Lab Notes

CMPU4021 Distributed Systems

Securing Java

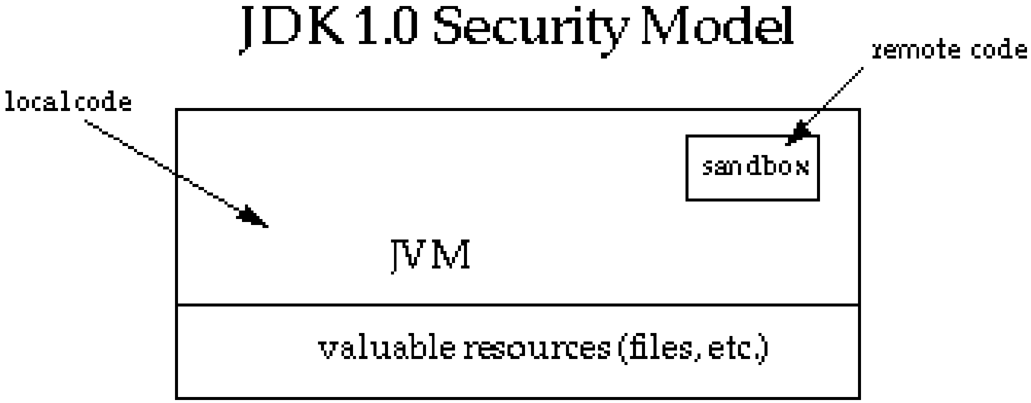
Java 2 Platform Security

- Partly done with language features such as
    - Automatic memory management
    - Garbage collection
    - Range checking on strings and arrays

- Code is executed in the Java Virtual Machine (JVM).
    - Compiler and bytecode verifier ensure that only legitimate Java bytecodes are executed.
    - Bytecode verifier and JVM guarantee language safety at run time.
    - Classloader defines a local name space:
        - An untrusted applet cannot interfere with the running of other programs.
    - JVM mediates access to crucial system resources.
        - SecurityManager class restricts the actions of Java code.

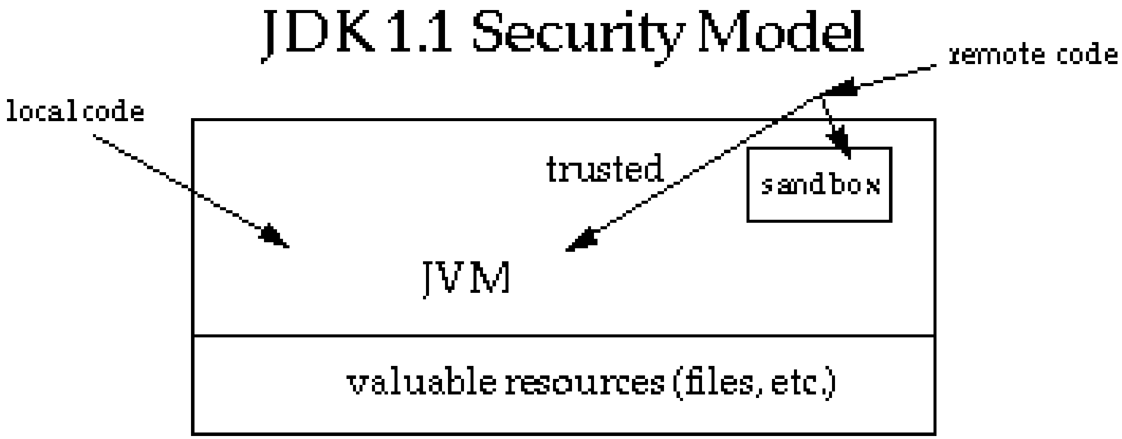
Java 2 Platform Security

- Original Sandbox Model
  - Code is executed in the Java Virtual Machine (JVM).
    - JVM simulates execution of Java Byte Code.
  - Sandbox model allows code to run in a very restricted environment.
  - But, local code has full access to valuable system resources.



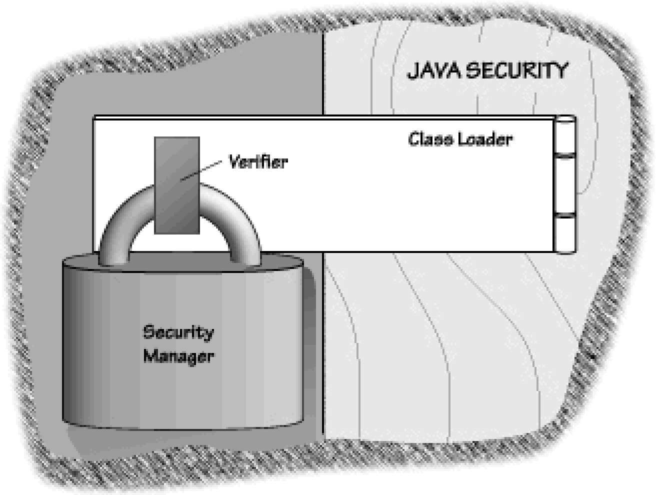
The Sandbox Model

- JDK 1.1: “Signed Applet”
  - – Correctly signed applet is treated as trusted applet.
  - – Applets and signatures are delivered in the JAR (Java Archive) format.
  - – Untrusted applets run in the sandbox.



Java Sandbox

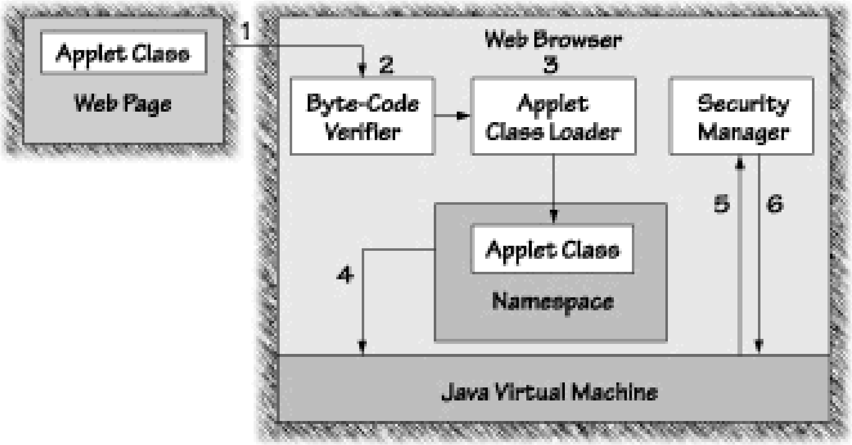
- The default sandbox is made of three interrelated parts:
  - The *Verifier* - helps ensure type safety.
  - The *Class Loader* - loads and unloads classes dynamically from the Java runtime environment.
  - The *Security Manager* - acts as a security gatekeeper guarding potentially dangerous functionality.



