Java Security Mechanism

Jing Deng
Foundations of Computer and Network Security
Computer Science Department
Tuesday, November 27, 2001

Outline

- Java security architecture
- Control applets and applications
- Signature and Certificate
- Tools for signing and verifying code
- API for signing and verifying code
- Security manager API
- Example

References

- http://java.sun.com/docs/books/tutorial/security1.2/index.html
- http://www.securingjava.com/toc.html
- http://developer.java.sun.com/developer/onlineTraining/Security/Fundamentals/index.html
- Java Security, *Scott Oaks*, O'Reilly, c1998

Java Security

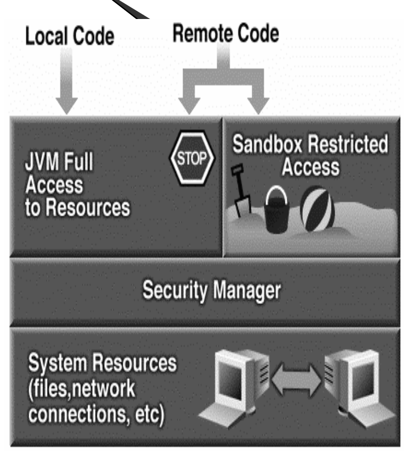
- No pointer arithmetic operations
- Clearly defines behavior of uninitialized variables
- Garbage collection
- Strong type checking

Java Security

- Sandbox
- Byte code verification
- Runtime checking
- Managing security
  - Security manager
  - Security policy
  - Access controller and permissions

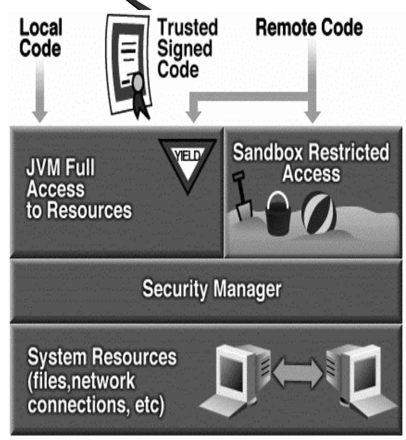
JDK 1.0 -- Sandbox

- Local code has full access to vital resources
- Remote code can only access limited resources inside the sandbox



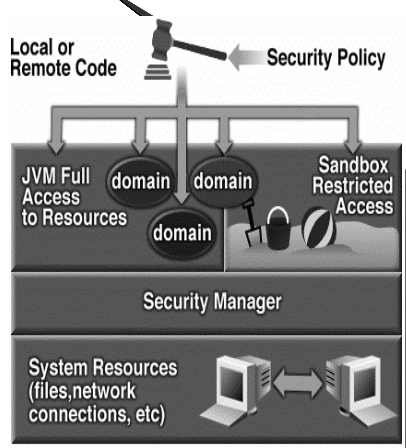
JDK 1.1– “Signed Applet”

- A signed applet is treated like local code
- Unsigned applet still runs in sandbox
- Signed applets are delivered in signed JAR file



JDK 1.2 – Security Policy

- policy-based, easily-configurable, fine-grained access control
- new cryptographic services, certificate, key management classes and interfaces



