SSL

SSL

- A protocol on top of TCP/IP
- The SSL API is an extension to the TCP/IP API.
- (SSL clients and servers are sometimes known as peers.)

SSL Capabilities

- SSL Server Authentication
  - Requires Digital Certificate
- SSL Client Authentication
  - Optional, usually not used
- Encryption of data.
  - Optional, usually used

SSL Handshake

- Exchange of messages in order to
  - Authenticate the server
  - Optionally authenticate the client
  - Allow client and server to select the cipher used.
  - Generate shared secret (symmetric key)
  - Optionally establish an encrypted SSL connection.

Generating Shared Secret

- The Diffie Hellman algorithm is one option (considered the safest) for key exchange.
- A symmetric key can be generated by the DH algorithm and this used subsequently to encrypt the data.

Keytool

Keytool

- A utility program for managing keys.
- keytool.exe in JDK bin folder.

KeyStores and TrustStores

- A Keystore is used to store private keys and associated certificates (which contains the public key).
- A TrustStore is a keystore used to store certificates of trusted Certificate Authorities.

KeyStores and TrustStores

- A Java program looks for TrustStores as follows.
  - In the location given by the System property javax.net.ssl.trustStore
  - In lib/security/jssecerts off Java Home
  - In lib/security/cacerts off Java Home
- The cacerts file (binary file) has a number of entries for Verisign/Thawte.

[[[Keytool arguments]

- -genkey
  - generate a self-signed private key/certificate pair (keystore pair)
- -keystore
  - create a new keystore with this name
- -keyalg
  - the encryption algorithm to be used
- -alias
  - unique name associated with keystore entry

Keytool arguments

- -storepass
  - password needed to access the keystore
- -keypass
  - password needed to obtain the private key associated with the alias
- -certreq
  - generate a certificate signing request
- -file
  - outputfile for certreq

Creating a (self-signed) keystore entry

- keytool -genkey -keystore CERTS -keyalg rsa -alias paul -storepass serverkspw -keypass serverpw

- You will be prompted for name, organization unit, organization, city, state or province, two-letter code.

Creating a (self-signed) keystore entry

- Note that this Certificate is digitally signed by yourself, not a trusted Certificate Authority.
- Signer and subject are the same.
- You might want to set up a Certificate authority within an organisation.

Generate a Certificate Signing Request

- keytool -certreq -keystore CERTS -alias paul -storepass serverkspw -keypass serverpw -file CERTREQ

- This can be sent to an appropriate CA and a digital certificate sent back say DIGITALCERT.

Import the Digital Certificate back into the Keystore]]]

