Architecture of Enterprise Applications 5
Security II

Haopeng Chen

RE*liable,* ***IN****telligent and* ***S****calable Systems Group (***REINS***)*
Shanghai Jiao Tong University
Shanghai, China
http://reins.se.sjtu.edu.cn/~chenhp
e-mail: chen-hp@sjtu.edu.cn

Contents

- SECURITY
  - USER AUTHENTICATION
  - DIGITAL SIGNATURES
  - CODE SIGNING
  - ENCRYPTION

User Authentication

REliable, INtelligent & Scalable Systems

- The Java Authentication and Authorization Service (JAAS) is a part of Java SE 1.4 and beyond.
  - The "authentication" part is concerned with ascertaining the identity of a program user.
  - The "authorization" part maps users to permissions.

```
grant principal com.sun.security.auth.UnixPrincipal "harry" {
   permission java.util.PropertyPermission "user.*", "read"; . . .
};
```

User Authentication

REliable, INtelligent & Scalable Systems

```
try
{
  System.setSecurityManager(new SecurityManager());
  LoginContext context = new LoginContext("Login1");
  // defined in JAAS configuration file
  context.login();
  // get the authenticated Subject
  Subject subject = context.getSubject();
  . . .
  context.logout();
}
catch (LoginException exception)
// thrown if login was not successful
{
  exception.printStackTrace();
}
```

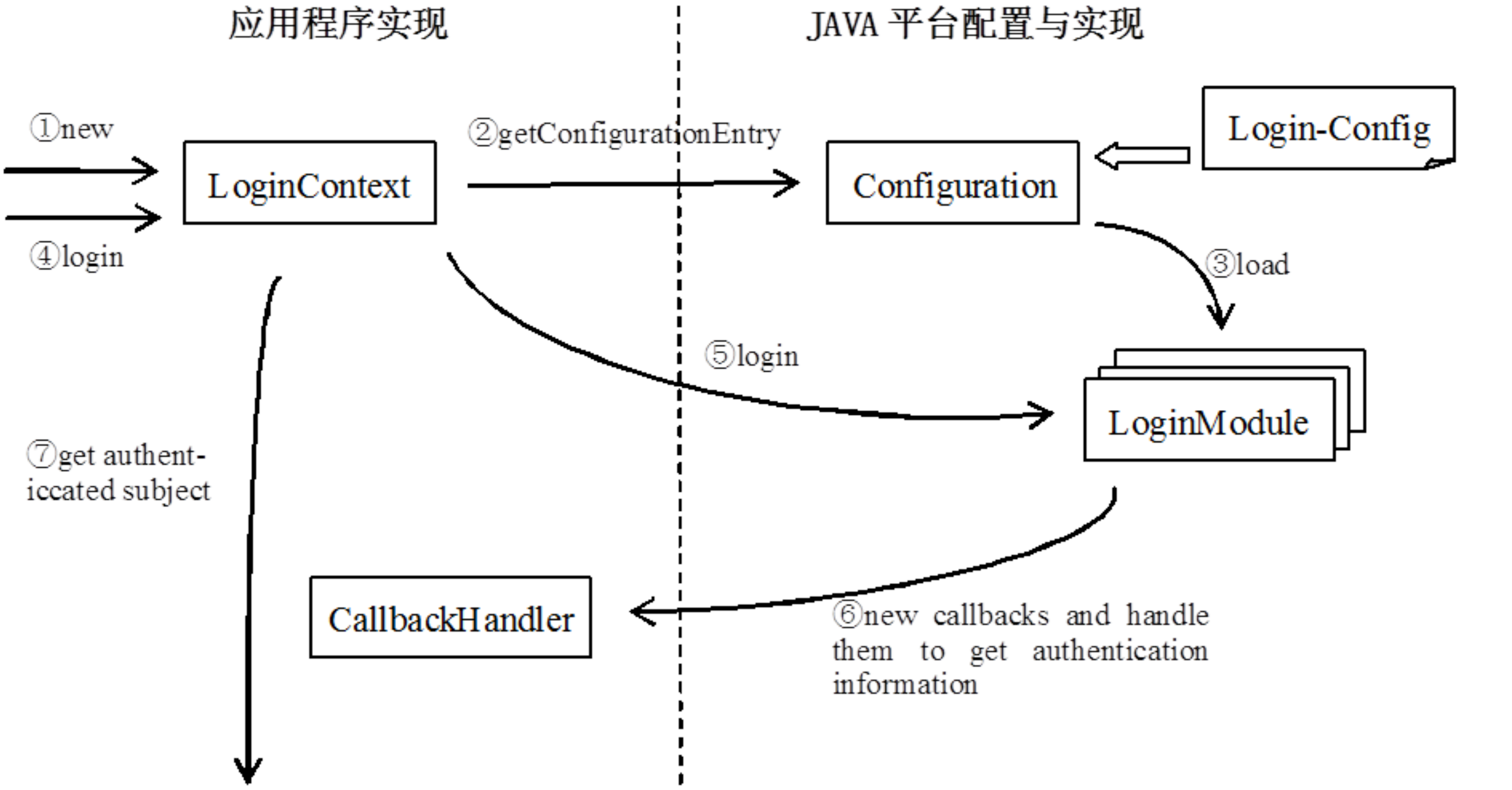
User Authentication

REliable, INtelligent & Scalable Systems

- Jaas.config

```
Login1
{
  com.sun.security.auth.module.UnixLoginModule required;
  com.whizzbang.auth.module.RetinaScanModule sufficient;
};
Login2
{
  . . .
};
```

