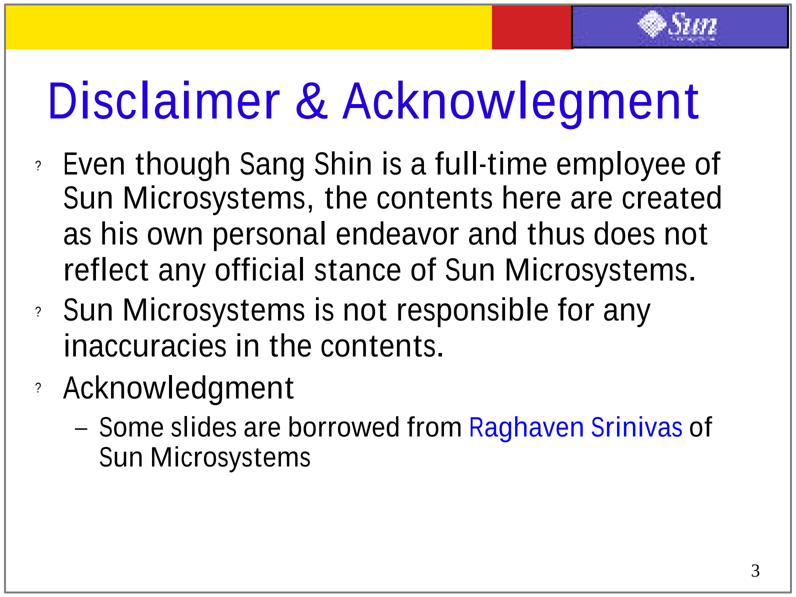
Java™ Security

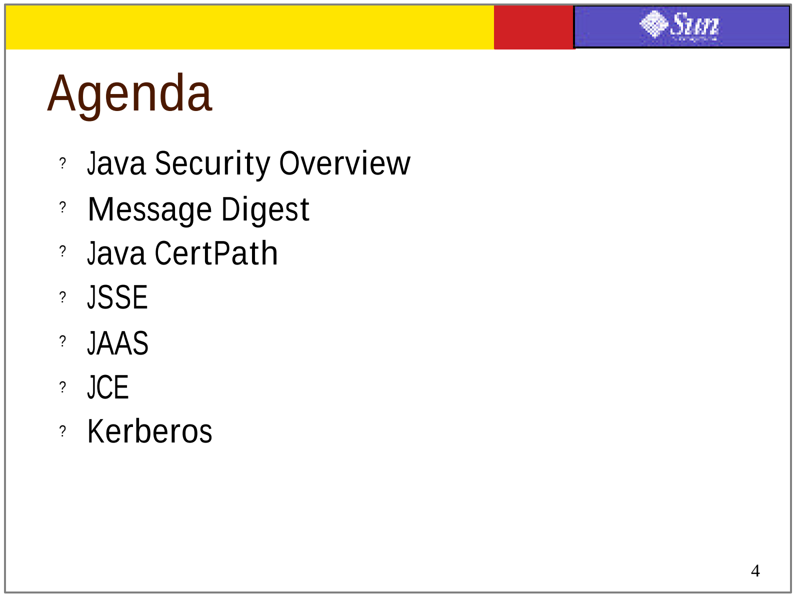
Sang Shin

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



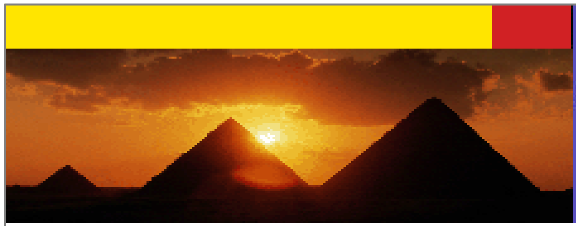
Disclaimer & Acknowlegment

- 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

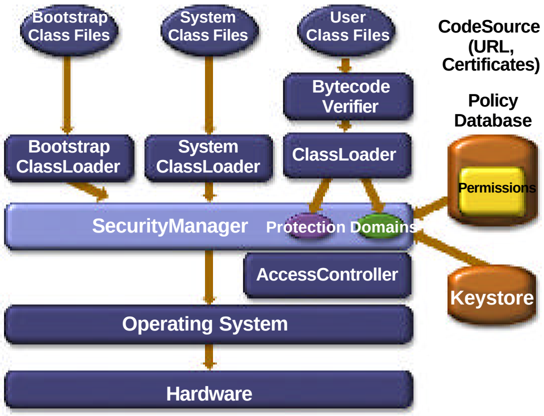


Java Security Overview

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



Bootstrap Class Files

System Class Files

User Class Files

Bytecode Verifier

Bootstrap ClassLoader

System ClassLoader

ClassLoader

SecurityManager

Protection Domains

AccessController

Operating System

Hardware

CodeSource (URL, Certificates)

