Java™ Security

Sang Shin
sang.shin@sun.com
Java™ Technology Evangelist
Sun Microsystems, Inc.

Disclaimer & Acknowledgment

Even though Sang Shin is a full-time employee of Sun Microsystems, the contents here are created as his own personal endeavor and thus does not reflect any official stance of Sun Microsystems.

Sun Microsystems is not responsible for any inaccuracies in the contents.

Acknowledgment
- Some slides are borrowed from Raghaven Srinivas of Sun Microsystems

Agenda
- Java Security Overview
- Message Digest
- Java CertPath
- JSSE
- JAAS
- JCE
- Kerberos
Java Security Overview

Java™ Technology-based Security ("Java Security")
- Java™-based software runs as designed
  - Adheres to the Java language specification and the JVM™ specification
  - Provides building blocks for secure applications
  - Resists attacks on the language and platform
  - Reduces the chance and impact of accidental programming errors

Java Security Architecture

Java Security Evolution
- Initial releases of JDK focussed on executable content threats
- Optional Packages and upcoming releases focus on distributed security threats
Java 2 Platform Security Goals

- Treating applet and application security in a consistent manner
- Fine-grained access control through policy file
- Well-defined Access Control Mechanism
- Concrete SecurityManager class

Java Platform Security Overview

| J2SE | JCA/JCE | Crypto APIs
| JSSE | J2SE | SSL/TLS APIs
| java CertPath | J2EE | Cert. Chain Building/validation Framework for SSO
| JAAS/ JGSS | J2E Security Sec. Interop. Bean/Container | CSIv2 security interoperability Container based security
| | J2ME | MIDP 2.0 https support, “sandbox” model
| | Java Card | Authentication and Crypto APIs

J2SE® Platform Security: Big Picture

JCA/ JCE Provider Architecture
**JCA/ JCE**

**JCA**
- Cryptographic Architecture
- Basic cryptography
- Export control free
- Signatures, Digests, etc.

**JCE**
- Cryptographic Extensions
- Advanced cryptography
- Export control restrictions (originally, not anymore)
- Ciphers

---

**Message Digest**

---

**Message Digest Example**

```java
01 import java.security.*;
02 03 // Use the MD5 Algorithm
04 MessageDigest md=
05     MessageDigest.getInstance("MD5");
06 byte buf[] = Message.getBytes();
07 08 // Update the data
09 md.update(buf);
10 // After input is ready, digest the data
11 byte digestBuf[] = md.digest();
```

---

**Message Digest Values**

<table>
<thead>
<tr>
<th>Message</th>
<th>MD5 Digest</th>
</tr>
</thead>
<tbody>
<tr>
<td>I pay Bill $1500.00</td>
<td>8021d0cda6ca230a5853f9d55ba4cecb</td>
</tr>
<tr>
<td>I pay Bill $15000.00</td>
<td>C0239506c9c9c68adcb62bd4a5c5</td>
</tr>
<tr>
<td>I pay Jill $15000.00</td>
<td>752ed078a31567bf81545d21da298be9</td>
</tr>
</tbody>
</table>
Java CertPath

Java CertPath Programming Model

```java
import java.security.*;
import java.security.cert.*;

// CertificateFactory for X.509
CertificateFactory cf = CertificateFactory.getInstance("X.509");

// Obtain CertPathValidator
CertPathValidator cpv = CertPathValidator.getInstance("PKIX");

// Set the Trust anchor
TrustAnchor anchor = new TrustAnchor((X509Certificate)tks.getCertificate("ca").null();

// Set the PKIX parameters
PKIXParameters params = new PKIXParameters( Collections.singleton(anchor));

// Revocation as false
params.setRevocationEnabled(false);

// Validate
PKIXCertPathValidatorResult result = (PKIXCertPathValidatorResult) cpv.validate(cp, params);
```

Java CertPath Model (Cont.)

```java
// Set the PKIX parameters
PKIXParameters params = new PKIXParameters(
    Collections.singletonList(anchor));

// Revocation as false
params.setRevocationEnabled(false);

// Validate
PKIXCertPathValidatorResult result = (PKIXCertPathValidatorResult) cpv.validate(cp, params);
```

Java Platform Security Extensions
Java™ Platform Security Extensions (optional packages)

- Java Secure Socket Extension (JSSE)
- Java Authorization and Authentication Service API (JAAS)
- Java Cryptography Extensions (JCE)
  - Common API for applications
  - Standard SPI for security service providers

JSSE

What is JSSE?

- Java API for Secure Sockets Layer (SSL)
- SSL provides security at Session level
  - Confidentiality (Privacy)
  - Data integrity (Tamper-proofing)
  - Server authentication (Proving a server is what it claims it is)
  - Optional client authentication
- Uses algorithms, keys transparently
Secure Socket Layer (SSL)

- By far, the dominant security protocol on the web
  - HTTPS is HTTP over SSL
- Responsible for the emergence of e-commerce, other security sensitive services on the web
- Beneficiary of several years of public scrutiny

SSL Overview

- Operates atop bi-directional, reliable byte stream. Typically TCP
- Offers end-to-end security even when the underlying reliable byte stream is proxied

SSL's Layered Architecture

- Record layer offers bulk encryption/authentication using symmetric-key algorithms
- Cleartext flow until symmetric key is established
- Handshake protocol uses public-key algorithms to establish a "master-secret" used to derive MAC secrets, cipher keys/IVs