User Authentication



User Authentication

- The following modules are supplied in the com.sun.security.auth.module package:
  - UnixLoginModule
  - NTLoginModule
  - Krb5LoginModule
  - JndiLoginModule
  - KeyStoreLoginModule

- A login policy consists of a sequence of login modules, each of which is labeled required, sufficient, requisite, or optional:
  - The modules are executed in turn, until a sufficient module succeeds, a requisite module fails, or the end of the module list is reached.
  - Authentication is successful if all required and requisite modules succeed, or if none of them were executed, if at least one sufficient or optional module succeeds.

User Authentication

REliable, INtelligent & Scalable Systems

```
grant principalClass "principalName "

PrivilegedAction action = new
  PrivilegedAction()
  {
    public Object run()
    {
      // run with permissions of subject principals
       . . .
    }
  };
Subject.doAsPrivileged(subject, action, null);
// or doAs(subject, action)
```

User Authentication

- An Example
  - AuthTest.java

```
import java.security.*;
import javax.security.auth.*;
import javax.security.auth.login.*;

public class AuthTest
{
  public static void main(final String[] args)
  {
    try
    {
      System.setSecurityManager(new SecurityManager());
      LoginContext context = new LoginContext("Login1");
      context.login();
      System.out.println("Authentication successful.");
```

User Authentication

REliable, INtelligent & Scalable Systems

- An Example
  - AuthTest.java

```
    Subject subject = context.getSubject();
    System.out.println("subject=" + subject);
    PrivilegedAction action = new SysPropAction("user.home");
    Object result = Subject.doAsPrivileged(subject, action, null);
    System.out.println(result);
    context.logout();
  }
  catch (LoginException e)
  {
    e.printStackTrace();
  }
 }
}
```

REliable, INtelligent & Scalable Systems

- An Example
  - SysPropAction.java

```
import java.security.*;
public class SysPropAction implements PrivilegedAction
{
  /**
    Constructs an action for looking up a given property.
    @param propertyName the property name (such as "user.home")
  */
  public SysPropAction(String propertyName) { this.propertyName = propertyName; }

  public Object run()
  {
    return System.getProperty(propertyName);
  }

  private String propertyName;
}
```

User Authentication

REliable, INtelligent & Scalable Systems

- An Example
  - AuthTest.policy

```
grant codebase "file:login.jar"
{
  permission javax.security.auth.AuthPermission "createLoginContext.Login1";
  permission javax.security.auth.AuthPermission "doAsPrivileged";
};

grant principal com.sun.security.auth.UnixPrincipal "harry"
{
  permission java.util.PropertyPermission "user.*", "read";
};
```

  - Jaas.config

```
Login1
{
  com.sun.security.auth.module.UnixLoginModule required;
};
```

User Authentication

- An Example

```
javac AuthTest.java
jar cvf login.jar AuthTest*.class
javac SysPropAction.java
jar cvf action.jar SysPropAction.class

java -classpath login.jar:action.jar
  -Djava.security.policy=AuthTest.policy
  -Djava.security.auth.login.config=jaas.config
  AuthTest
```

JAAS Login Module

- we can add role-based permissions into a policy file:

```
grant principal SimplePrincipal "role=admin" { . . . }
```

- Our login module looks up users, passwords, and roles in a text file that contains lines like this:

```
harry|secret|admin
carl|guessme|HR
```

- The Login Module checks whether the user name and password match a user record in the password file. If so, we add two SimplePrincipal objects to the subject's principal set:

```
Set<Principal> principals = subject.getPrincipals();
principals.add(new SimplePrincipal("username", username));
principals.add(new SimplePrincipal("role", role));
```

JAAS Login Module

REliable, INtelligent & Scalable Systems

- An Example: SimpleLoginModule.
  - The initialize method receives
    - The Subject that is being authenticated.
    - A handler to retrieve login information.
    - A sharedState map that can be used for communication between login modules.
    - An options map that contains name/value pairs that are set in the login configuration.
  - For example, we configure our module as follows:
    - SimpleLoginModule required pwfile="password.txt";
  - The login module retrieves the pwfile settings from the options map.
  - The handler is specified when you construct the LoginContext
  - For example,
    - LoginContext context = new LoginContext("Login1", new com.sun.security.auth.callback.DialogCallbackHandler());

JAAS Login Module

REliable, INtelligent & Scalable Systems

- An Example: SimpleLoginModule.
  - The handle method of handlers

```
public void handle(Callback[] callbacks) {
   for (Callback callback : callbacks) {
      if (callback instanceof NameCallback) . . .
      else if (callback instanceof PasswordCallback) . . .
      else . . .
   }
}
```

  - Prepare Callbacks for handler

```
NameCallback nameCall = new NameCallback("username: ");
PasswordCallback passCall = new PasswordCallback("password: ", false);
callbackHandler.handle(new Callback[] { nameCall, passCall });
```