Java Sandbox

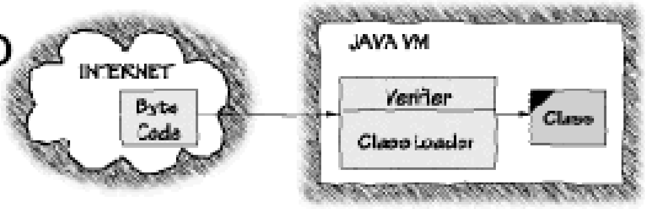
Browser asked to download applet:

1. Fetches the code from the Web
2. Verifies it
3. Instantiates it as a class
4. Applet executes and invokes a dangerous method causing the Security Manager to be consulted before the method runs.
5. The Security Manager performs runtime checks based on the calling class's origin and may veto some activities.



The Verifier

- When Java code arrives at the VM and is formed into a Class by the Class Loader, the Verifier examines it to
  - Make sure that the format of a code fragment is correct.
  - Make sure that byte code does not forge pointers, violate access restrictions, or access objects using incorrect type information.



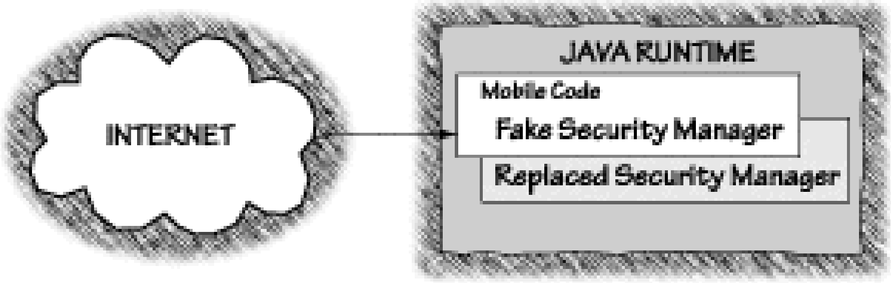
- If the Verifier discovers a problem with a class file, it throws an exception, loading ceases, and the class file never executes

The Verifier

- Once byte code passes through verification, the following things are guaranteed:
  - The class file has the correct format
  - Stacks will not be overflowed or underflowed
  - Byte code instructions all have parameters of the correct type
  - No illegal data conversions (casts) occur
  - Private, public, protected, and default accesses are legal
  - All register accesses and stores are valid

The Class Loader

- Every mobile code system requires the ability to load code from outside a system into the system dynamically
- Class loaders determine when and how classes can be added to a running Java environment
- The fake Security Manager shown must be disallowed from loading into the Java environment and replacing the real Security Manager. This is known as *class spoofing*.



The Security Manager

- The job of the Security Manager is to keep track of who is allowed to do which dangerous operations.
- A standard Security Manager will disallow most operations when they are requested by untrusted code, and will allow trusted code to do whatever it wants.
- Java's Security Manager works as follows:

  1. A Java program makes a call to a

  potentially dangerous operation in the Java API.

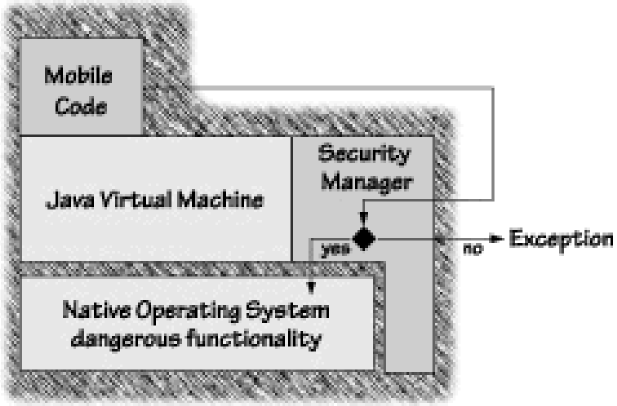
  2. The Java API code asks the

  Security Manager whether the operation

  should be allowed

  3. The Security Manager throws

  a SecurityException if the operation is denied.



Evolving the Sandbox Model: Java 2 Platform Security Model

- Easily configurable security policy.
  - Allows application builders and users to configure security policies without having to program

- Easily extensible access control structure.
  - The new architecture allows typed permissions (each representing an access to a system resource) and automatic handling of all permissions (including yet-to-be-defined permissions) of the correct type.
  - No new method in the `SecurityManager` class needs to be created in most cases.

Java 2 Platform Security Model

- Extension of security checks to all Java programs, including applications as well as applets.

