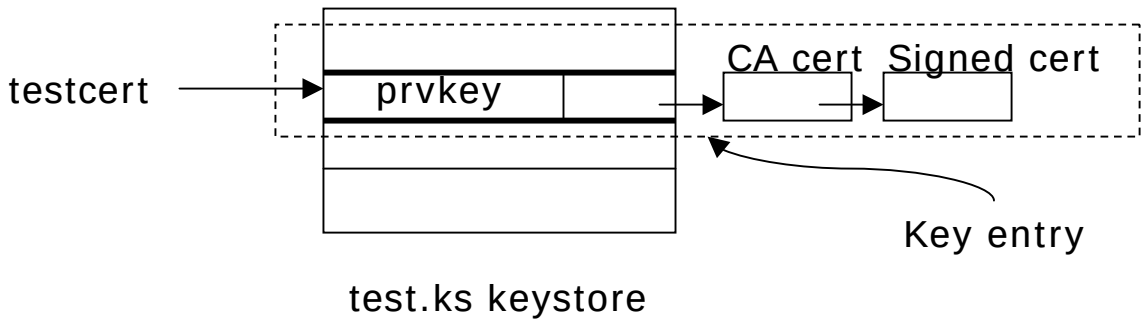
C:\jstk>**bin\certtool show -infile testcert.cer**

```
CertPath:
CertPath Component: 0
  Certificate:
    Data:
      Version: 3
      Serial Number: 1000
      Signature Algorithm: SHA1withRSA
      Issuer: CN=JSTK Test Root CA, OU=JSTK Operations, O=JSTK Inc, C=US
      Validity:
        Not Before: Sun Jan 25 15:19:00 PST 2004
        Not After: Mon Jan 24 15:19:00 PST 2005
      Subject: CN=Test User, OU=Test Unit, O=Test Org, L=Test Land, ST=Test State, C=Test Country
      Extensions:
        X509v3 Basic Constraints:
          CA: FALSE
CertPath Component: 1
  Certificate:
    Data:
      Version: 3
      Serial Number: 100
      Signature Algorithm: SHA1withRSA
      Issuer: CN=JSTK Test Root CA, OU=JSTK Operations, O=JSTK Inc, C=US
      Validity:
        Not Before: Sun Jan 25 15:18:00 PST 2004
        Not After: Sat Oct 21 16:18:00 PDT 2006
      Subject: CN=JSTK Test Root CA, OU=JSTK Operations, O=JSTK Inc, C=US
      Extensions:
        X509v3 Basic Constraints:
          CA: TRUE, pathLen: 2
```

Importing cert-path into keystore

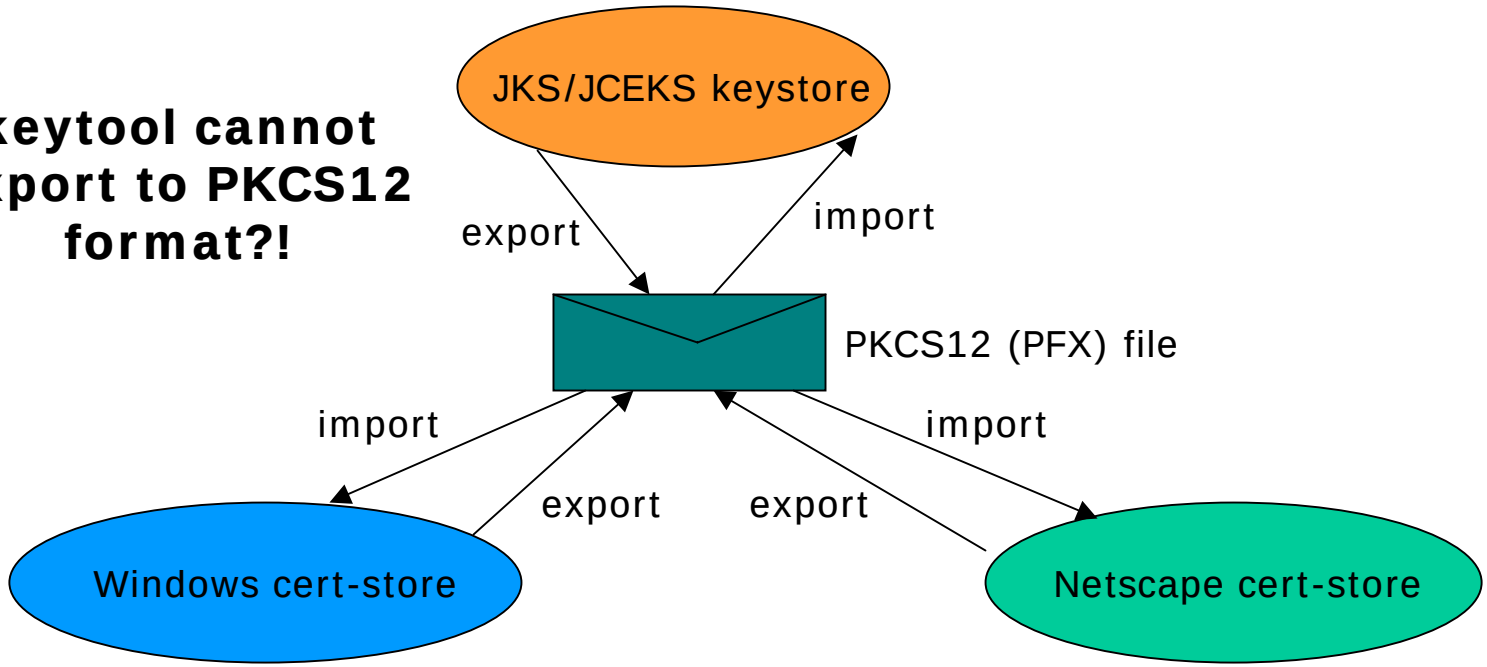
C:\jstk>**keytool -import -keystore test.ks -storepass changeit **
-alias testcert -file testcert.cer

```
Top-level certificate in reply:

Owner: CN=JSTK Test Root CA, OU=JSTK Operations, O=JSTK Inc, C=US
Issuer: CN=JSTK Test Root CA, OU=JSTK Operations, O=JSTK Inc, C=US
Serial number: 64
Valid from: Sun Jan 25 15:18:00 PST 2004 until: Sat Oct 21 16:18:00 PDT 2006
Certificate fingerprints:
         MD5:  6A:FA:E9:B7:C0:F0:79:CB:EB:1A:CE:34:0D:CF:5D:99
         SHA1: A7:57:6D:B8:D1:98:88:28:AB:B5:36:EE:B6:43:81:C5:9C:3F:F4:4D

... is not trusted. Install reply anyway? [no]:  yes
Certificate reply was installed in keystore
```



Moving around contents of a key-entry

keytool cannot export to PKCS12 format?!