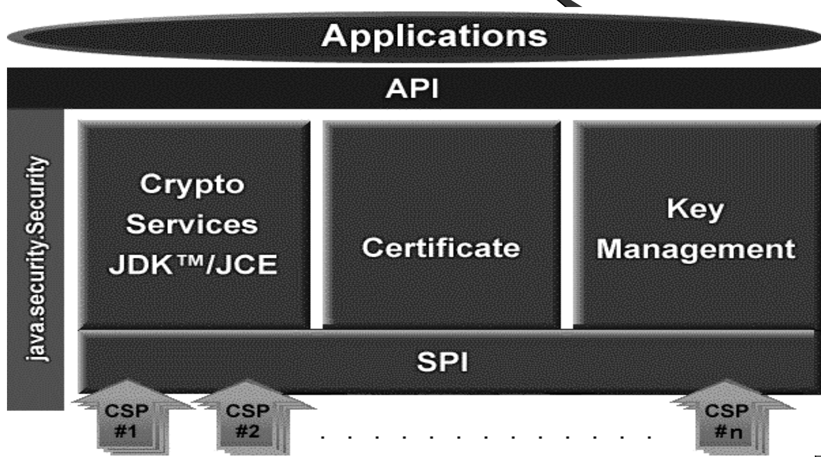
Cryptography Architecture

- Java cryptography architecture (JCA)
  - a framework for accessing and developing cryptographic functionality for the Java platform
- Cryptographic service provider (CSP)
  - A package (or a set of package) that supplies a concrete implementation of a subset of the cryptography aspects of the JDK security API
  - It includes…

Cryptographic Service Provider

- JDK 1.1
  - Digital signature
  - Message digest
  - Key generation
- JDK 1.2
  - Keystore creation and management
  - Algorithms parameter management
  - Algorithms parameter generation
  - Key factory support to convert between different key representations
  - Random number generation (RNG)

Cryptography Architecture Extensions



Tools

- ***Policytool***
  - Create and modify the external policy configuration files
- ***keytool***
  - Create public/private key pairs, issue certificate requests, import certificate replies, designate public keys belonging to other parties as trusted, and manage keystore.
- ***Jarsigner***
  - Sign JAR files, verify signatures on signed JAR files.

Control Applets

- Default permission of applet
- Use *policytool* to define a policy file
  - Add policy entry
  - Define code base
  - Add permission
- Use the policy file to define the permission of the applet

Control Applets

- Default security properties file
  - ${java.home}/lib/security/java.security
- Use command line to indicate policy file
  - appletviewer -J-Djava.security.policy=$DIR/mypolicy http://…
- Add entry into security properties file
  - Policy.url.3=file:${DIR}/test/mypolicy
  - appletviewer http://…

Control Applications

- Default permission of applications
- Security manager restricts the permission of applications
- Use *policytool* to define a policy file
  - Add policy entry
  - Define code base
  - Add permission
- Use the policy file to define the permission of the application

Control Applications

- Default security properties file
  - ${java.home}/lib/security/java.security
- Use command line to indicate policy file
  - appletviewer -J-Djava.security.policy=$DIR/mypolicy http://…
- Add entry into security properties file
  - Policy.url.3=file:${DIR}/test/mypolicy
  - appletviewer http://…

Digital Signatures

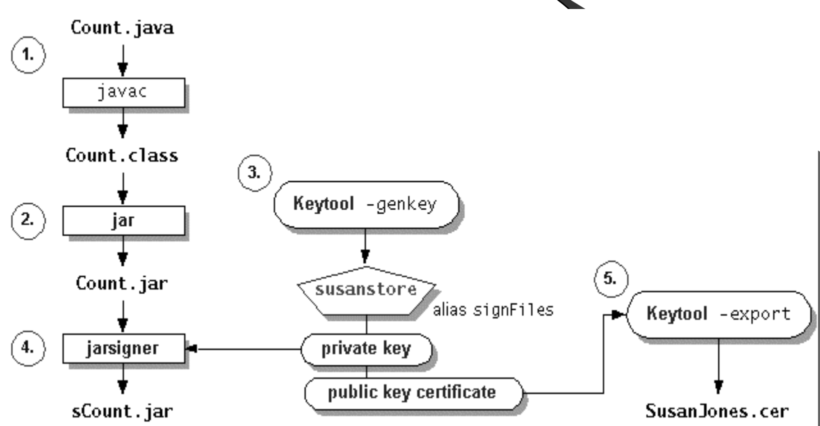
- Signs the document with private key
- Sends to receiver the document and the signature
- Sends to receiver the public key with certificate

- Receiver uses the public key to verify the authentication and integrity of the document

Certificates

- A public key
- “distinguished-name” information of the entity
- A digital signature issued by CA