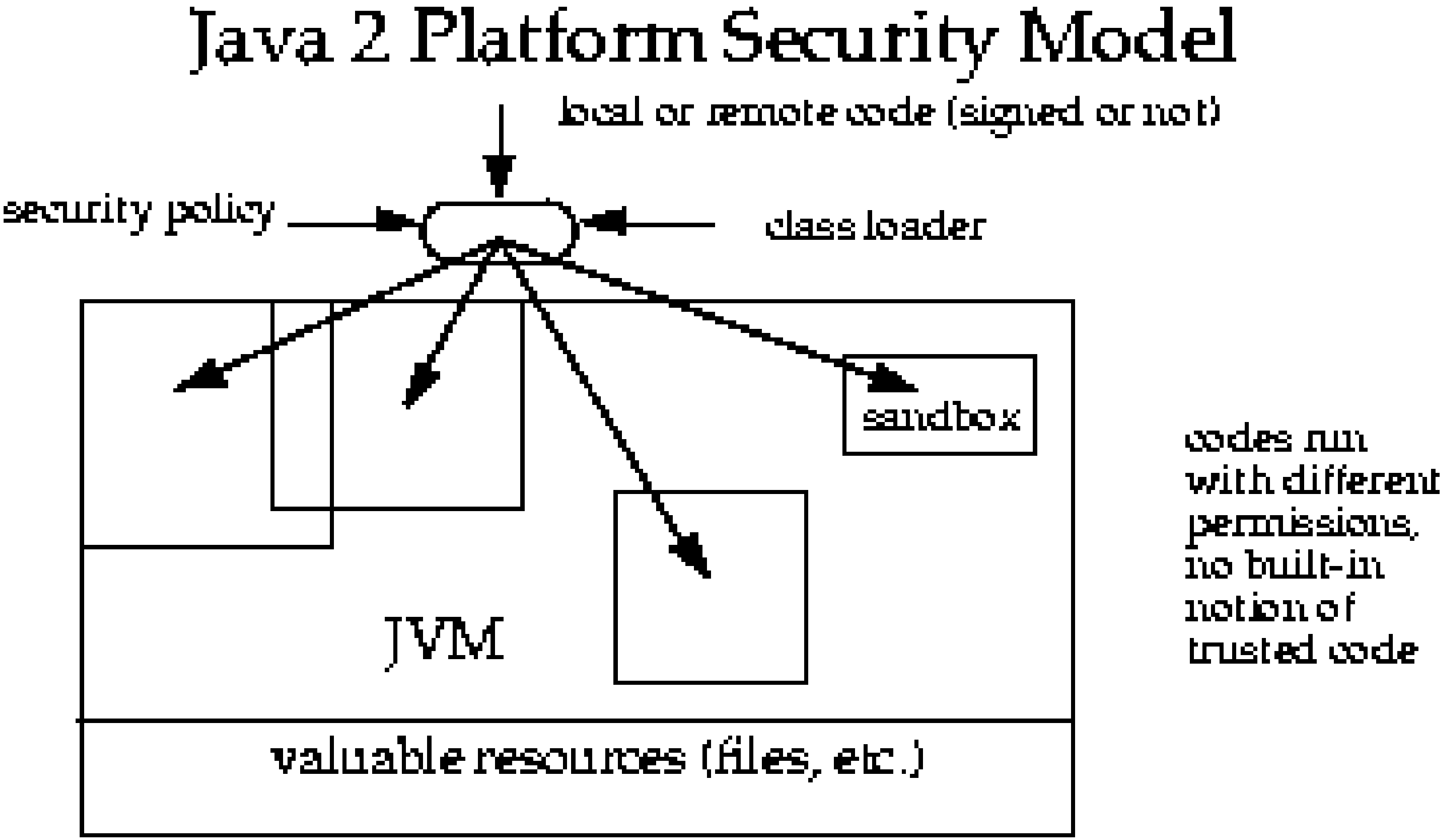
- There is no longer a built-in concept that all local code is trusted.

- Local code (e.g., non-system code, application packages installed on the local file system)
  - is subjected to the same security control as applets, although it is possible, if desired, to declare that the policy on local code (or remote code) be the most liberal, thus enabling such code to effectively run as totally trusted.

Java 2 Platform Security

- Fine-grained access control
- Previously, the application writer had to do substantial programming
  - e.g., by subclassing and customizing the SecurityManager and ClassLoader classes.
- Easily extensible access control structure.
- Extension of security checks to all Java programs, including applications as well as applets.
- Trust of local code is no longer a built-in concept.

Java 2 Platform Security



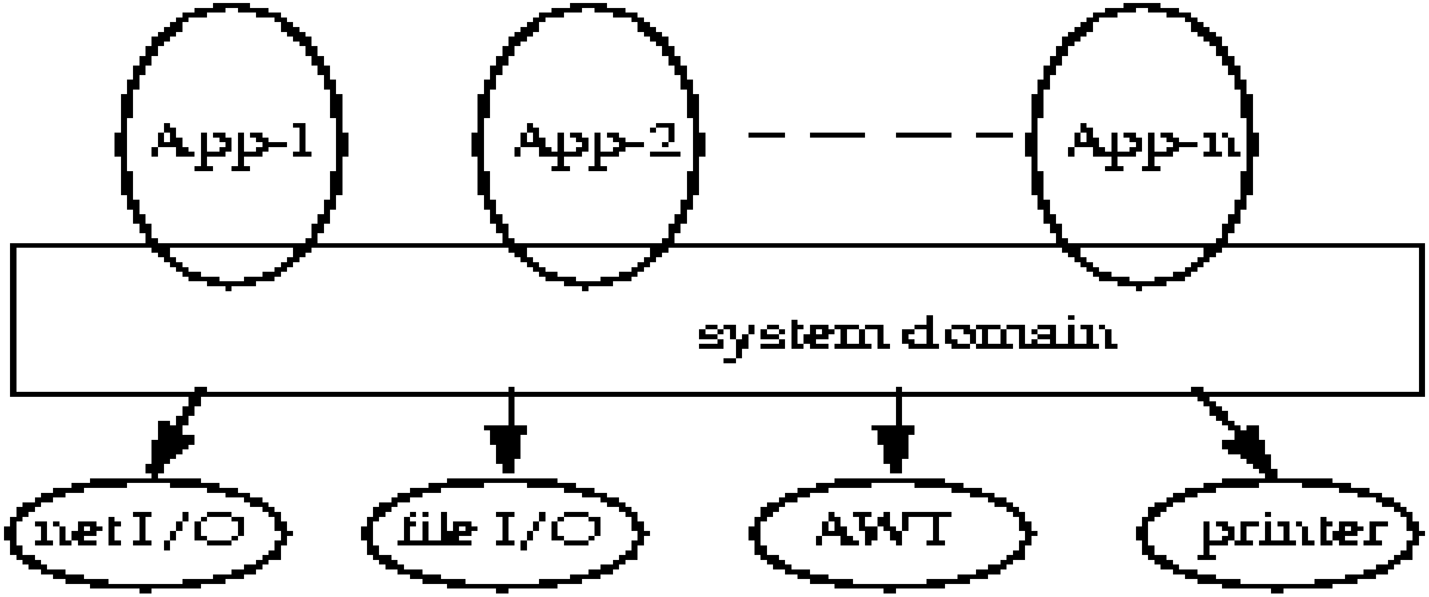
Java 2 Platform Security Model
local or remote code (signed or not)
security policy
class loader
sandbox
JVM
valuable resources (files, etc.)
codes run with different permissions, no built-in notion of trusted code