Import from JKS/JCEKS to PKCS12 file

C:\jstk>**bin\crypttool export -keystore test.ks -storepass changeit **
-alias testcert -outstore testcert.p12

```
Key entry exported to keystore: testcert.p12
```

C:\jstk>**keytool -list -v -keystore testcert.p12 -storetype PKCS12 **
-storepass changeit -alias testcert

```
Alias name: testcert
Creation date: Jan 27, 2004
Entry type: keyEntry
Certificate chain length: 2
Certificate[1]:
Owner: CN=Test User, OU=Test Unit, O=Test Org, L=Test Land, ST=Test State, C=Tes
t Country
Issuer: CN=JSTK Test Root CA, OU=JSTK Operations, O=JSTK Inc, C=US
Serial number: 3e8
Valid from: Sun Jan 25 15:19:00 PST 2004 until: Mon Jan 24 15:19:00 PST 2005
Certificate fingerprints:
         MD5:  D3:2E:DD:7B:93:FA:B0:84:57:E1:7A:1C:61:81:6F:A3
         SHA1: 4F:CE:F4:F0:37:A7:A1:4F:D5:C0:EB:09:86:48:D0:BB:09:E7:5E:9A
Certificate[2]:
Owner: CN=JSTK Test Root CA, OU=JSTK Operations, O=JSTK Inc, C=US
Issuer: CN=JSTK Test Root CA, OU=JSTK Operations, O=JSTK Inc, C=US
Serial number: 64
Valid from: Sun Jan 25 15:18:00 PST 2004 until: Sat Oct 21 16:18:00 PDT 2006
Certificate fingerprints:
         MD5:  6A:FA:E9:B7:C0:F0:79:CB:EB:1A:CE:34:0D:CF:5D:99
         SHA1: A7:57:6D:B8:D1:98:88:28:AB:B5:36:EE:B6:43:81:C5:9C:3F:F4:4D
```

Making things work

- Web Access over SSL
  - Setup Apache Tomcat for SSL (w/o requiring client certificate)
  - Setup requiring client certificate
- Exchange of signed and encrypted mail
- Key-based authentication for SSH
- Web Services with WS-Security

Certificate Signing Request with Keytool

Generate keypair

```
C:\WINNT\system32\cmd.exe

C:\pankaj\temp\jstk-1.0.1>keytool -genkey -keystore test.ks -keyalg rsa -dname "
CN=www.dreamsite.com" -alias dreamsite
Enter keystore password:  changeit
Enter key password for <dreamsite>
        (RETURN if same as keystore password):

C:\pankaj\temp\jstk-1.0.1>keytool -list -keystore test.ks
Enter keystore password:  changeit

Keystore type: jks
Keystore provider: SUN

Your keystore contains 1 entry

dreamsite, Jan 13, 2004, keyEntry,
Certificate fingerprint (MD5): F8:43:B6:3F:63:BD:53:E2:36:4D:66:D7:C5:C4:CB:71

C:\pankaj\temp\jstk-1.0.1>keytool -certreq -alias dreamsite -file dreamsite.csr
-keystore test.ks -storepass changeit -keypass changeit

C:\pankaj\temp\jstk-1.0.1>type dreamsite.csr
-----BEGIN NEW CERTIFICATE REQUEST-----
MIIBWzCBxQIBADAcMRowGAYDVQQDExF3d3cuZHJlYW1zaXRlLmNvbTCBnzANBgkqhkiG9w0BAQEF
AAOBjQAwgYkCgYEAxYqnP5XwB+mqqG1mTURN+CmrUkgfvHS44gonhm+1fJPjUvvntZDNHxRh4wXk
ereHmy/OfDmTcKb+vWzVHo1KfuZlfqUM0JgoAUQ726j8SLW/G2iR+LuZdy2Uq3zEDSnGb1n9d4yv
av6hikCR3C/TxKytKSp6nUsKhgl6x2MJlvcCAwEAAaAAMA0GCSqGSIb3DQEBBAUAA4GBAGs88jmy
a/jQDyU/nGufSHG18ERgfhLtkN2leyE4eddLBVge+3M4zpDsaP$fSAjnO1ZZFXRMZ0RL+j7USDdn
Bez4acCAiIqZHc+WRXpbA6udImZ7eLHakeyU+mjJv1Ei7AtSGVrvShCymEECxNv8oBCNz6ti6dbN
92auyjyHXqA9
-----END NEW CERTIFICATE REQUEST-----

C:\pankaj\temp\jstk-1.0.1>_
```