- keytool -import -keystore CERTS -alias paul -file DIGITALCERT -truststore

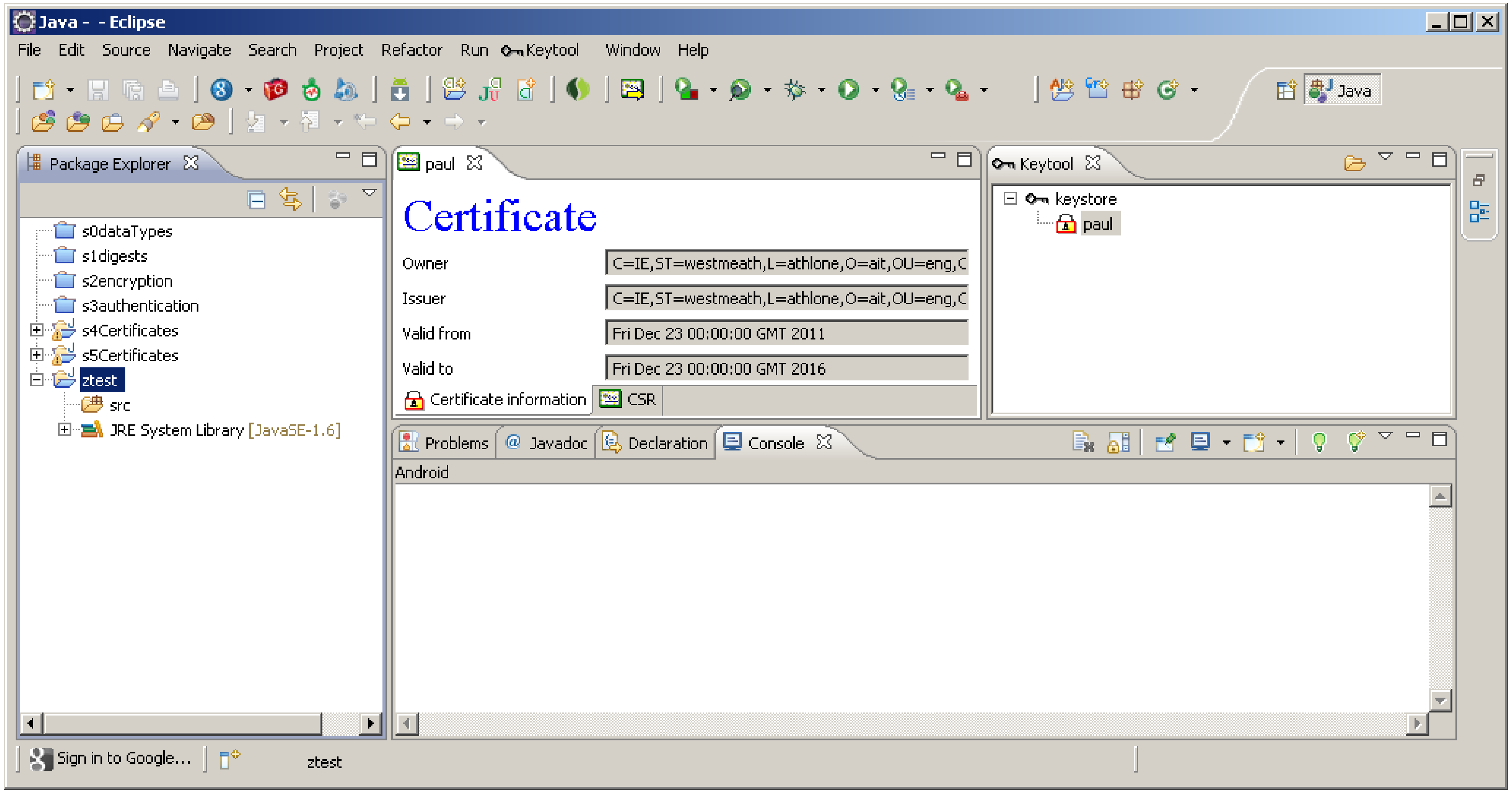
- Now the Digital Certificate is signed by Verisign.

Keytool – Eclipse Plugin

Update site

- **http://keytool.sourceforge.net/update**
- Help - Software Updates - Find and install - Search for new feature....
- Click New Remote Site and add http://keytool.sourceforge.net/update in name and URL, and make sure its checked.
- Hit Finish.
- (1.4.2)

Keytool – Eclipse Plugin



Certificate

Owner	C=IE,ST=westmeath,L=athlone,O=ait,OU=eng,C
Issuer	C=IE,ST=westmeath,L=athlone,O=ait,OU=eng,C
Valid from	Fri Dec 23 00:00:00 GMT 2011
Valid to	Fri Dec 23 00:00:00 GMT 2016

```
Android
```

keytool

- Create a new certificate.
- Choose – create a new keystore
- Specify an alias (for the key)
- Specify a key password (1234)
- Specify information.
- Specify a filename
- And file password (file)

Keytool – Open a keystore

- The keystore is created in the eclipse folder.
- Copy it into your project.

- Open the keystore.
- Can see the certificate in it.
- Note that the issuer and the subject are the same.
- This is a "self signed certificate".