Java 2 Platform Security: Protection Domains

- Protection Domains
  - Set of objects that are currently directly accessible by a principal.
  - Principal is an entity in the computer system to which permissions are granted.
  - Serves to group and to isolate between units of protection.
  - Protection domains are either system domains or application domains.

Java 2 Platform Security: Protection Domains

- Protection domains generally fall into two distinct categories:
  - system domain
  - application domain.
- It is important that all protected external resources, such as the file system, the networking facility, and the screen and keyboard, be accessible only via system domains.

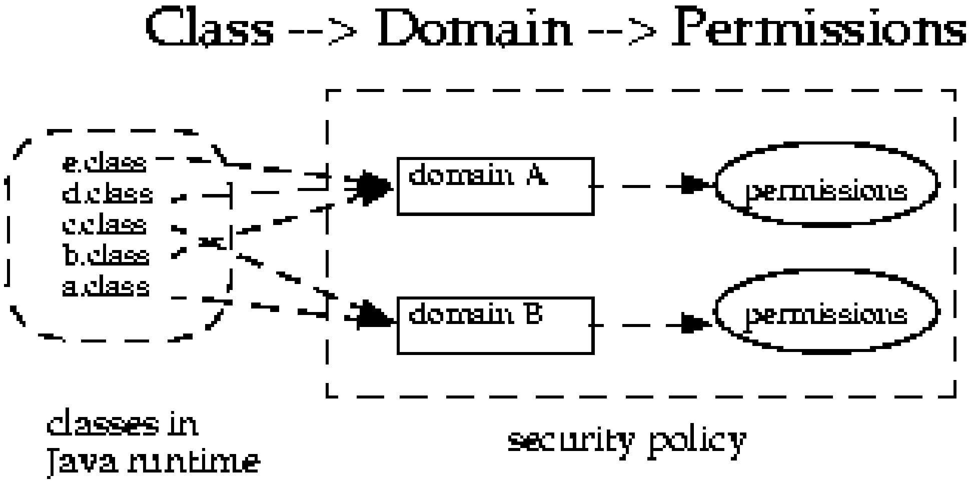


Protection Domains … cont’d

- A domain conceptually encloses a set of classes whose instances are granted the same set of permissions.

- Protection domains are determined by the policy currently in effect.

- The Java application environment maintains a mapping from code (classes and instances) to their protection domains and then to their permissions.



Java 2 Platform Security: Protection Domains

- A domain conceptually encloses a set of classes whose instances are granted the same set of permissions.

- Protection domains are determined by the policy currently in effect.

- The Java application environment maintains a mapping from code (classes and instances) to their protection domains and then to their permissions.

Java 2 Platform Security: Protection Domains

- Java thread can completely occur within a single protection domain.

- Can also involve application domain and system domain.

- Examples:
  - Application prints out a message.
  - Needs to interact with system domain that is the access point to an output stream.
  - AWT system domain calls an applet’s paint method to display it.

- Important:
  - A less "powerful" domain can NOT gain additional permissions as a result of calling or being called by a more powerful domain.

Java 2 Platform Security: Protection Domains

- Normal rule:
  - The permission set of an execution thread is the intersection of the permissions of all protection domains traversed by the execution thread.
  - Exception: doPrivileged call
    - Enables a piece of trusted code to temporarily enable access to more resources than are available directly to the application that called it.
  - Example:
    - Application may not be allowed direct access to files that contain files, but the system utility displaying those fonts needs to obtain them on behalf of the user.

Java 2 Platform Security: Protection Domains

- When access to a critical system resource (such as file I/O and network I/O) is requested:
  - the resource-handling code invokes a special AccessController class method
    - Evaluates the request
    - Decides if the request should be granted or denied.

Java 2 Platform Security: Protection Domains

- Each domain needs to implement additional protection of internal resources.

- Example:
  - Banking application needs to maintain internal concepts of
    - checking accounts
    - deposits
    - withdrawals

Permissions and Security Policy

- The permission classes represent access to system resources.
- `Java.security.Permission` class is an abstract class and is subclassed, as appropriate, to represent specific accesses.

- As an example of a permission, the following code can be used to produce a permission to read the file named "abc" in the /tmp directory:

```
perm = new java.io.FilePermission("/tmp/abc", "read");
```

`java.security.ProtectionDomain`

- `ProtectionDomain` class encapsulates the characteristics of a domain.

- Such a domain encloses a set of classes whose instances are granted a set of permissions when being executed on behalf of a given set of `Principals`.

Java 2 Platform Security Why:

- Original Problem:
    - Users download programs that contain viruses and worms (even in commercial software).
    - Java machines executes downloaded codes, which make the problem worse.
  - Early work focuses on this issue:
    - Java programs are secure because they cannot install, run, or propagate viruses.

- Java 2
  The changes incorporated at this stage were to make the security solutions on the Java platform easy to use and more robust, thereby correcting limitations of earlier platform versions.