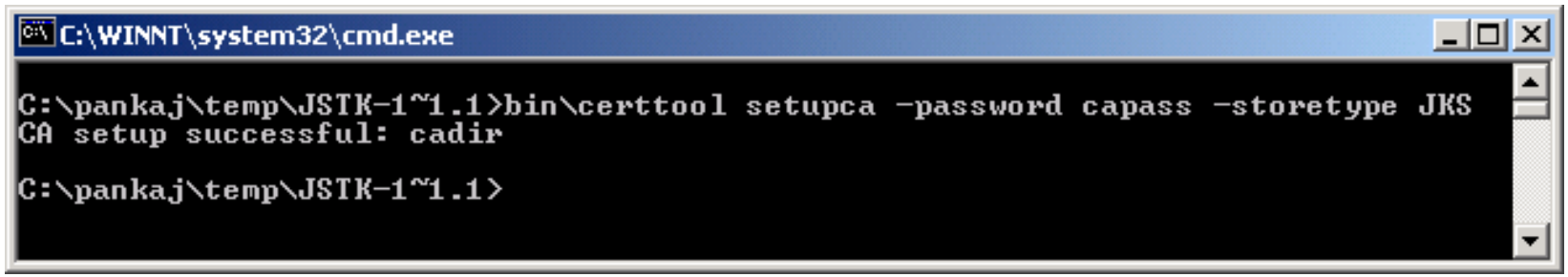
Create certificate Signing request

Getting a CA signed Certificate

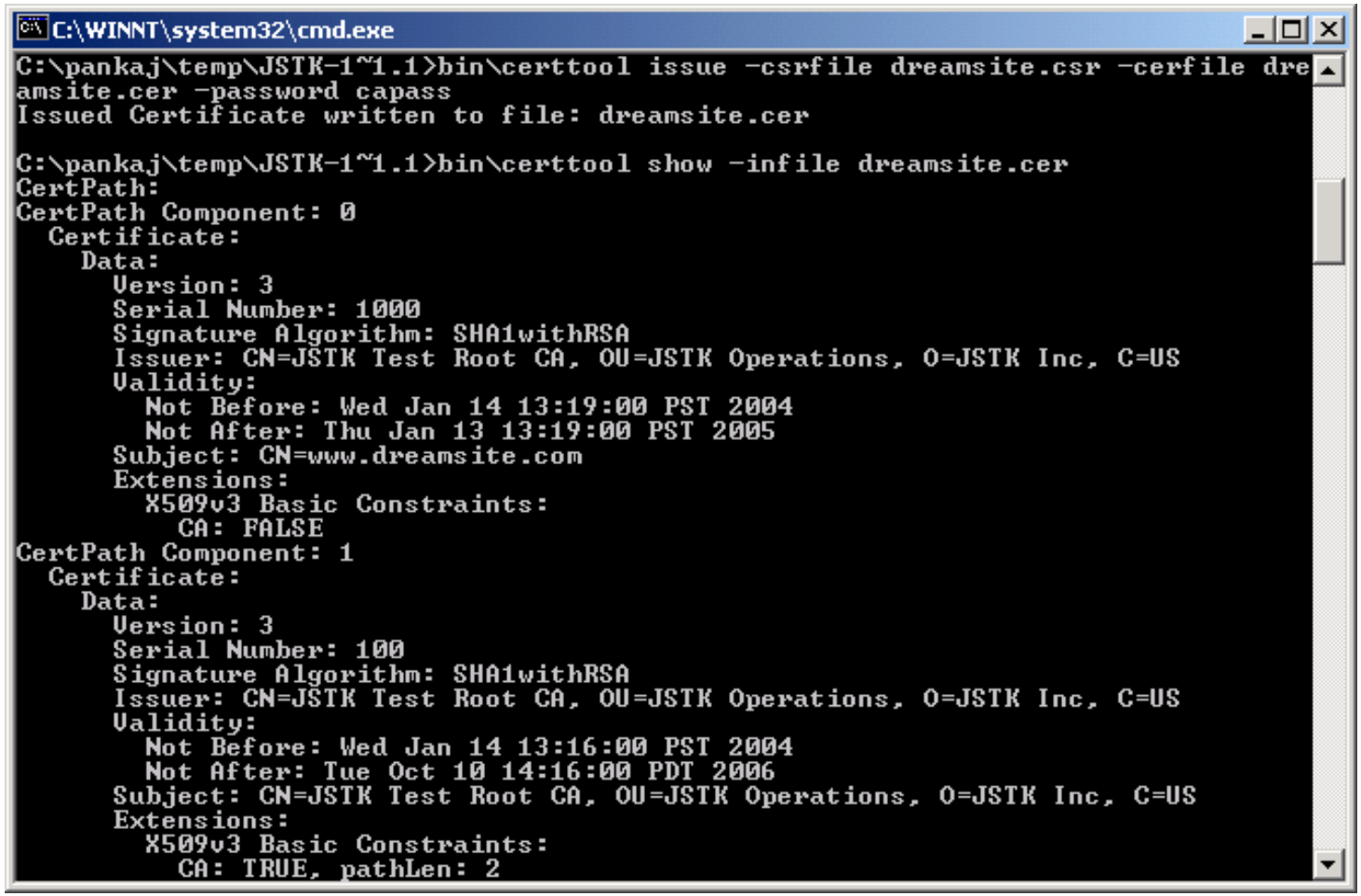
- Contact a well-known CA (VeriSign, Thawte, ...)
- Setup Your Own CA
  - Use JSTK (Available under an OSL 2.0)
  - Use OpenSSL
  - Others ...

Setting up your own CA

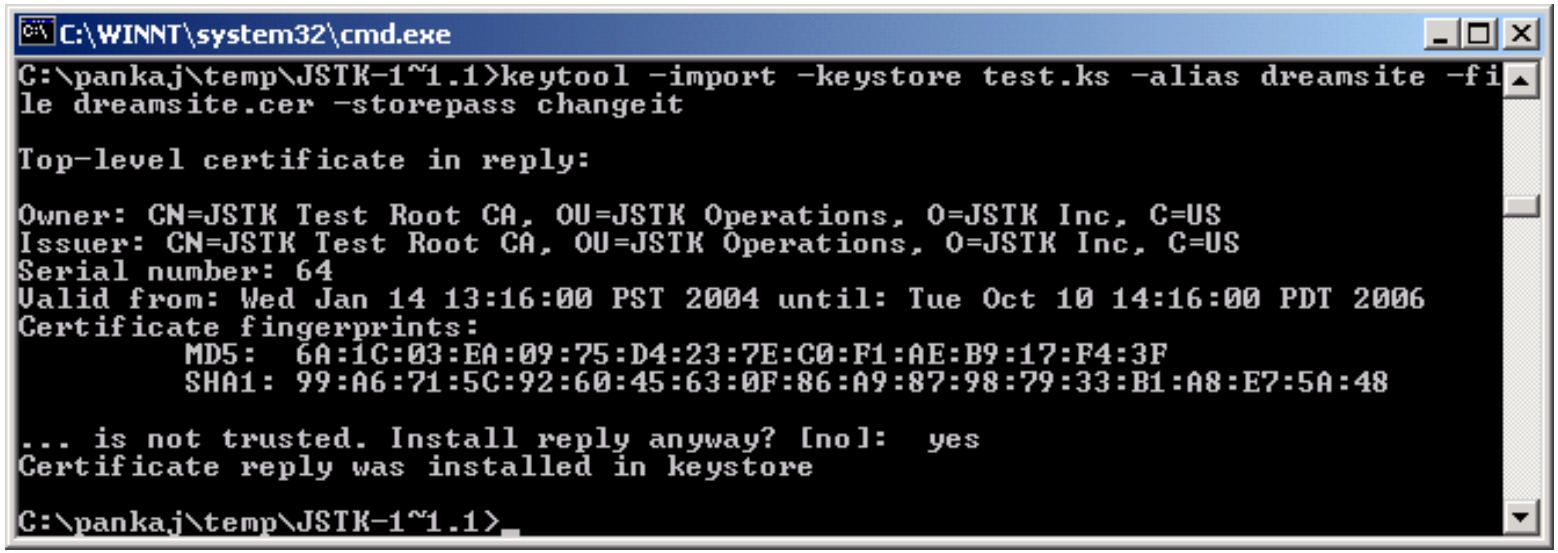
- Get JSTK s/w (jstk-1.0.1.zip) from http://www.j2ee-security.net.
- Unzip the distribution in your working directory.
- Change Directory to jstk-1.0.1
- Execute: bin\certtool setupca –password changeit



Issue Signed Certificate



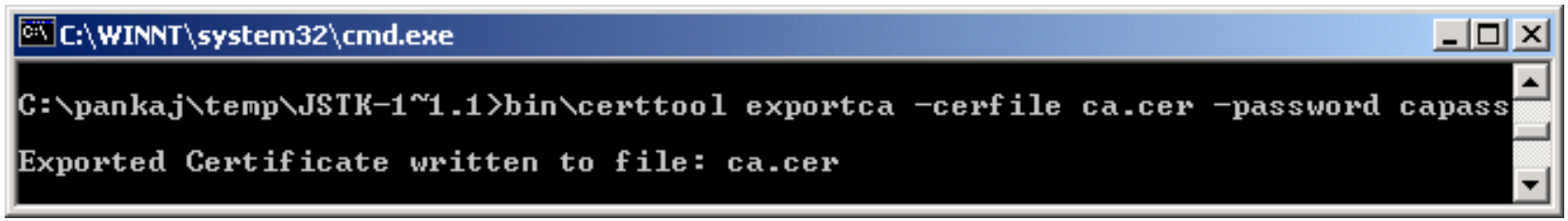
Import the Signed Certificate into Java keystore