C1show.java

```
String keystoreFilename = "keystore";
char[] password = "file".toCharArray();
String alias = "paul";

FileInputStream fIn = new
            FileInputStream(keystoreFilename);
KeyStore keystore = KeyStore.getInstance("JKS");

keystore.load(fIn, password);
Certificate cert = keystore.getCertificate(alias);
System.out.println(cert);
```

Export Certificate from keystore

- RC- export certificate
- Specify the name of the certificate file (say paul.cer)

C3PrintCert.java

```
FileInputStream fr = new FileInputStream("paul.cer");
CertificateFactory cf = CertificateFactory.getInstance("X509");
X509Certificate c = (X509Certificate) cf.generateCertificate(fr);
System.out.println("Read in the following certificate:");
System.out.println("\tCertificate for: " + c.getSubjectDN());
System.out.println("\tCertificate issued by: " + c.getIssuerDN());
System.out.println("\tThe certificate is valid from "
+ c.getNotBefore() + " to " + c.getNotAfter());
System.out.println("\tCertificate SN# " + c.getSerialNumber());
System.out.println("\tGenerated with " + c.getSigAlgName());
```

TrustStore- cacerts

- Certificate Authority Certificates
- Contains the (self-signed) certificates for the <u>trusted</u> Certificate Authorities.
- In C:\Program Files\Java\jdk1.6.0_20\jre\lib\security

TrustStore- cacerts

- Clients will use this to check Digital Certificates.
- The Digital Certificates will be signed by a C.A.
- Client looks up the TrustStore.
- Gets the public key of the C.A.
- Checks the digital signature on the Digital Certificate.

SSL Example

- S1Server.java will accept SSL connections from clients.
- It authenticates itself by means of a certificate stored in ″keystore″ (the original keystore).
- It needs to know the password for the keystore as well as the password for the key (paul).
- Now this certificate is self-signed by paul.

SSL Example

- The only way it will be accepted by the client is if the client thinks that paul is a valid C.A.
- We are going to fool the client by putting pauls certificate in the cacerts file (truststore).
-

SSL Example - E1Server.java

```
char[] KSPASSWORD = "kkkk".toCharArray();
char[] PASSWORD = "1234".toCharArray();

KeyStore keyStore = KeyStore.getInstance("JKS");
keyStore.load(new FileInputStream("keystore"), KSPASSWORD);

KeyManagerFactory keyManagerFactory = KeyManagerFactory
        .getInstance("SunX509");
keyManagerFactory.init(keyStore, PASSWORD);

SSLContext sslContext = SSLContext.getInstance("TLS");
sslContext.init(keyManagerFactory.getKeyManagers(), null, null);

ServerSocketFactory ssFactory = sslContext.getServerSocketFactory();
```

SSL Example - E1Server.java

```
int port = 443;
ServerSocket ss = ssFactory.createServerSocket(port);
Socket socket = ss.accept();

// Create streams to securely send and receive data to the client
InputStream in = socket.getInputStream();
BufferedReader r = new BufferedReader(new InputStreamReader(in));
String name = r.readLine();

OutputStream o = socket.getOutputStream();
PrintWriter p = new PrintWriter(o);
p.println("Hello " + name);
p.close();
```

SSL Example – E2SSLClient.java

```
System.setProperty("javax.net.ssl.trustStore", "cacerts");
System.setProperty("javax.net.ssl.trustStorePassword", "changeit");

SSLSocketFactory factory = (SSLSocketFactory)SSLSocketFactory.getDefault();
SSLSocket socket = (SSLSocket)factory.createSocket("localhost", 443);
socket.startHandshake();

OutputStream o = socket.getOutputStream();
PrintWriter p = new PrintWriter(o) ;
p.println("paul") ; p.flush() ;

InputStream in = socket.getInputStream();
BufferedReader r = new BufferedReader(new InputStreamReader(in)) ;
String reply = r.readLine();
System.out.println("From Server " + reply);
```

Summary: SSL Server

- Send
  - Digital Certificate
  - Random piece of text
  - Digitally signed (encrypted one-way hash of text)
  - Agree cipher
  - Exchange DH public keys
  - Genereate symmetric key
  - Encrypt data

Summary: SSL Client

- Verifies the Digital Certificate, i.e. checks the Digital Signature on the Certificate (using the CA’s public key).
- Reads the Public Key of the sender from the Certificate.
- Uses this to check the Digital Signature of the sender.
- Agree cipher
- Exchange DH public keys
- Genereate symmetric key
- Encrypt data