JSSE Programming: Server Side

```java
01 import java.io.*;
02 import java.net.*;
03 import javax.net.ssl.*;
04
05 // Create server side SSL socket
06 SSLServerSocketFactory sslsrvfact =
07     SSLServerSocketFactory.getDefault();
08 SSLServerSocket s =
09     sslsrvfact.createServerSocket(port);
10 s.accept();
```
JSSE Programming: Client Side

```java
01 import java.io.*;
02 import java.net.*;
03 import javax.net.ssl.*;
04
05 // Create Client SSL socket
06 SSLSocketFactory ssfact =
07     SSLSocketFactory.getDefault();
08 SSLSocket s =
09     ssfact.createSocket(host, port);
```

JaSA

Java™ Authentication and Authorization Service (JAAS) API

- Java platform security are based on (without JAAS)
  - Where the code originated
  - Who signed the code
- The JAAS API augments this with
  - who’s running the code
- Pluggable authentication
- User-based authentication/ authorization

JAAS Pluggable Authentication
JAAS File Entries

```java
// Example Java 2 Security Policy Entry
grant Codebase "www.sun.com", Signedby "duke" {
    FilePermission "/cdrom/-", "read";
}
```

```java
// Example JAAS Security Policy Entry
grant Codebase "www.sun.com", Signedby "duke",
Principal com.sun.Principal "charlie" {
    FilePermission "/cdrom/charlie/-", "read";
}
```

```java
// Example login module configuration entry
Login2 {
    sample.SampleLoginModule required;
    com.sun.security.auth.module.NTLoginModule sufficient;
    com.foo.SmartCard requisite debug=true;
    com.foo.Kerberos optional debug=true;
}
```

JAAS Programming

```java
import java.security.*;
import javax.security.auth.*; //exts

// Instantiate a login context
LoginContext ctx = new LoginContext
        ("name", CallbackHandler);

// Authenicate the subject
ctx.login();

// Retrieve authenticated subject
Subject sub = ctx.getSubject();

// Enforce Access Controls
Subject.doAs(sub, action);
```

JAAS Programming Model

```java
import java.security.*;
import javax.security.auth.*; //exts

// Instantiate a login context
LoginContext ctx = new LoginContext
        ("name", CallbackHandler);

// Authenicate the subject
ctx.login();

// Retrieve authenticated subject
Subject sub = ctx.getSubject();

// Enforce Access Controls
Subject.doAs(sub, action);
```

JCE
Java™ Cryptography Extensions (JCE)

- Cryptographic APIs supplementing the Java 2 platform
- Framework for multiple CSPs (Cryptographic Service Providers)
  - Comes with Sun JCE provider
  - Multiple independent providers

Cryptographic Process

- M is the original message
- Kenc is encryption key
- M' is the scrambled message
- Kdec is decryption key
- From M' only, "hard" to get M
- E and D are related such that
  \[ E(K_{enc}, M) = M' \]
  \[ D(K_{dec}, M') = M \]
  \[ D(K_{dec}, E(K_{enc}, M)) = M \]

Crypto. Processes (compared)

- Public Key Cryptography
  - Encryption and decryption keys are different
  - Key distribution is easier
  - Public key cryptography is very slow
  - Examples: RSA

- Private Key Cryptography
  - Encryption and decryption keys are same
  - Key distribution is an issue
  - Private key cryptography is faster
  - Examples: DES, AES

- Session Key Cryptography
  - Key negotiation and encryption separate
  - Key distribution is not an issue
  - Best of both approaches
  - Examples: SSL

JCE 1.2 Release

- JCE 1.2 [invoc. crypto]
  - Cipher
  - KeyAgreement
  - KeyExchange
  - Signature/Wrappery
  - MAC
JCE Programming model

```java
01 import java.security.*;
02 import javax.crypto.*;
03
04 // Get Provider
05 Provider sunJce = new
06     com.sun.crypto.provider.SunJCE();
07
08 // Obtain Cipher
09 Cipher c = Cipher.getInstance
10     ("Blowfish");
11
12 // Get the key Generator
13 KeyGenerator kgen =
14     KeyGenerator.getInstance("Blowfish");
15
16 // Generate the key specs
17 byte[] raw = skey.getEncoded();
18 SecretKeySpec kspec = new
19     SecretKeySpec(raw, "Blowfish");
20
21 // Initialize cipher with keys, etc.
22 cipher.init(Cipher.ENCRYPT_MODE, kspec);
23
24 // Update buffers
25 while (msg[i] != null)
26   enc = cipher.update(msg[i].getBytes());
27
28 // Finish up
29 enc = cipher.doFinal();
```

Kerberos

Single Sign-On Using Kerberos
Summary & Roadmap

J2SE Security Summary

- J2SE Security APIs are simple to use and has a provider architecture
- J2SE Security APIs are part of the Java 2 SDK instead of being optional packages
- The Java 2 SDK supports simple security tools for code signing

Resources

- Java Security
  http://java.sun.com/security
- J2EE Security
  http://java.sun.com/j2ee
  http://java.sun.com/j2ee/tutorial
- J2ME Security
  http://java.sun.com/j2me
Resources

- Java Card Security
  http://java.sun.com/javacard
- Sun ONE Products (incl. Security)
- Solaris Security
  http://www.sun.com/security
- Solaris Security Blueprints