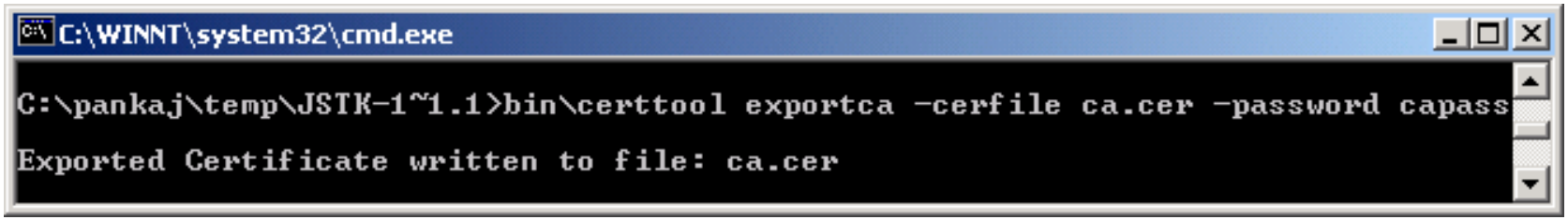
Import CA Cert. Into Browser

Not required for established CAs. (Browsers have their certs pre-imported).

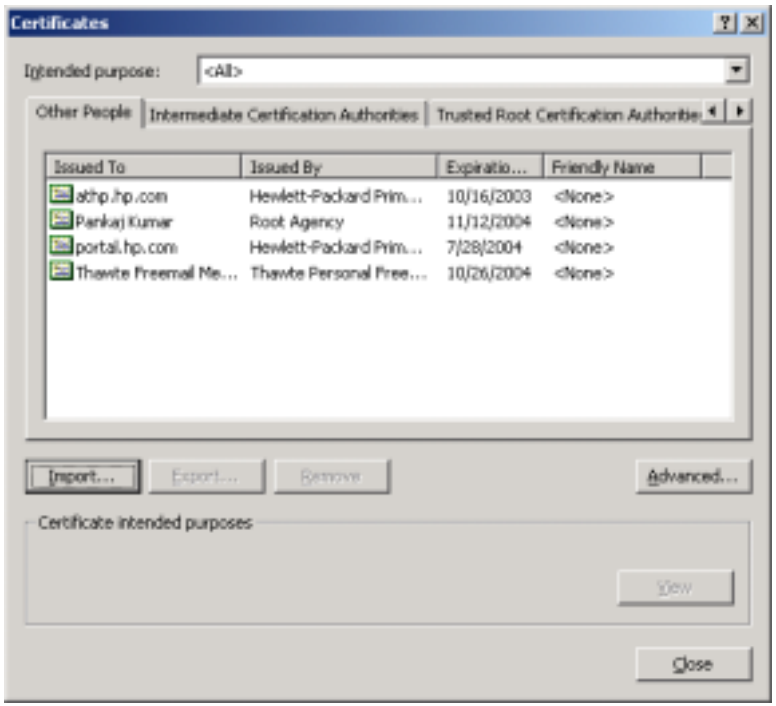
For your own CA, extract the CA cert. using JSTK utility certtool.



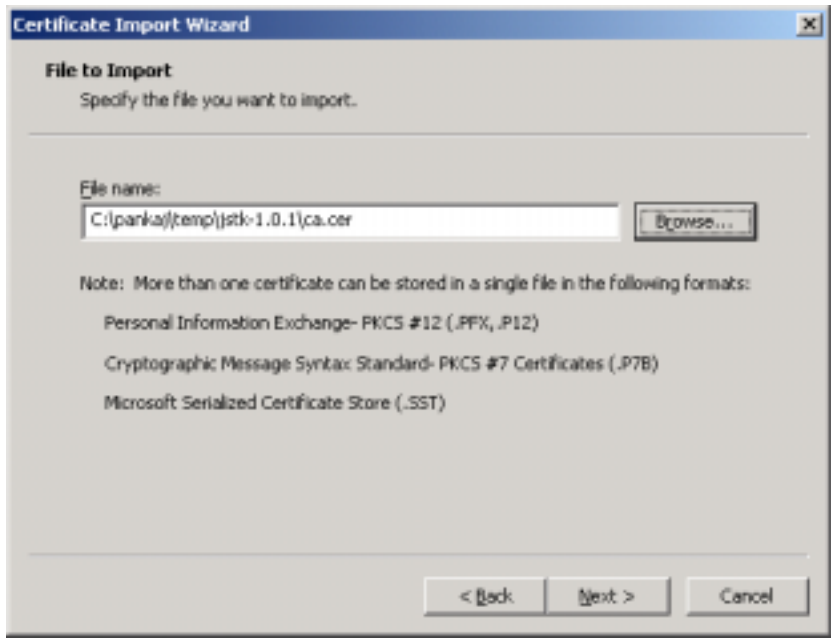
Import CA Cert. Into Browser (IE) Keystore