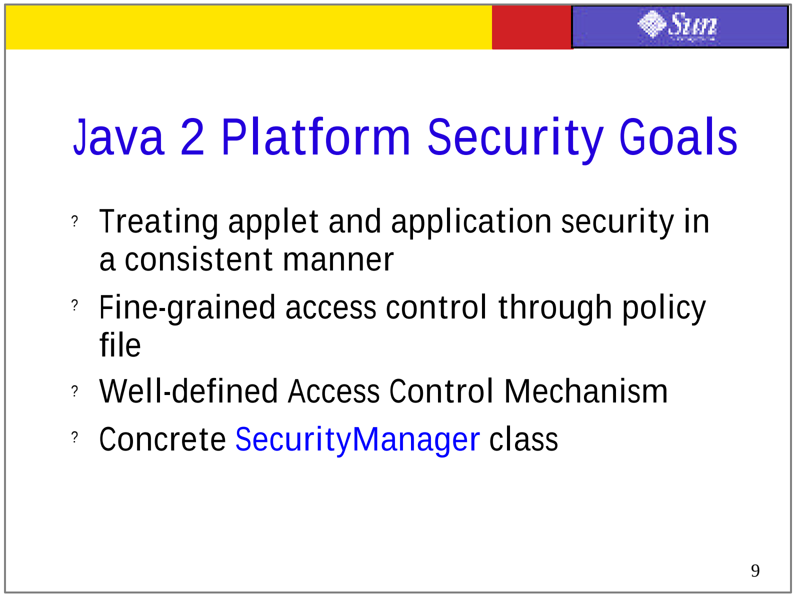
Policy Database

Permissions

Keystore

Java Security Evolution

- Initial releases of JDK focussed on executable content threats
- Optional Packages and upcoming releases focus on distributed security threats