JAAS Login Module

REliable, INtelligent & Scalable Systems

- An Example: SimpleLoginModule.
  - SimpleLoginModule.java

```
import java.io.*;
import java.lang.reflect.*;
import java.security.*;
import java.util.*;
import javax.security.auth.*;
import javax.security.auth.login.*;
import javax.security.auth.callback.*;
import javax.security.auth.spi.*;

import javax.swing.*;

/**
  This login module authenticates users by reading
  usernames, passwords, and roles  from a text file.
*/
public class SimpleLoginModule implements LoginModule
{
```

JAAS Login Module

REliable, INtelligent & Scalable Systems

- An Example: SimpleLoginModule.
  - SimpleLoginModule.java

```
public void initialize(Subject subject, CallbackHandler callbackHandler,
  Map<String, ?> sharedState, Map<String, ?> options)
{
  this.subject = subject;
  this.callbackHandler = callbackHandler;
  this.sharedState = sharedState;
  this.options = options;
}

public boolean login() throws LoginException
{
  if (callbackHandler == null)
    throw new LoginException("no handler");

  NameCallback nameCall = new NameCallback("username: ");
  PasswordCallback passCall = new PasswordCallback("password: ", false);
  try
  {
    callbackHandler.handle(new Callback[] { nameCall, passCall });
  }
```

JAAS Login Module

REliable, INtelligent & Scalable Systems

- An Example: SimpleLoginModule.
  - SimpleLoginModule.java

```
    catch (UnsupportedCallbackException e)
    {
      LoginException e2 = new LoginException("Unsupported callback");
      e2.initCause(e);
      throw e2;
    }
    catch (IOException e)
    {
      LoginException e2 = new LoginException("I/O exception in callback");
      e2.initCause(e);
      throw e2;
    }

    return checkLogin(nameCall.getName(), passCall.getPassword());
  }
```

JAAS Login Module

REliable, INtelligent & Scalable Systems

- An Example: SimpleLoginModule.
  - SimpleLoginModule.java

```
/**
  Checks whether the authentication information is valid. If it is, the subject acquires
  principals for the user name and role.
  @param username the user name
  @param password a character array containing the password
  @return true if the authentication information is valid
*/
private boolean checkLogin(String username, char[] password) throws LoginException
{
  try
  {
    Scanner in = new Scanner(new FileReader("" + options.get("pwfile")));
    while (in.hasNextLine())
    {
      String[] inputs = in.nextLine().split("\\|");
      if (inputs[0].equals(username) && Arrays.equals(inputs[1].toCharArray(), password))
      { String role = inputs[2];
        Set<Principal> principals = subject.getPrincipals();
        principals.add(new SimplePrincipal("username", username));
        principals.add(new SimplePrincipal("role", role));
        return true;
      }
    }
```

JAAS Login Module

REliable, INtelligent & Scalable Systems

- An Example: SimpleLoginModule.
  - SimpleLoginModule.java

```
        in.close();
        return false;
      }
      catch (IOException e)
      {
        LoginException e2 = new LoginException("Can't open password file");
        e2.initCause(e);
        throw e2;
      }
    }

    public boolean logout() { return true; }
    public boolean abort() { return true; }
    public boolean commit() { return true; }

    private Subject subject;
    private CallbackHandler callbackHandler;
    private Map<String, ?> sharedState;
    private Map<String, ?> options;
}
```

JAAS Login Module

REliable, INtelligent & Scalable Systems

- An Example: SimpleLoginModule.
  - SimplePrincipal.java

```
import java.security.*;
/**
  A principal with a named value (such as "role=HR" or "username=harry").
*/
public class SimplePrincipal implements Principal
{
  /**
    Constructs a SimplePrincipal to hold a description and a value.
    @param roleName the role name
  */
  public SimplePrincipal(String descr, String value) {
    this.descr = descr; this.value = value;
  }

  /**
    Returns the role name of this principal
    @return the role name
  */
```

JAAS Login Module

REliable, INtelligent & Scalable Systems

- An Example: SimpleLoginModule.
  - SimplePrincipal.java

```
public String getName() { return descr + "=" + value; }

  public boolean equals(Object otherObject)
  {
    if (this == otherObject) return true;
    if (otherObject == null) return false;
    if (getClass() != otherObject.getClass()) return false;
    SimplePrincipal other = (SimplePrincipal) otherObject;
    return getName().equals(other.getName());
  }
  public int hashCode() { return getName().hashCode(); }

  private String descr;
  private String value;
}
```

JAAS Login Module

REliable, INtelligent & Scalable Systems

- An Example: SimpleLoginModule.
  - SimpleCallbackHandler.java

```
import javax.security.auth.callback.*;

/**
  This simple callback handler presents the given user name and password.
*/
public class SimpleCallbackHandler implements CallbackHandler
{
  /**
    Constructs the callback handler.
    @param username the user name
    @param password a character array containing the password
  */
  public SimpleCallbackHandler(String username, char[] password)
  {
    this.username = username;
    this.password = password;
  }
```

JAAS Login Module

REliable, INtelligent & Scalable Systems

- An Example: SimpleLoginModule.
  - SimpleCallbackHandler.java

```
public void handle(Callback[] callbacks)
 {
   for (Callback callback : callbacks)
   {
     if (callback instanceof NameCallback)
     {
       ((NameCallback) callback).setName(username);
     }
     else if (callback instanceof PasswordCallback)
     {
       ((PasswordCallback) callback).setPassword(password);
     }
   }
 }

 private String username;
 private char[] password;
}
```