Internet Options → Content → Certificates

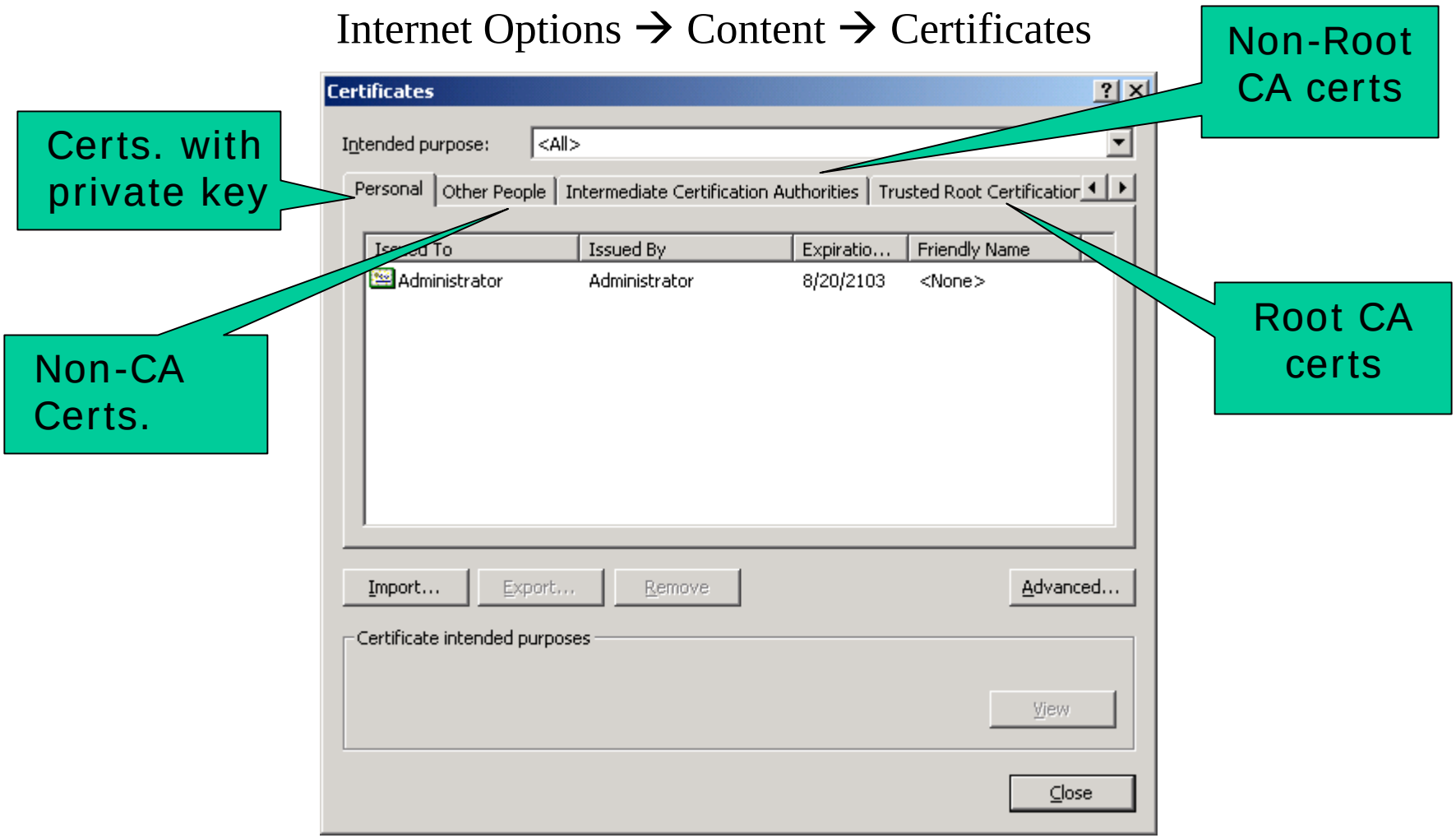


Import → Next → Browse



Import CA Cert. Into IE

Internet Options → Content → Certificates

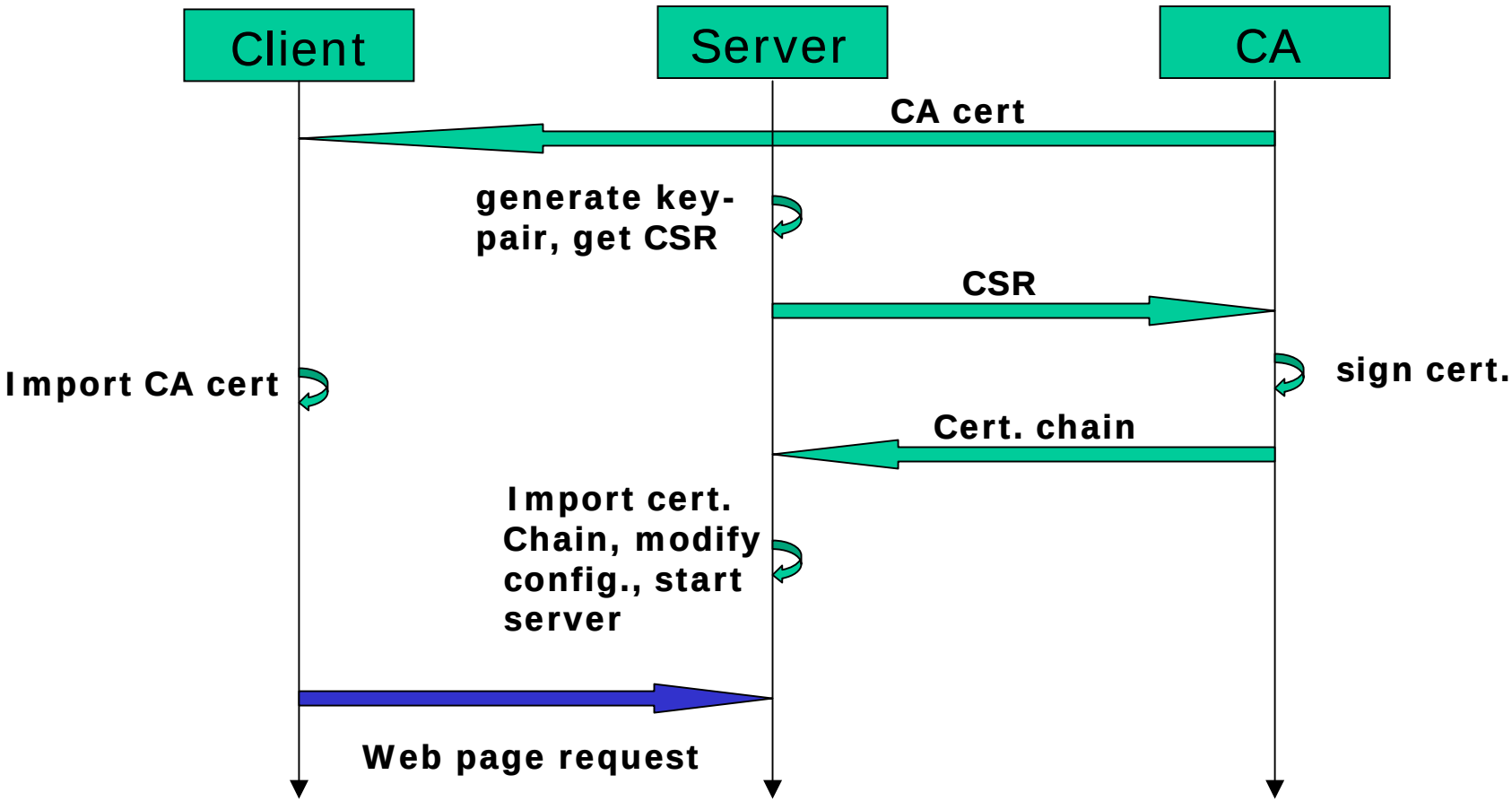


Configure Apache Tomcat

Modify `conf\server.xml`

```
 ...
<Connector className="org.apache.coyote.tomcat4.CoyoteConnector"
       port="8443" minProcessors="5" maxProcessors="75"
       enableLookups="true"
       acceptCount="100" debug="0" scheme="https" secure="true"
       useURIValidationHack="false" disableUploadTimeout="true">
    <Factory
          className="org.apache.coyote.tomcat4.CoyoteServerSocketFactory"
          clientAuth="false" protocol="TLS"
          keystoreFile="c:\\pankaj\\temp\\jstk-1.0.1\\test.ks"
          keystorePass="changeit" />
    </Connector>
...
```

Recap.