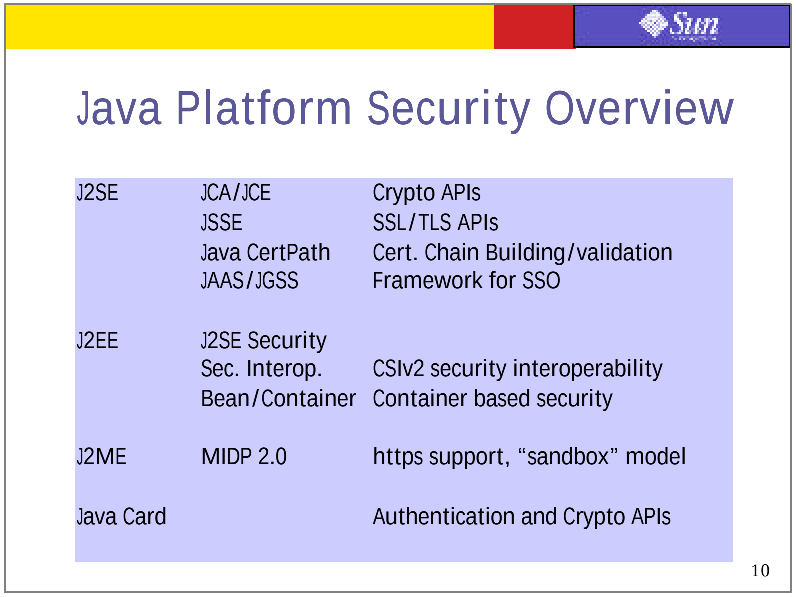
Java 2 Platform Security Goals

? Treating applet and application security in a consistent manner

? Fine-grained access control through policy file

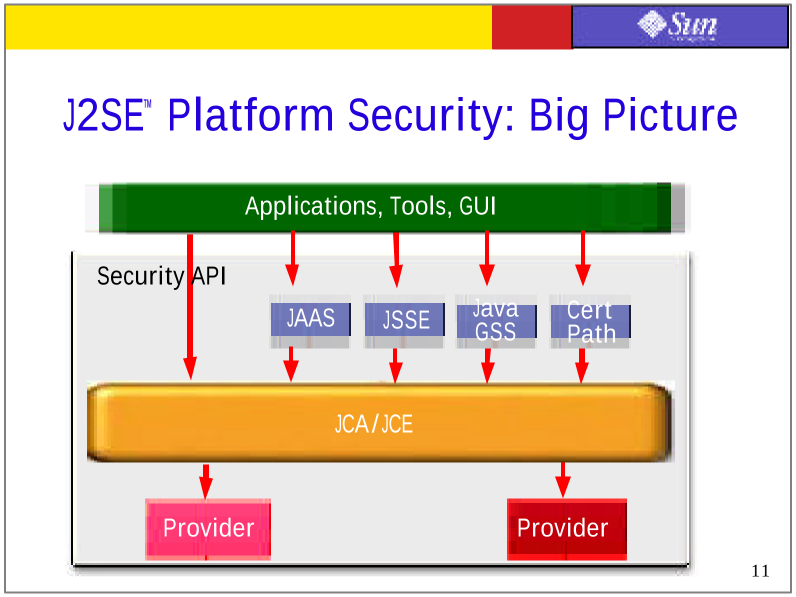
? Well-defined Access Control Mechanism

? Concrete SecurityManager class



Java Platform Security Overview

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



J2SE™ Platform Security: Big Picture

Applications, Tools, GUI

Security API

JAAS

JSSE

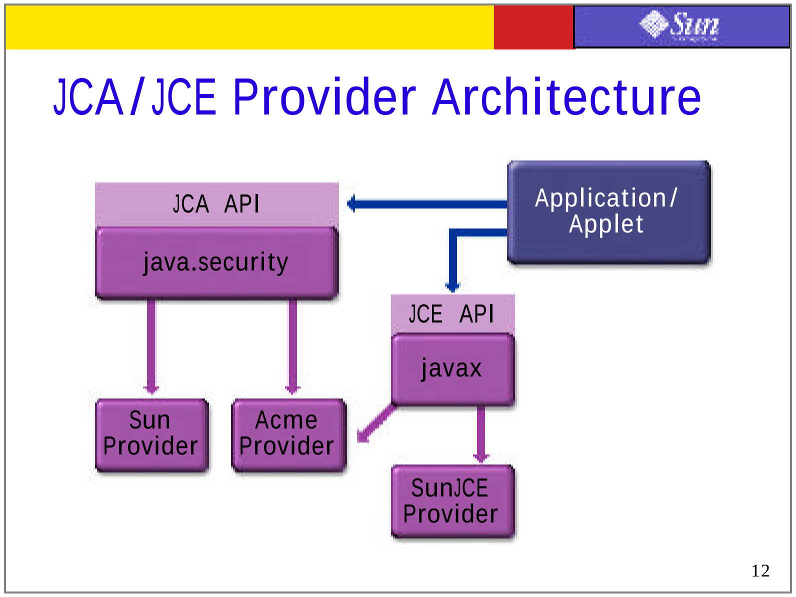
Java GSS

Cert Path

JCA/JCE

Provider

Provider



JCA/JCE Provider Architecture

JCA API

java.security

Application/ Applet

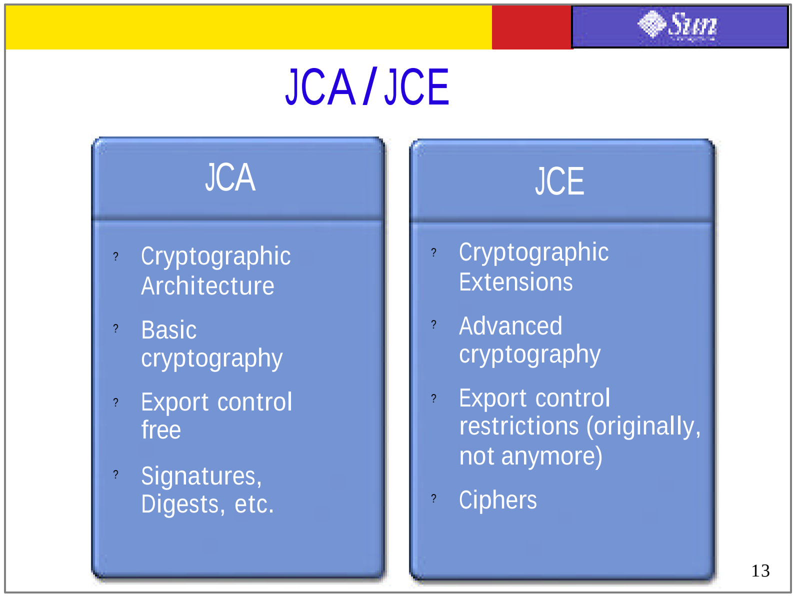
JCE API

javax

Sun Provider

Acme Provider

SunJCE Provider



JCA / JCE

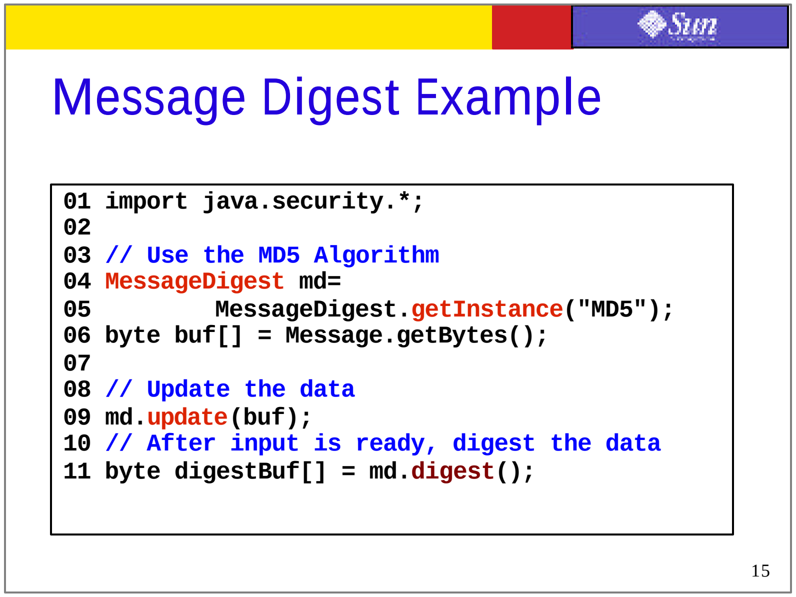