PUBLIC KEY AUTHENTICATION

Public Key Authentication

- Demonstrate we can encrypt or decrypt a nonce (a random bunch of bits)
  - This shows we know the key

- Alice wants to authenticate herself to Bob:
- Bob: generates nonce, *S*
  - Sends it to Alice
- Alice: encrypts *S* with her private key (signs it)
  - Sends result to Bob

Public key authentication

Bob:

1. Look up “alice” in a database of public keys
2. Decrypt the message from Alice using Alice’s public key
3. If the result is S, then Bob is convinced he’s talking with Alice

For *mutual authentication*

Alice has to present Bob with a nonce that Bob will encrypt with his private key and return

Public key authentication

- Public key authentication relies on binding identity to a public key
  - How do you know it really is Alice’s public key?
- One option:
  - get keys from a trusted source
  - Problem: requires always going to the source
    - Can not pass keys around
- Another option:
  - sign the public key
    - Contents cannot be modified
    - digital certificate

Certificate standards and certificate authorities

- X.509 is the most widely used standard format for public key certificates

- X509 Certificate format

Subject	Distinguished Name, Public Key
Issuer	Distinguished Name, Signature
Period of validity	Not Before Date, Not After Date
Administrative information	Version, Serial Number
Extended Information	

Reminder

- Digital signature
  - Hash of a message encrypted with the signer’s private key

- A hash function
  - used to only verify the message integrity
  - takes an input (or 'message') and returns a fixed-size string of bytes
    - the string is called the 'hash value', 'message digest', 'digital fingerprint', 'digest' or 'checksum'.
  - if a message changes, the **hash of** a message will change

Public Key Infrastructure

- Public Key Infrastructure (PKI) is a term used for a framework that enables secure exchange of information based on public key cryptography.

- It allows identities (of people, organizations, etc.) to be bound to digital certificates and provides a means of verifying the authenticity of certificates.

- PKI encompasses
  - Keys
  - Certificates
  - public key encryption, and
  - trusted Certification Authorities (CAs) who generate and digitally sign certificates.

The Secure Socket Layer (SSL)

The Secure Socket Layer (SSL)

- SSL is a widely-used system component that supports secure and authenticated communication.

- Key distribution and secure channels for internet commerce
  - Hybrid protocol; depends on public-key cryptography
  - Originally developed by Netscape Corporation (1994)
  - Extended and adopted as an Internet standard with the name *Transport Level Security (TLS)*
  - Provides the security in all web servers and browsers and in secure versions of Telnet, FTP and other network applications;
  - Used to secure HTTP interactions for use in Internet e-commerce and other security sensitive applications.

SSL/TLS

- Design requirements
  - Secure communication without prior negotiation or help from 3rd parties
  - Free choice of crypto algorithms by client and server
  - Communication in each direction can be authenticated, encrypted or both

- Main features:
  - Negotiable encryption and authentication algorithms:
    - algorithms negotiated during the initial handshake.
  - Bootstrapped secure communication:
    - Unencrypted communication is used for the initial exchanges, then public key-key cryptography and finally switching to secret-key cryptography once a shared key has been established.

- Protocol prefix *https*:
  - in URLs initiates the establishment of an TLS secure channel between a browser and a web server.

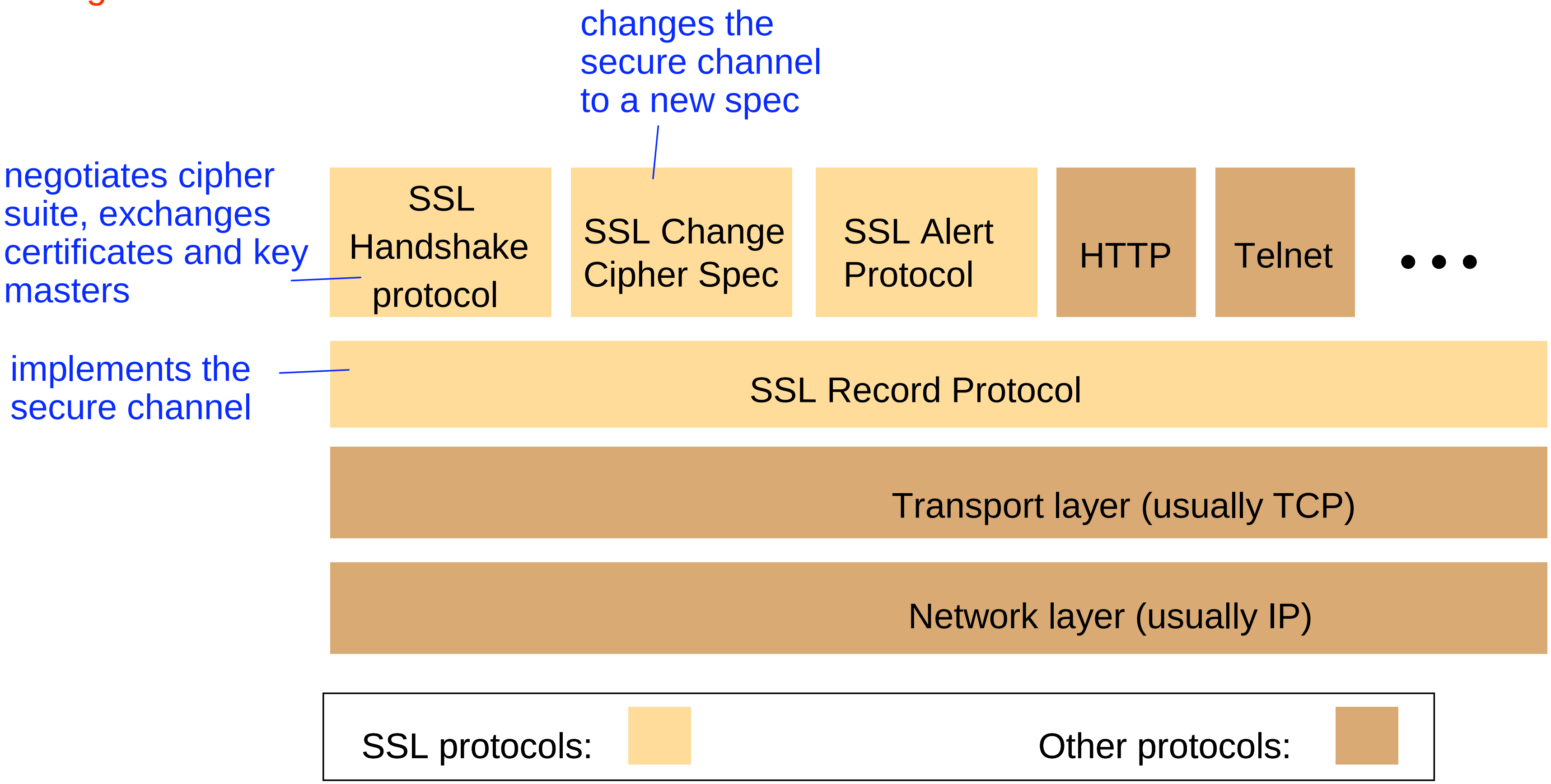
SSL Layers

- Two layers:
  - *SSL Record Protocol layer* - which implements a secure channel, encrypting and authenticating messages transmitted through any connection-oriented protocol

  - *handshake layer* – containing the SSL handshake protocol and two other related protocols that establish and maintain an SSL session (that is, a secure channel).