Client
Server
CA
CA cert
generate key-pair, get CSR
CSR
sign cert.
Import CA cert
Cert. chain
Import cert. Chain, modify config., start server
Web page request

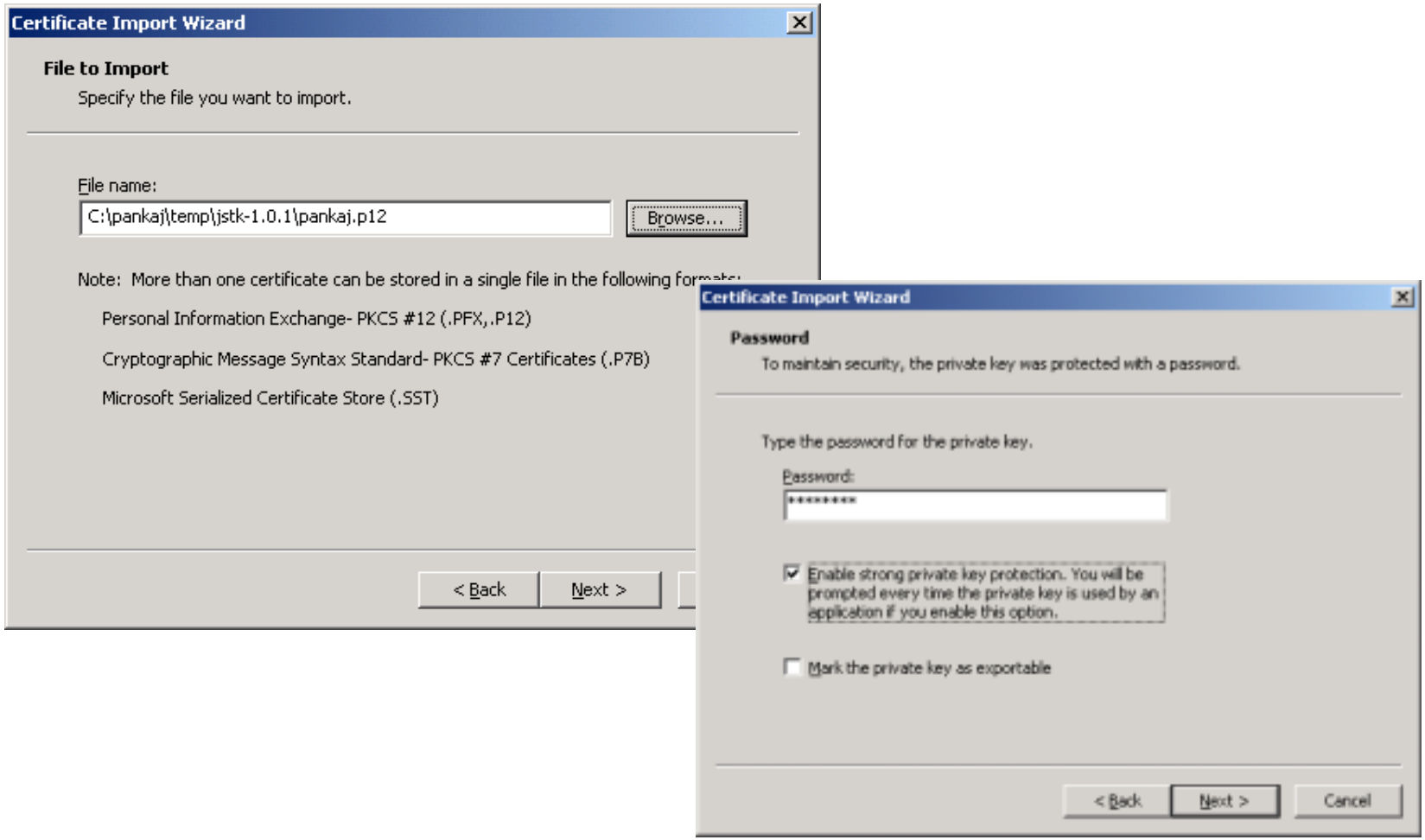
Getting a CA signed Personal Cert.

- Same steps as getting server certificate. (so we can skip the details!)
  - Generate self-signed cert (keytool –genkey), provide identity information.
  - Get Certificate Signing Request
  - Get CA signed cert. in a cert-chain
  - Import the cert-chain in Java keystore.
- But how to get the private key and the certificate in the Browser? – Cannot import from Java keystore.

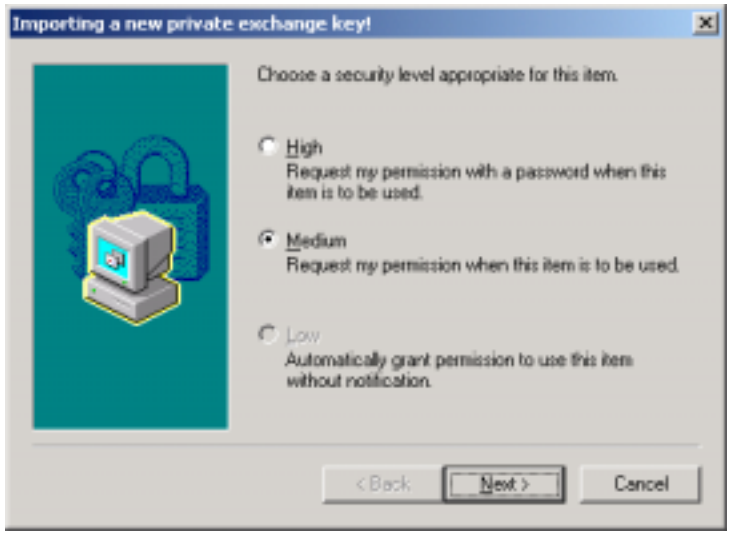
Moving private key and cert from Java keystore to Browser

- Export key-entry from java keystore to a PKCS12 file.
  - Cannot do with keytool. Use JSTK utility crypttool.

Import PKCS12 file to Browser



Import PKCS12 file … (Contd.)