JAAS Login Module

- An Example: SimpleLoginModule.
  - JAASTest.policy

```
grant codebase "file:login.jar"
{
  permission java.awt.AWTPermission "showWindowWithoutWarningBanner";
  permission javax.security.auth.AuthPermission "createLoginContext.Login1";
  permission javax.security.auth.AuthPermission "doAsPrivileged";
  permission javax.security.auth.AuthPermission "modifyPrincipals";
  permission java.io.FilePermission "password.txt", "read";
};

grant principal SimplePrincipal "role=admin"
{
  permission java.util.PropertyPermission "*", "read";
};
```

  - Jaas.config

```
Login1
{
  SimpleLoginModule required pwfile="password.txt";
};
```

JAAS Login Module

- An Example: SimpleLoginModule.
  - You must separate the login and action code. Create two JAR files:

```
javac *.java
jar cvf login.jar JAAS*.class Simple*.class
jar cvf action.jar SysPropAction.class
```

  - Then run the program as

```
java -classpath login.jar:action.jar
        -Djava.security.policy=JAASTest.policy
        -Djava.security.auth.login.config=jaas.config JAASTest
```

Digital Signatures

- To give more trust to an applet, we need to know two things:
  - Where did the applet come from?
  - Was the code corrupted in transit?

Message Digests

- A message digest is a digital fingerprint of a block of data.
  - For example, the so-called SHA1 (secure hash algorithm #1) condenses any data block, no matter how long, into a sequence of 160 bits (20 bytes).
- A message digest has two essential properties:
  - If one bit or several bits of the data are changed, then the message digest also changes.
  - A forger who is in possession of a given message cannot construct a fake message that has the same message digest as the original.

Message Digests

REliable, INtelligent & Scalable Systems

- Consider the following message by the billionaire father:
  - "Upon my death, my property shall be divided equally among my children; however, my son George shall receive nothing."
  - That message has an SHA1 fingerprint of
    - 2D 8B 35 F3 BF 49 CD B1 94 04 E0 66 21 2B 5E 57 70 49 E1 7E
  - Now, suppose George wants to change the message so that Bill gets nothing. That changes the fingerprint to a completely different bit pattern:
    - 2A 33 0B 4B B3 FE CC 1C 9D 5C 01 A7 09 51 0B 49 AC 8F 98 92

Message Digests

REliable, INtelligent & Scalable Systems

```
MessageDigest alg = MessageDigest.getInstance("SHA-1");

InputStream in = . . .
int ch;
while ((ch = in.read()) != -1)
  alg.update((byte) ch);

byte[] bytes = . . .;
alg.update(bytes);

byte[] hash = alg.digest();
```

Message Signing

- The message digest algorithms are publicly known, and they don't require secret keys.
  - In that case, the recipient of the forged message and the recomputed fingerprint would never know that the message has been altered.
  - Digital signatures solve this problem.

Message Signing

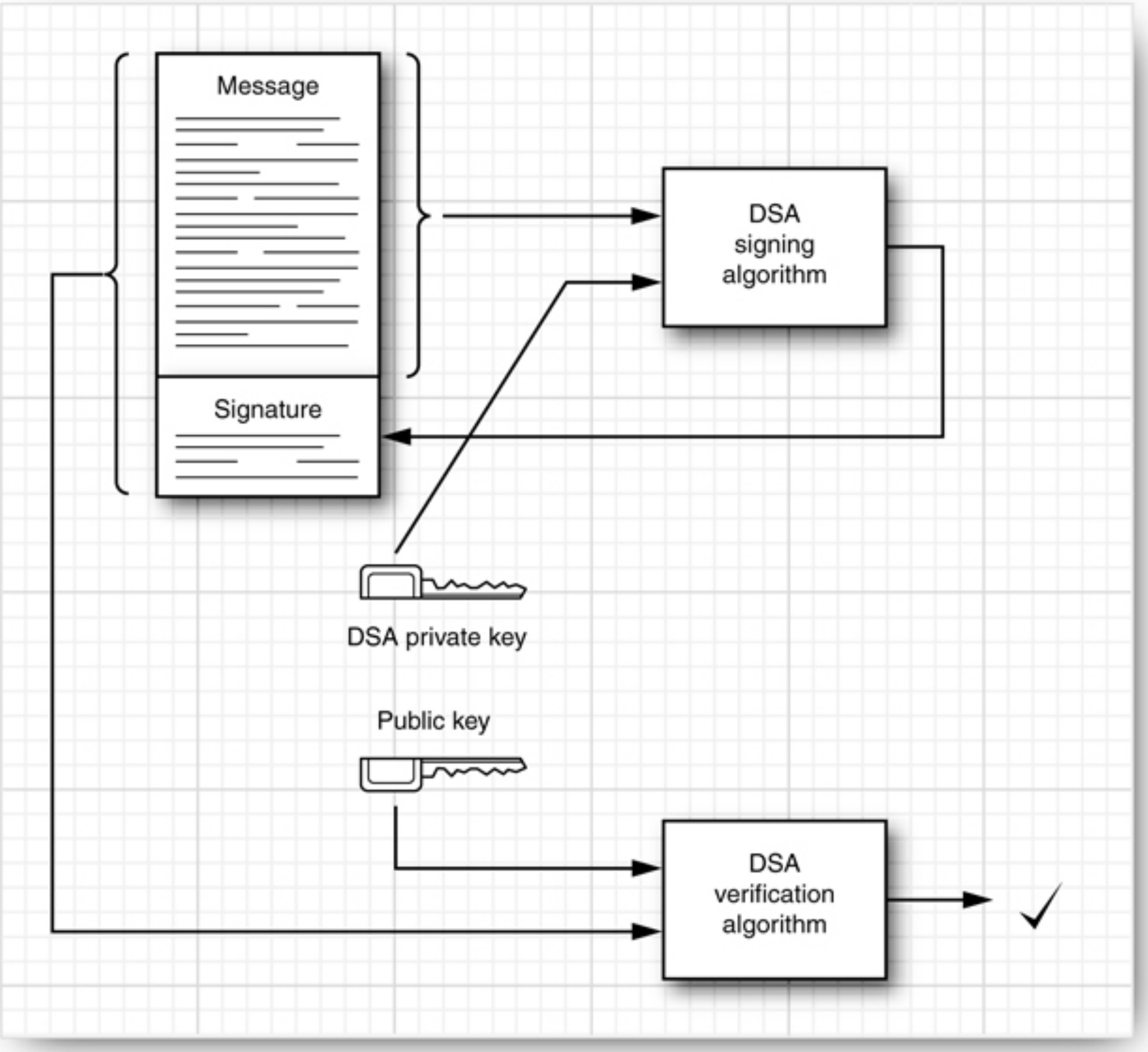
REliable, INtelligent & Scalable Systems