SSL protocol stack

Figure 7.17



changes the secure channel to a new spec

negotiates cipher suite, exchanges certificates and key masters

implements the secure channel

SSL Handshake protocol

SSL Change Cipher Spec

SSL Alert Protocol

HTTP

Telnet

• • •

SSL Record Protocol

Transport layer (usually TCP)

Network layer (usually IP)

SSL protocols: □

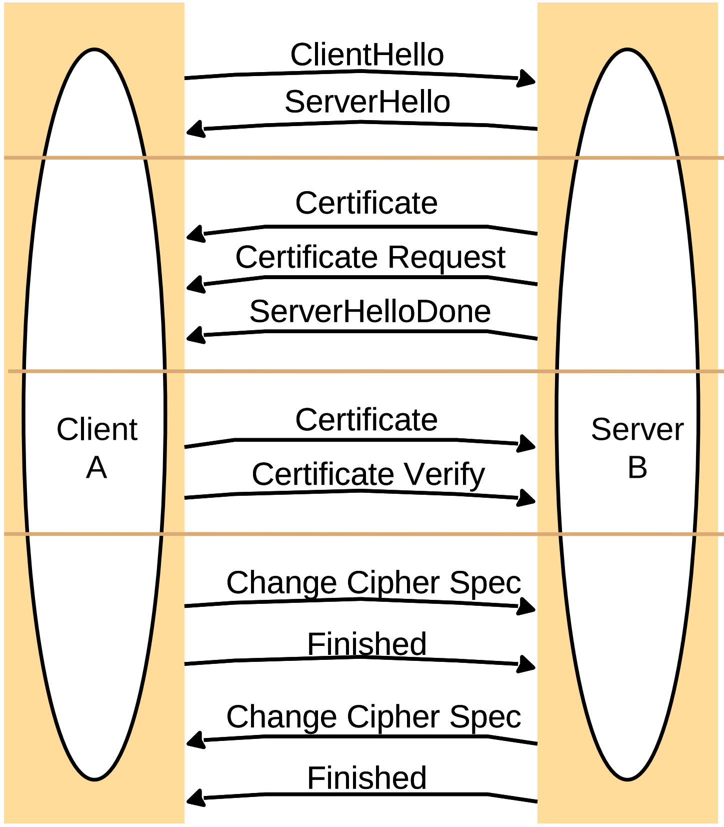
Other protocols: ■

SSL handshake protocol

- Performed over an existing connection.
- Initial handshake
  - The public key used to verify the first certificate received may be delivered by a separate channel
- The partners optionally authenticate each other by exchanging signed public key certificate in X.509 format
- A *pre-master secret* is sent to the other partner encrypted with the public key. A pre-master secret is a large random value that is used by both partners to generate the two session keys (called *write* keys)
- *Session keys* used for encrypting data in each direction and the message authentication secrets to be used for message authentication.
- Secure session can start: triggered by the *ChangeCipherSpec* messages, followed *Finished* messages.
- Once the Finished messages have been exchanged, all further communication is encrypted and signed.

SSL handshake protocol

Figure 7.18



Establish protocol version, session ID, cipher suite, compression method, exchange random start values

Optionally send server certificate and request client certificate

Send client certificate response if requested

Change cipher suite and finish handshake

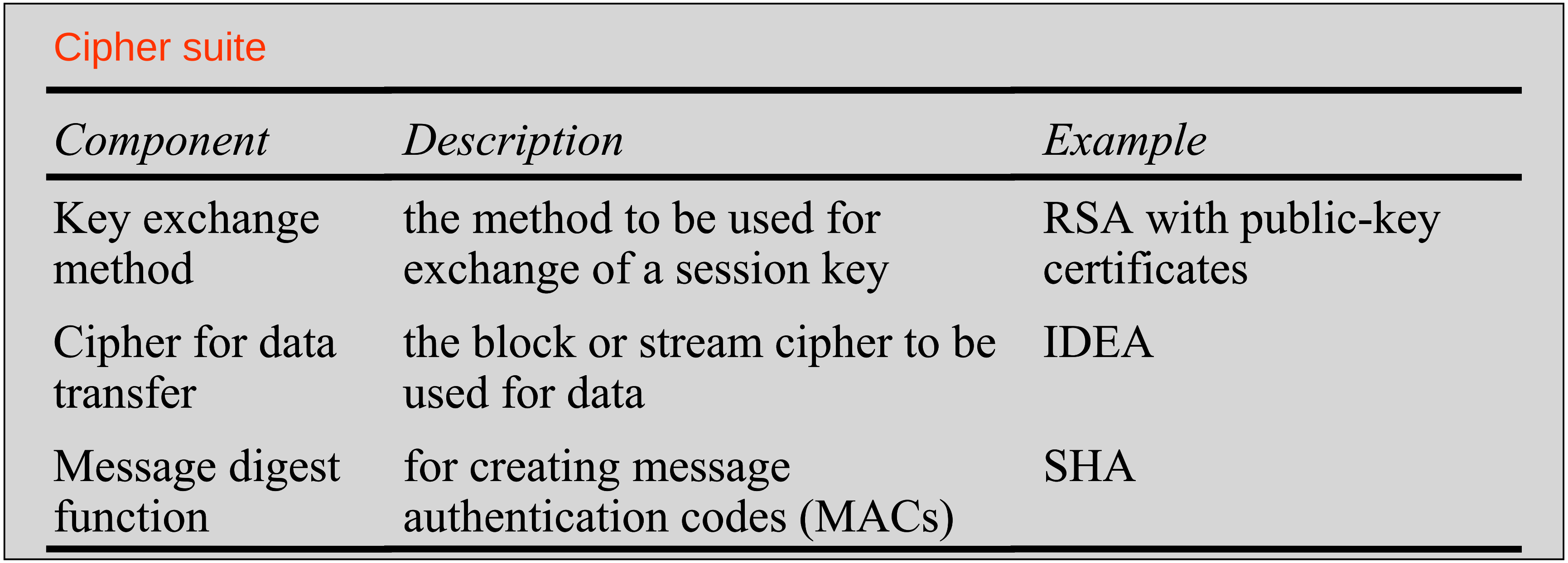
Includes key master exchange.
Key master is used by both A and B to generate:

2 session keys *2 MAC keys* □

K_{AB} M_{AB}
K_{BA} M_{BA}

SSL handshake configuration options

- SSL supports a variety of options for the cryptographic functions to be used – *cipher suites*.
- A cipher suite includes a single choice for each of the features shown:



Cipher suite

Component	*Description*	*Example*
Key exchange method	the method to be used for exchange of a session key	RSA with public-key certificates
Cipher for data transfer	the block or stream cipher to be used for data	IDEA
Message digest function	for creating message authentication codes (MACs)	SHA

SSL record protocol

- A message for transmission is first fragmented into blocks of manageable size.
- Then the blocks are optionally compressed.
- The encryption and message authentication (MAC) transformations deploy the algorithms specified in the agreed cipher suite.
- The signed and encrypted block is transmitted to the partner through associated TCP connection, where the transformations are reversed to produce the original data block.

SSL record protocol

Figure 7.20

Application data

Fragment/combine

Record protocol units

Compress

Compressed units

Hash

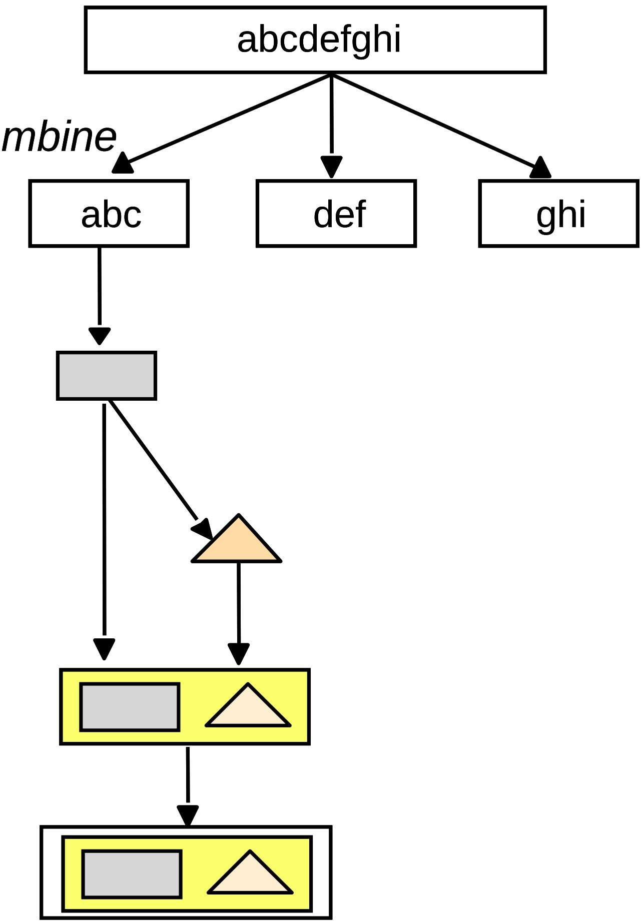
MAC

Encrypt

Encrypted

Transmit

TCP packet



SSL

- A practical implementation of a *hybrid* encryption scheme with authentication and key exchange based on public keys.

- Because the ciphers are negotiated in the handshake, it does not depend upon the availability of any particular algorithms, nor any secure services at the time of session establishment.

- The only requirement is for public-key certificates issued by an authority that is recognized by both parties.

SECURE FILE EXCHANGE IN JAVA

JAVA API: Public Key Infrastructure

- The Java platform includes APIs and provider support for X.509 digital certificates and Certificate Revocation Lists (CRLs

- The classes related to PKI are located in the `java.security` and `java.security.cert` packages.

Secure File Exchange
Code and Document Security

- Digital documents are easy to generate, copy and alter.

- If you electronically send someone an important document (or documents), or an applet or application to run, the recipient needs a way to verify that the document or code came from you and was not modified in transit (for example, by a malicious user intercepting it).

- Digital signatures, certificates, and `keystores` all help ensure the security of the files you send.

Secure Code & File Exchange

The basic idea in the use of digital signatures is as follows.

1. You "sign" the document or code using one of your *private keys*, which you can generate by using `keytool` or security API methods.
   - That is, you generate a digital signature for the document or code, using the `jarsigner` tool or API methods.

2. You send to the other person, the "receiver," the document or code and the signature.

3. You also supply the receiver with the public key corresponding to the private key used to generate the signature, if the receiver doesn't already have it.

4. The receiver uses the *public key* to verify the authenticity of the signature and the integrity of the document/code.

A receiver needs to ensure that the public key itself is authentic before reliably using it to check the signature's authenticity.

Therefore it is more typical to supply a *certificate* containing the public key rather than just the public key itself.

Secure File Exchange
Digital Certificates

A certificate contains:

1. A public key.

2. The "distinguished-name" information of the entity (i.e., the certificate subject or owner).

   Includes attributes such as: the entity's name, organisational unit, organization, city or locality, state or province, and country code.

3. A digital signature.

   A certificate is signed by one entity, the issuer, to vouch for the fact that the enclosed public key is the actual public key of another entity, the owner.

4.The distinguished-name information for the signer (issuer).