Apache Tomcat Configuration for CERT-based Client Authentication

Modify `conf\server.xml`

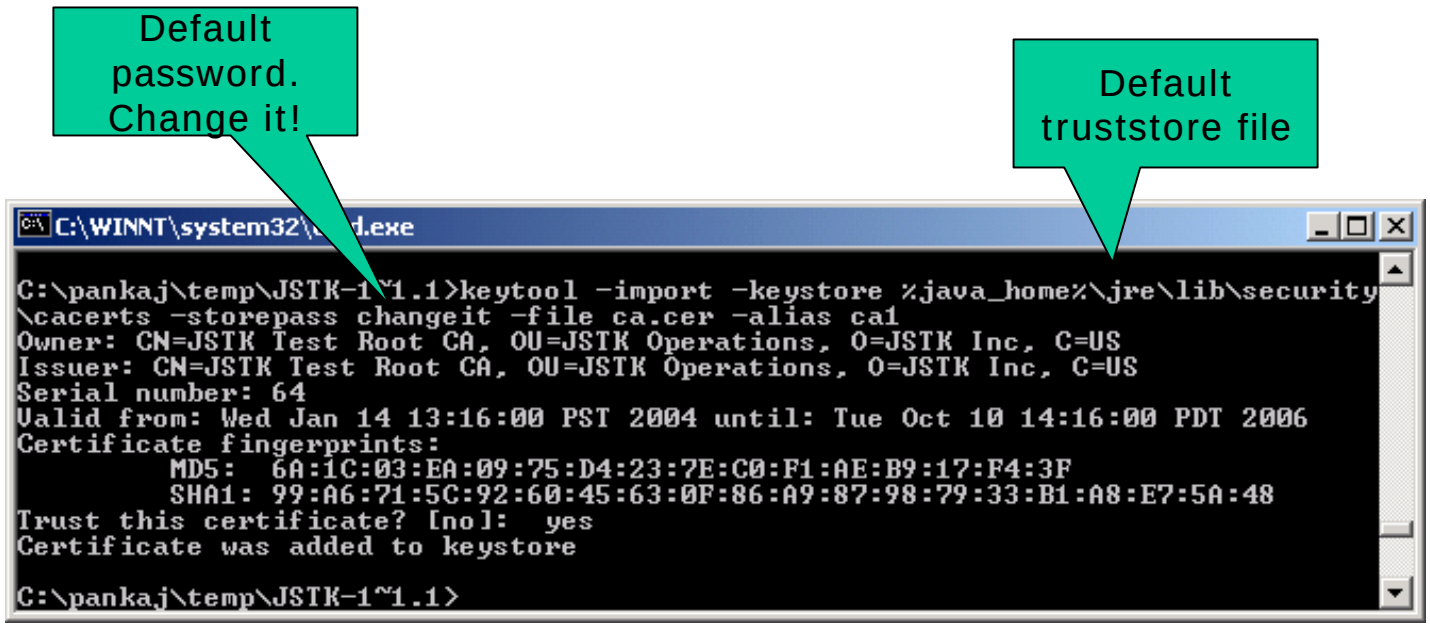
```
...
<Connector className="org.apache.coyote.tomcat4.CoyoteConnector"
       port="8443" minProcessors="5" maxProcessors="75"
       enableLookups="true"
       acceptCount="100" debug="0" scheme="https" secure="true"
       useURIValidationHack="false" disableUploadTimeout="true">
    <Factory
          className="org.apache.coyote.tomcat4.CoyoteServerSocketFactory"
          clientAuth="true" protocol="TLS"
          keystoreFile="c:\\pankaj\\temp\\jstk-1.0.1\\test.ks"
          keystorePass="changeit" />
    </Connector>
...
```

Change "false" to "true"

Trusted CA Certs for Apache Tomcat

- By default Tomcat relies on the CA certs supplied with the JRE.
- These certs are in truststore file %JAVA_HOME%\jre\lib\security\cacerts
- For our own CA, we would either
  - import the CA cert in the default truststore; or
  - specify the truststore with the CA cert as the truststore for Tomcat through JVM system properties.

Importing CA cert into the default truststore



Creating Certificate for Secure email communication

User

```
C:\JSTK-1~1.1>keytool -genkey -alias veenae -keystore
test.ks -storepass changeit -keyalg rsa -dname
"CN=Veena Prakash, EMAIL=veena@j2ee-security.net,
OU=Home, O=Family, L=Sunnyvale, S=CA, C=US"
Enter key password for <veenae>
        (RETURN if same as keystore password):
```

```
C:\JSTK-1~1.1>keytool -certreq -alias veenae
-keystore test.ks -storepass changeit -file veenae.csr
```

CA

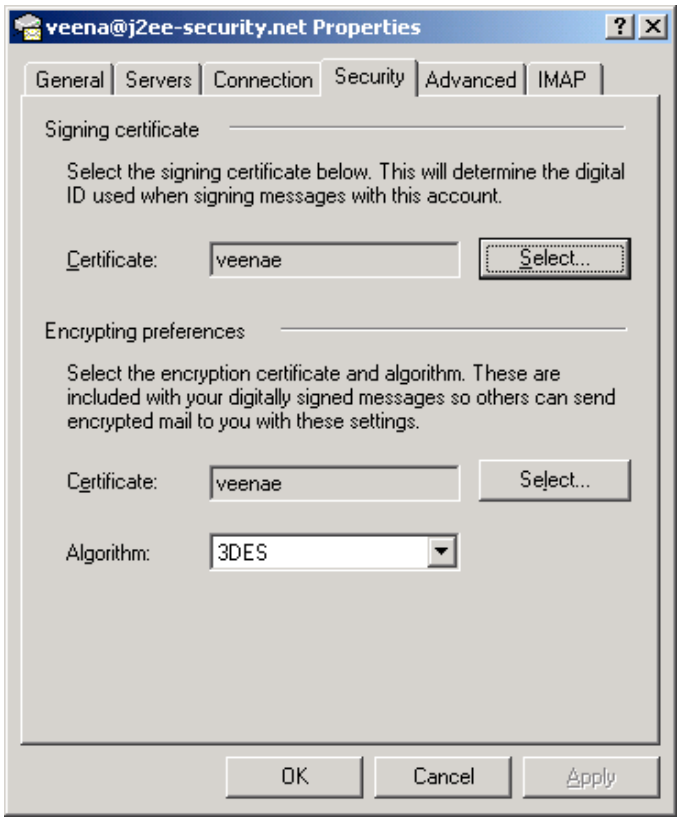
```
C:\JSTK-1~1.1>bin\certtool issue -password capass
-csrfile veenae.csr -cerfile veenae.cer
Issued Certificate written to file: veenae.cer
```

```
C:\JSTK-1~1.1>keytool -import -alias veenae -keystore test.ks
-storepass changeit -file veenae.cer
```