- The keys are quite long and complex. For example, here is a matching pair of public and private Digital Signature Algorithm (DSA) keys.
- Public key:
- Code View:
  - p: fca682ce8e12caba26efccf7110e526db078b05edecbcd1eb4a208f3ae1617ae01f35b91a47e6df63413c5e12ed0899 bcd132acd50d99151bdc43ee737592e17 q: 962eddcc369cba8ebb260ee6b6a126d9346e38c5 g:678471b27a9cf44ee91a49c5147db1a9aaf244f05a434d6486931d2d14271b9e35030b71fd73da179069b32e29356 30e 1c2062354d0da20a6c416e50be794ca4 y: c0b6e67b4ac098eb1a32c5f8c4c1f0e7e6fb9d832532e27d0bdab9ca2d2a8123ce5a8018b8161a760480fadd040b927 281ddb22cb9bc4df596d7de4d1b977d50

- Private key:
- Code View:
  - p: fca682ce8e12caba26efccf7110e526db078b05edecbcd1eb4a208f3ae1617ae01f35b91a47e6df63413c5e12ed0899 bcd132acd50d99151bdc43ee737592e17 q: 962eddcc369cba8ebb260ee6b6a126d9346e38c5 g: 678471b27a9cf44ee91a49c5147db1a9aaf244f05a434d6486931d2d14271b9e35030b71fd73da179069b32e2935630 e1c2062354d0da20a6c416e50be794ca4 x: 146c09f881656cc6c51f27ea6c3a91b85ed1d70a

Message Signing

REliable, INtelligent & Scalable Systems



X.509 Certificate

- To take advantage of public key cryptography, the public keys must be distributed.
  - One of the most common distribution formats is called X.509.

- The keytool program manages keystores, databases of certificates and private/public key pairs.
  - Each entry in the keystore has an alias.
  - Here is how Alice creates a keystore, alice.certs, and generates a key pair with alias alice.
  - keytool -genkeypair -keystore alice.certs -alias alice

X.509 Certificate

- When generating a key, you are prompted for the following information:

Enter keystore password: password
What is your first and last name?
  [Unknown]: Alice Lee
What is the name of your organizational unit?
  [Unknown]: Engineering Department
What is the name of your organization?
  [Unknown]: ACME Software
What is the name of your City or Locality?
  [Unknown]: Cupertino
What is the name of your State or Province?
  [Unknown]: California
What is the two-letter country code for this unit?
  [Unknown]: US
Is <CN=Alice Lee, OU=Engineering Department, O=ACME Software, L=Cupertino, ST=California,
C=US> correct?
[no]: Y

X.509 Certificate

REliable, INtelligent & Scalable Systems

- Alice exports a certificate file:
  - keytool -exportcert -keystore alice.certs -alias alice -file alice.cer

- Bob receives the certificate, he can print it:
  - keytool -printcert -file alice.cer

- The printout looks like this:

```
Owner: CN=Alice Lee, OU=Engineering Department, O=ACME Software,
       L=San Francisco, ST=CA, C=US
Issuer: CN=Alice Lee, OU=Engineering Department, O=ACME Software,
       L=San Francisco, ST=CA, C=US
Serial number: 470835ce
Valid from: Sat Oct 06 18:26:38 PDT 2007 until: Fri Jan 04 17:26:38 PST 2008
Certificate fingerprints:
     MD5: BC:18:15:27:85:69:48:B1:5A:C3:0B:1C:C6:11:B7:81
     SHA1: 31:0A:A0:B8:C2:8B:3B:B6:85:7C:EF:C0:57:E5:94:95:61:47:6D:34
     Signature algorithm name: SHA1withDSA
     Version: 3
```