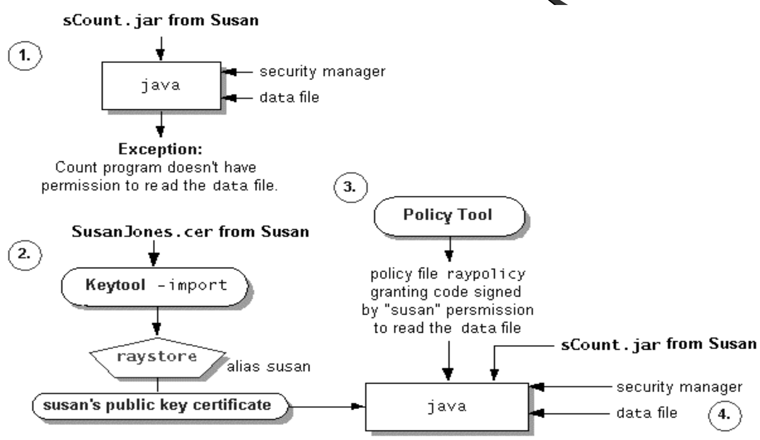
Tool - Sign Code



Tool - Sign Code

- Generate keys
  - *Keytool –genkey –alias signFiles –keypass kpi135 –keystore susanstore –storepass ab987c*
- Sign the JAR file
  - *jarsigner -keystore susanstore -signedjar sCount.jar Count.jar signFiles*
- Export the public key certificate
  - *keytool -export -keystore susanstore -alias signFiles -file SusanJones.cer*

Tool - Verify Code



Tool - Verify Signature and Grant Code

- Execute code directly
- Execute code with security manager
- Use *keytool* to import certificate
  – *keytool -import -alias susan -file SusanJones.cer -keystore raystore*
- Use *keytool* to print certificate information
  – *keytool -printcert -file SusanJones.cer*
- Use *policytool* to set permission

API - Generator Key Pair

- Define and initialize key pair generator
  - KeyPairGenerator keyGen = KeyPairGenerator.getInstance("DSA", "SUN");
  - SecureRandom random = SecureRandom.getInstance("SHA1PRNG", "SUN");
  - keyGen.initialize(1024, random);
- Generator public key and private key
  - KeyPair pair = keyGen.generateKeyPair();
  - PrivateKey priv = pair.getPrivate();
  - PublicKey pub = pair.getPublic();

API - Sign the Data

- Initialize the Signature Object
  - Signature dsa = Signature.getInstance("SHA1withDSA", "SUN");
  - dsa.initSign(priv);
- Get data
  - Read data from file and put into dsa
- Generate the signature
  - byte[] realSig = dsa.sign();
- Write signature and public key to files
  - sigfos.write(realSig);
  - keyfos.write(pub.getEncoded());

API - Verify Signature

- Read encoded public key fiile
  - Read file to byte[] encKey
- Initialize DSA public key
  - X509EncodedKeySpec pubKeySpec = new X509EncodedKeySpec(encKey);
  - KeyFactory keyFactory = KeyFactory.getInstance("DSA", "SUN");
- Generate public key
  - PublicKey pubKey = keyFactory.generatePublic(pubKeySpec);

API - Verify Signature

- Read signature
  - From private key file to byte[] sigToVerify
- Initialize signature object
  - Signature sig = Signature.getInstance("SHA1withDSA", "SUN");
  - sig.initVerify(pubKey);
- Send data to signature object
  - Read signature file and send data to sig
- Verify signature
  - boolean verifies = sig.verify(sigToVerify);

API for Policy Management

- Install security manager
  - *System.getSecurityManager()*
- Check permission
  - *checkPermission()*
- Get temporarily permission
  - *doPrivileged()*

Example

- Developers Terrys and Chris
  - Create a keystore and keys for signing
  - Sign the JAR file
  - Export public key certificate
- User Kim
  - Import certificates of Terrys and Chris
  - Setup policy file
  - Run program

Policy File Example

- Define keystore file
- Define *grant Signed by*
- Define permission for
  - File (read, write, etc…)
  - Class (read, write, etc…)