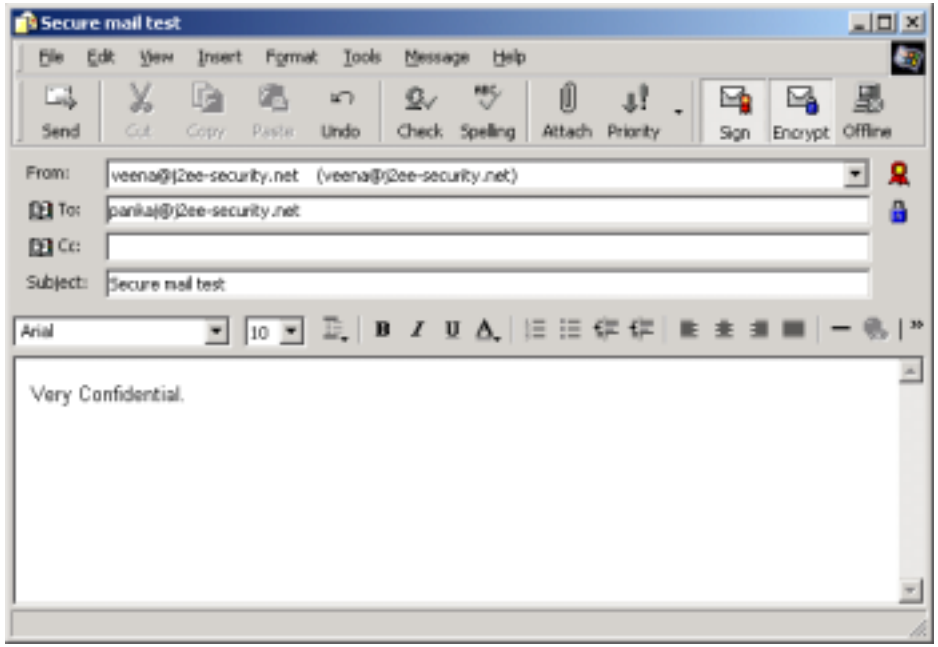
```
C:\SDWEST~1\src>java ToPKCS12 -keystore c:\jstk-1.0.1\test.ks
-storepass changeit -alias veenae -outfile veenae.p12
```

Sending a signed and encrypted Message with Outlook Express

Setup Certificate
(Tools → Accounts → Mail →Properties →Security)



compose Mail and secure it



Key-based authentication for SSH

Generate Key-pair:

C:\jstk>**ssh-keygen -t rsa**

```
Generating public/private rsa key pair.
Enter file in which to save the key (/home/Administrator/.ssh/id_rsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/Administrator/.ssh/id_rsa.
Your public key has been saved in /home/Administrator/.ssh/id_rsa.pub.
The key fingerprint is:
f0:14:86:eb:57:a6:07:ba:24:aa:13:2e:8c:26:12:7b Administrator@vishnu
```

Copying public key to the remote Server and establishing session:

C:\jstk>**ssh -l pankaj j2ee-security.net "mkdir .ssh; chmod 0700 .ssh"**

```
pankaj@j2ee-security.net's password:******
```

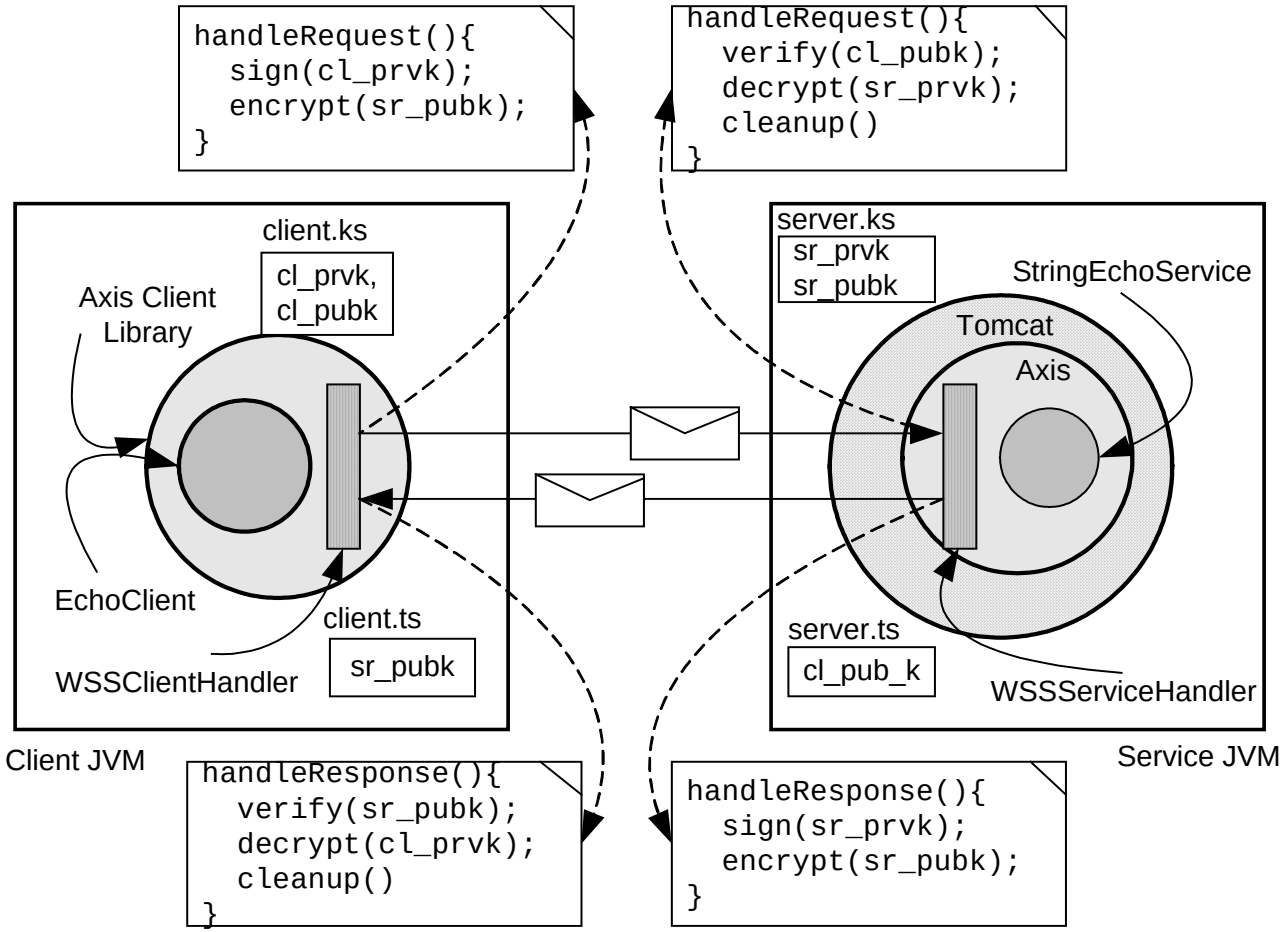
C:\jstk>**scp ~Administrator/.ssh/id_rsa.pub **
pankaj@ j2ee-security.net:.ssh/authorized_keys2

```
pankaj@j2ee-security.net's password:******
id_rsa.pub                                    100%  230    224.6KB/s   00:00
```

C:\jstk>**ssh -l pankaj j2ee-security.net**

```
[pankaj@j2ee-security pankaj]$
```

Exchange of WS-Security protected SOAP messages



Want to know more?

- J2EE Security for Servlets, EJBs and Web Services (http://www.j2ee-security.net)
- PKCS series (http://www.rsasecurity.com/rsalabs/pkcs/)
- Practical Cryptography (http://www.schneier.com/book-practical.html)
- IETF RFCs: 1421, 1422, 1423, 1424 and many more