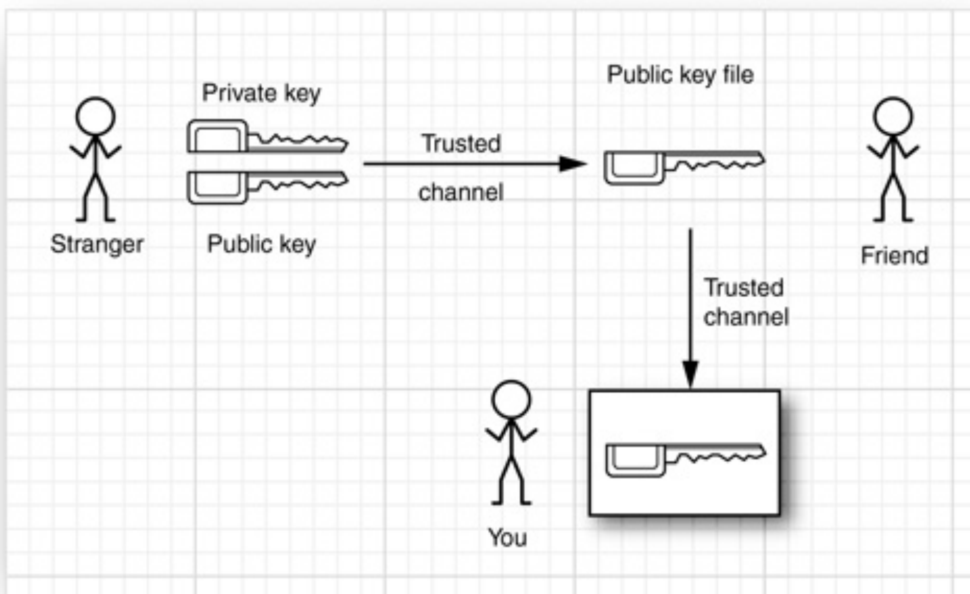
X.509 Certificate

REliable, INtelligent & Scalable Systems

- Once Bob trusts the certificate, he can import it into his keystore.
  - keytool -importcert -keystore bob.certs -alias alice -file alice.cer
- Now Alice can start sending signed documents to Bob.
  - jar cvf document.jar document.txt
  - jarsigner -keystore alice.certs document.jar alice
- When Bob receives the file, he uses the -verify option of the jarsigner program.
  - jarsigner -verify -keystore bob.certs document.jar
- If the JAR file is not corrupted and the signature matches, then the jarsigner program prints
  - jar verified.
  - Otherwise, the program displays an error message.

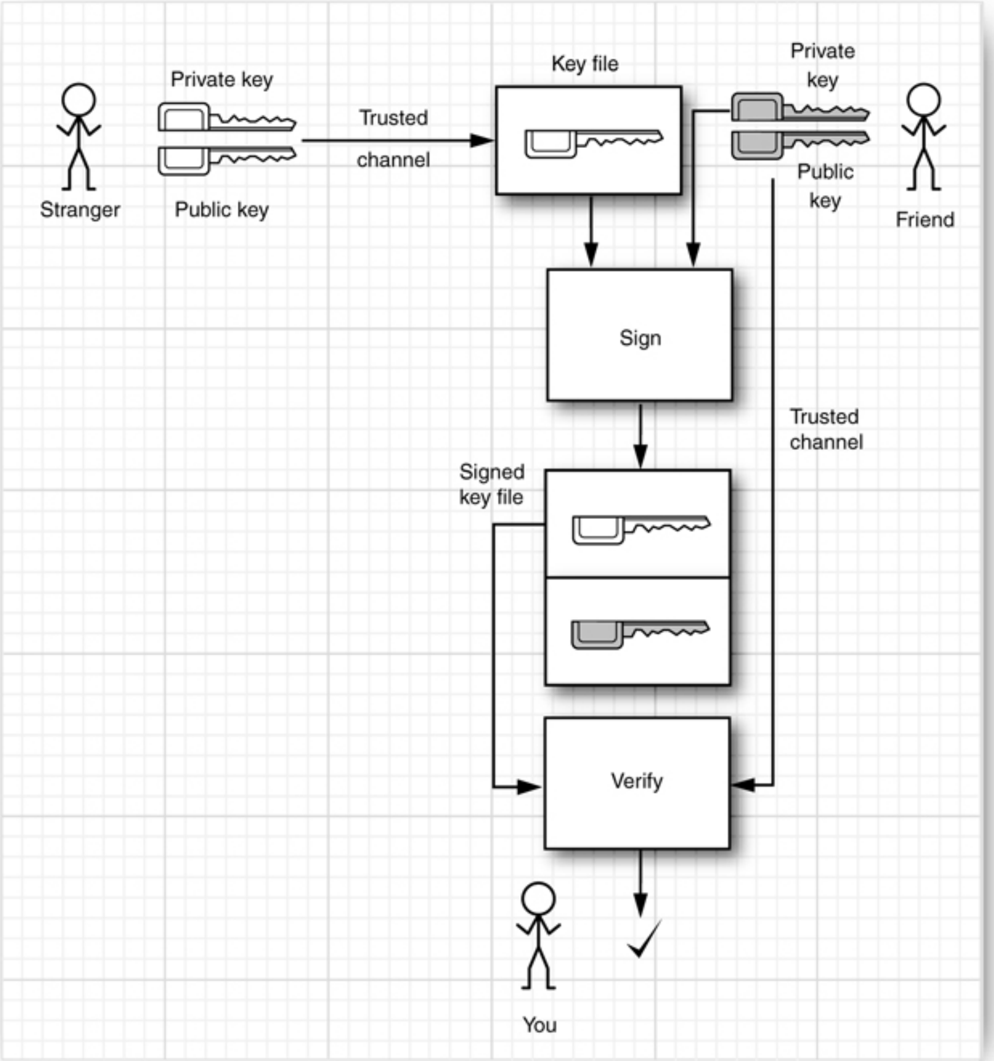
Authentication Problem

- Be careful:
  - You still have no idea who wrote the message. Anyone could have generated a pair of public and private keys, signed the message with the private key, and sent the signed message and the public key to you.
  - The problem of determining the identity of the sender is called the authentication problem.