Secure File Exchange Digital Certificates … cont’d

Validating Certificates

1. A recipient check if a certificate is valid by verifying its digital signature, using the issuer's (signer's) public key.

2. That key can be stored within another certificate whose signature can also be verified by using the public key of that next certificate's issuer, and that key may also be stored in yet another certificate, and so on.
   - You can stop checking when you reach a public key that you already trust and use it to verify the signature on the corresponding certificate.

3. If a recipient cannot establish a trust chain, then he/she can calculate the certificate fingerprint(s), using the `keytool -import or -printcert` command.
   - A fingerprint is a relatively short number (hash value of the certificate information) that uniquely and reliably identifies the certificate.
   - If the fingerprints are the same, the certificates are the same.

Signing Code and Granting Permissions

- In this scenario, Susan wishes to send code to Ray.
- Ray wants to ensure that when the code is received, it has not been tampered with along the way - for instance someone could have intercepted the code exchange e-mail and replaced the code with a virus.

Signing Code and Granting Permissions –Sending side

Susan:

1. Creates an application
2. Create a JAR File containing the class file, using the `jar` tool.
3. Generate keys (if they don't already exist), using the

   `keytool -genkey`

   command.
4. Sign the JAR file, using the `jarsigner` tool and the private key.
5. Export the public key certificate, using the

   `keytool -export`

   command. Then supply the signed JAR file and the certificate to the receiver Ray.

Signing Code and Granting Permissions - Receiving side

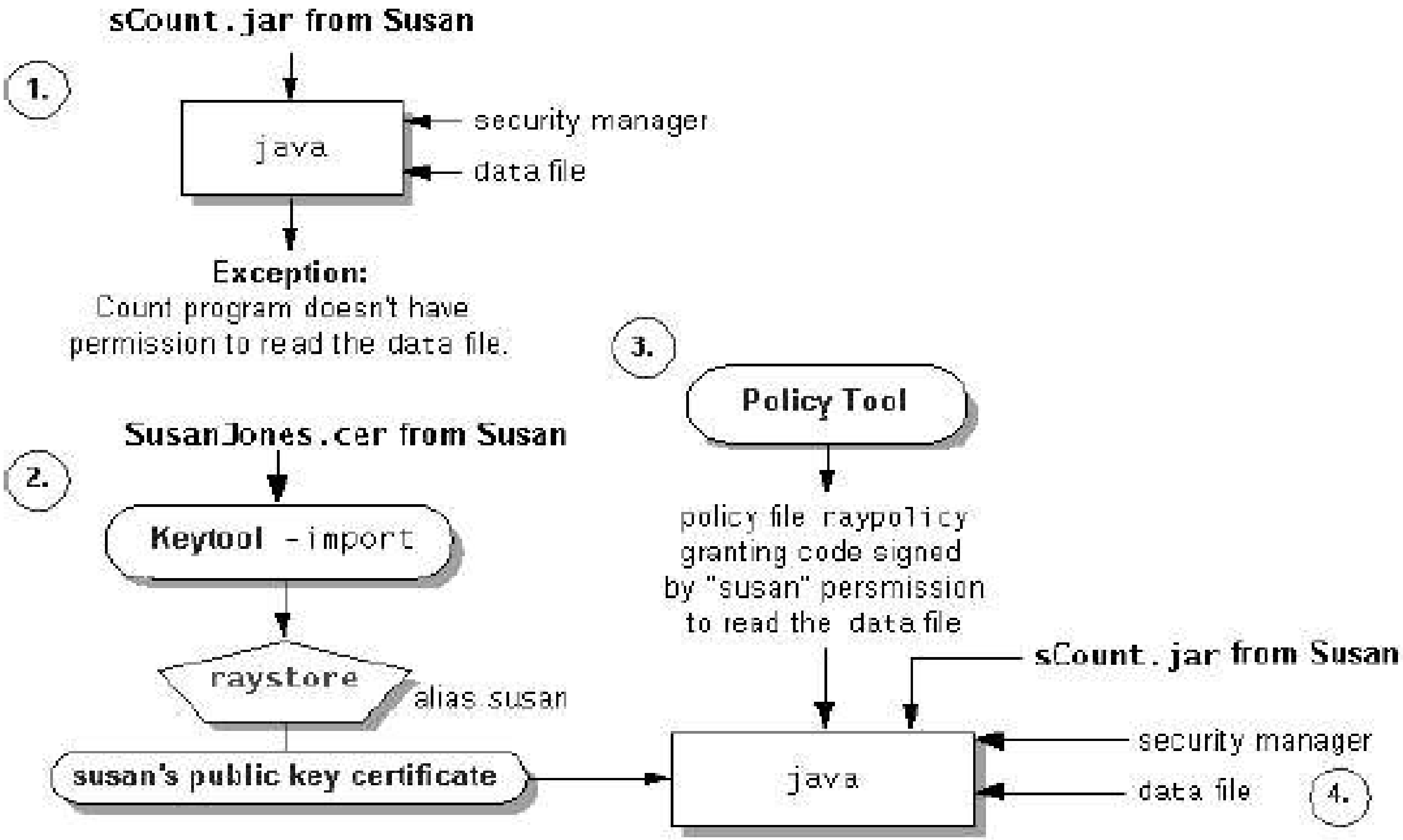
Ray:

1. Tries to run the jar code received from Susan and gets:

   ```
   Exception in thread "main"
   java.security.AccessControlException:
   ```

2. Import the certificate as a trusted certificate, using the `keytool -import` command, and give it the alias susan.
3. Set up a policy file to grant the required permission to permit classes signed by susan to read the specified file.
4. See the policy file effects, that is, see how the application can now read the file.

Signing Code and Granting Permissions - Receiving side



References

- Java Security Tutorials
  - Security Architecture
    https://docs.oracle.com/javase/8/docs/technotes/guides/security/
  - Security Features in Java SE
    https://docs.oracle.com/javase/tutorial/security/toolsign/index.html

- Chapter 11 - Security: Coulouris, Dollimore and Kindberg, Distributed Systems: Concepts and Design