Cryptography and Security Lab
Keystores and Truststores

• A keystore is a database of keys
• Keystores contain:
  – Private keys
  – Certificates with corresponding public keys
• You need a keystore if you are the server
Keystores and Truststores

• A truststore is also a database
• It is a keystore that is used when making decision on what to trust.
  – It is where you keep info about trusting other sites
• Truststores contain:
  – Certificate Authorities that you trust to identify other parties.
  – Certificates from others you trust and expect to communicate with
• A client who wants to connect to trusted servers needs this.
Transfer can happen many ways, for example:
- By default (installed in your browser)
- By hand (install your company's certificates)
- As part of a SSL handshake.
Doing RSA with keytool

- keytool is an application that makes it easy to:
  - create an RSA key pair in a keystore
  - generate a self-signed digital certificate
  - import the certificate into a truststore

- keytool implemented the keystore as a file
  - In a proprietary Oracle format named "jks".
  - Application developers have the choice of using 3rd party keystore formats
  - This is important to know for you will need it when accessing it by code.
Generating a RSA key pair

```bash
>>> keytool -genkey -alias joe -keyalg RSA -keystore joekeystore
Enter keystore password: ********
Re-enter new password: ********
What is your first and last name?
[Unknown]: Joseph Mertz
What is the name of your organizational unit?
[Unknown]: Heinz
What is the name of your organization?
[Unknown]: Carnegie Mellon
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=Joseph Mertz, OU=Heinz, O=Carnegie Mellon, L=Pittsburgh, ST=PA, C=US correct?
[no]: yes
Enter key password for <joe>
  (RETURN if same as keystore password):
```
Create a self-signed RSA certificate

```bash
>>> keytool -export -alias joe -keystore joekeystore -file joe.cer
Enter keystore password:
Certificate stored in file <joe.cer>
```
Import the certificate into a truststore

This would be done on a client:

```plaintext
<<< keytool -import -alias joeclient -keystore joeclient.truststore -file joe.cer
Enter keystore password: •••••• This is not the same key as on the server
Re-enter new password: ••••••
Owner: CN=Joseph Mertz, OU=Heinz, O=Carnegie Mellon, L=Pittsburgh, ST=PA, C=US
Issuer: CN=Joseph Mertz, OU=Heinz, O=Carnegie Mellon, L=Pittsburgh, ST=PA, C=US
Serial number: 50bcced6
Valid from: Mon Dec 03 11:09:58 EST 2012 until: Sun Mar 03 11:09:58 EST 2013
Certificate fingerprints:
Signature algorithm name: SHA1withRSA
Version: 3
Trust this certificate? [no]: yes
Certificate was added to keystore
```
// Setup to read the certificate file
FileInputStream cfis = new FileInputStream("joe.cer");

// Read in the certificate
CertificateFactory cF = CertificateFactory.getInstance("X.509");
Certificate cert = cF.generateCertificate(cfis);
cfis.close();

// Get the Public Key from the certificate to print it.
RSAPublicKey Kpub = (RSAPublicKey) cert.getPublicKey();
System.out.println("The public key exponent is: "
    +Kpub.getPublicExponent()
    +" and the modulus is " + Kpub.getModulus());

See ReadCertificate.java
How to generate an AES key

// Get the KeyGenerator
KeyGenerator kgen = KeyGenerator.getInstance("AES");
kgen.init(128);

// Generate the secret key
SecretKey skey = kgen.generateKey();

See AES.java
How to encrypt and decrypt with a key

// Instantiate the cipher for Encrypting with AES
Cipher cipher = Cipher.getInstance("AES");
cipher.init(Cipher.ENCRYPT_MODE, skey);

// Encrypt the first command line argument
byte[] encrypted = cipher.doFinal(args[0].getBytes());
System.out.println("encrypted string: " + asHex(encrypted));

// Initiate the cipher for Decrypting
cipher.init(Cipher.DECRYPT_MODE, skey);

// Decrypt the encrypted string
byte[] original = cipher.doFinal(encrypted);
String originalString = new String(original);
System.out.println("Decrypted string: " + originalString);

See AES.java
1. Create a keystore and a Kpub Kpriv pair
2. Export a certificate
3. Import the certificate into a truststore
4. Read the certificate in Java
   – See ReadCertificate.java
5. Generate an AES key, and encrypt and decrypt text
   – See AES.java
6. Optional Advanced Challenges:
   a) Read Kpriv from your keystore
      • Hint: see the Java API for KeyStore for sample code.
   b) Encrypt and decrypt a message with your Kpub and Kpriv
      • Hint: combine 4 & 6a for getting the two keys, and 5 for encrypting and decrypting. Only use Cipher.getInstance("RSA") instead of "AES".