Authentication Problem

REliable, INtelligent & Scalable Systems



Certificate Signing

- Suppose Alice wants to send her colleague Cindy a signed message
  - but Cindy doesn't want to bother with verifying lots of signature fingerprints.
  - Now suppose that there is an entity that Cindy trusts to verify signatures. In this example, Cindy trusts the Information Resources Department at ACME Software.
- That department operates a certificate authority (CA).
  - Everyone at ACME has the CA's public key in their keystore, installed by a system administrator who carefully checked the key fingerprint.
  - The CA signs the keys of ACME employees.
  - When they install each other's keys, then the keystore will trust them implicitly because they are signed by a trusted key.

Certificate Signing

- Here is how you can simulate this process.
  - Create a keystore acmesoft.certs.
  - Generate a key par and export the public key:
    - keytool -genkeypair -keystore acmesoft.certs -alias acmeroot
    - keytool -exportcert -keystore acmesoft.certs -alias acmeroot -file acmeroot.cer
  - The public key is exported into a "self-signed" certificate.
  - Then add it to every employee's keystore.
    - keytool -importcert -keystore cindy.certs -alias acmeroot -file acmeroot.cer
  - An authorized staff member at ACME Software would verify Alice's identity and generate a signed certificate as follows:
    - java CertificateSigner -keystore acmesoft.certs -alias acmeroot -infile alice.cer -outfile alice_signedby_acmeroot.cer
  - Now Cindy imports the signed certificate into her keystore:
    - keytool -importcert -keystore cindy.certs -alias alice -file alice_signedby_acmeroot.cer

Code Signing

- One of the most important uses of authentication technology is signing executable programs.

- You now know how to implement this sophisticated scheme.
  - Use authentication to verify where the code came from.
  - Run the code with a security policy that enforces the permissions that you want to grant the program, depending on its origin.

JAR File Signing

- ACME decides to sign the JAR files that contain the program code.
  - First, ACME generates a root certificate:
    - keytool -genkeypair -keystore acmesoft.certs -alias acmeroot
  - Therefore, we create a second keystore client.certs for the public certificates and add the public acmeroot certificate into it.
    - keytool -exportcert -keystore acmesoft.certs -alias acmeroot -file acmeroot.cer
    - keytool -importcert -keystore client.certs -alias acmeroot -file acmeroot.cer
  - To make a signed JAR file, programmers add their class files to a JAR file in the usual way. For example,
    - javac FileReadApplet.java jar cvf FileReadApplet.jar *.class
  - Then a trusted person at ACME runs the jarsigner tool, specifying the JAR file and the alias of the private key:
    - jarsigner -keystore acmesoft.certs FileReadApplet.jar acmeroot

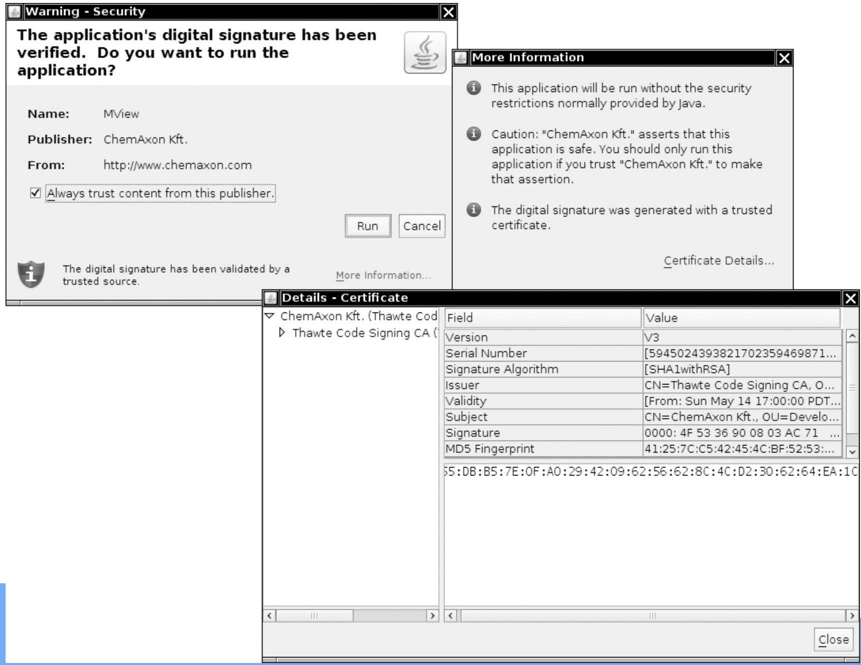
JAR File Signing

- ACME decides to sign the JAR files.
  - Next, let us turn to the client machine configuration. A policy file must be distributed to each client machine.
  - To reference a keystore, a policy file starts with the line
    - keystore "keystoreURL", "keystoreType";
  - The URL can be absolute or relative.
    - keystore "client.certs", "JKS";
  - Then grant clauses can have suffixes signedBy "alias", such as this one:
    - grant signedBy "acmeroot" { . . . };
  - Now create a policy file applet.policy with the contents:
    - keystore "client.certs", "JKS";
    - grant signedBy "acmeroot" {
    - permission java.lang.RuntimePermission "usePolicy";
    - permission java.io.FilePermission "/etc/*", "read";
    - };

Software Developer Certificates

REliable, INtelligent & Scalable Systems

- A program signed with a software developer certificate that is issued by a CA will trigger a pop-up dialog box identifies the software developer and the certificate issuer.
  - You now have two choices:
    - Run the program with full privileges.
    - Confine the program to the sandbox. (The Cancel button in the dialog box is misleading. If you click that button, the applet is not canceled. Instead, it runs in the sandbox.)



Symmetric Ciphers

- Cipher

  Cipher cipher = Cipher.getInstance(algorithmName);

  – or

  Cipher cipher = Cipher.getInstance(algorithmName, providerName);

  – The JDK comes with ciphers by the provider named "SunJCE".
  – The algorithm name is a string such as "AES" or "DES/CBC/PKCS5Padding".

  int mode = . . .; Key key = . . .; cipher.init(mode, key);

  – The mode is one of

  Cipher.ENCRYPT_MODE

  Cipher.DECRYPT_MODE

  Cipher.WRAP_MODE

  Cipher.UNWRAP_MODE

Symmetric Ciphers

REliable, INtelligent & Scalable Systems

```
int blockSize = cipher.getBlockSize();
byte[] inBytes = new byte[blockSize];
. . . // read inBytes
int outputSize= cipher.getOutputSize(inLength);
byte[] outBytes = new byte[outputSize];
int outLength = cipher.update(inBytes, 0, outputSize, outBytes);
. . . // write outBytes

outBytes = cipher.doFinal(inBytes, 0, inLength);
```

- Or

```
outBytes = cipher.doFinal();
```

- The call to doFinal is necessary to carry out padding of the final block.

L 01	if length(L) = 7
L 02 02	if length(L) = 6
L 03 03 03	if length(L) = 5
. . .	
L 07 07 07 07 07 07 07	if length(L) = 1
08 08 08 08 08 08 08 08	

Key generation

REliable, INtelligent & Scalable Systems

- Follow these steps:
  - Get a KeyGenerator for your algorithm.
  - Initialize the generator with a source for randomness. If the block length of the cipher is variable, also specify the desired block length.
  - Call the generateKey method.

```
KeyGenerator keygen = KeyGenerator.getInstance("AES");
SecureRandom random = new SecureRandom();
keygen.init(random);
Key key = keygen.generateKey();
```

Or

```
SecretKeyFactory keyFactory = SecretKeyFactory.getInstance("AES");
byte[] keyData = . . .; // 16 bytes for AES
SecretKeySpec keySpec = new SecretKeySpec(keyData, "AES");
Key key = keyFactory.generateSecret(keySpec);
```

Cipher Streams

- The JCE library provides a convenient set of stream classes that automatically encrypt or decrypt stream data.
- Encryption

```
Cipher cipher = . . .;
cipher.init(Cipher.ENCRYPT_MODE, key);
CipherOutputStream out = new CipherOutputStream(new
FileOutputStream(outputFileName), cipher);
byte[] bytes = new byte[BLOCKSIZE];
int inLength = getData(bytes); // get data from data source
while (inLength != -1) {
    out.write(bytes, 0, inLength);
    inLength = getData(bytes); // get more data from data source
} out.flush();
```

Cipher Streams

- The JCE library provides a convenient set of stream classes that automatically encrypt or decrypt stream data.
- Decryption

```
Cipher cipher = . . .;
cipher.init(Cipher.DECRYPT_MODE, key);
CipherInputStream in = new CipherInputStream(new
FileInputStream(inputFileName), cipher);
byte[] bytes = new byte[BLOCKSIZE];
int inLength = in.read(bytes);
while (inLength != -1) {
    putData(bytes, inLength); // put data to destination
    inLength = in.read(bytes);
}
```

Public Key Ciphers

- The Achilles heel of symmetric ciphers is key distribution.
  - Public key cryptography solves that problem.
- All known public key algorithms are much slower than symmetric key algorithms such as DES or AES.
  - It would not be practical to use a public key algorithm to encrypt large amounts of information.
- This problem can easily be overcome by combining a public key cipher with a fast symmetric cipher, like this:
  - Alice generates a random symmetric encryption key. She uses it to encrypt her plaintext.
  - Alice encrypts the symmetric key with Bob's public key.
  - Alice sends Bob both the encrypted symmetric key and the encrypted plaintext.
  - Bob uses his private key to decrypt the symmetric key.
  - Bob uses the decrypted symmetric key to decrypt the message.

References

- Core Java (volume II) 9th edition
  - http://horstmann.com/corejava.html
- The Java EE 7 Tutorial
  - http://docs.oracle.com/javaee/7/tutorial/doc/javaeetutorial7.pdf

Thank You!