JCA

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

JCE

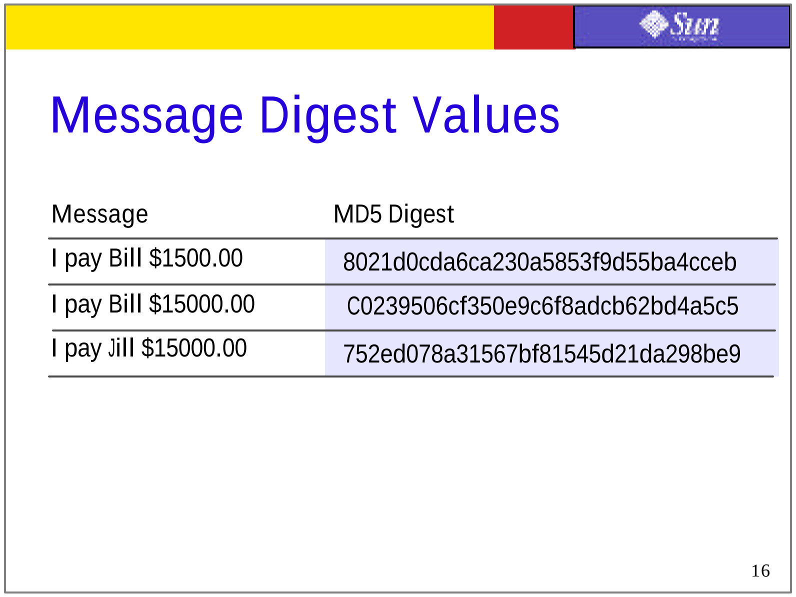
- Cryptographic Extensions
- Advanced cryptography
- Export control restrictions (originally, not anymore)
- Ciphers

Message Digest



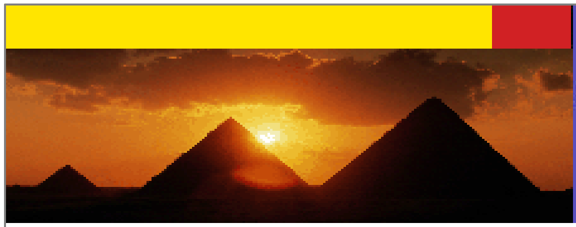
Message Digest Example

```
import java.security.*;

// Use the MD5 Algorithm
MessageDigest md=
        MessageDigest.getInstance("MD5");
byte buf[] = Message.getBytes();

// Update the data
md.update(buf);
// After input is ready, digest the data
byte digestBuf[] = md.digest();
```



Message Digest Values

Message	MD5 Digest
I pay Bill $1500.00	8021d0cda6ca230a5853f9d55ba4cceb
I pay Bill $15000.00	C0239506cf350e9c6f8adcb62bd4a5c5
I pay Jill $15000.00	752ed078a31567bf81545d21da298be9



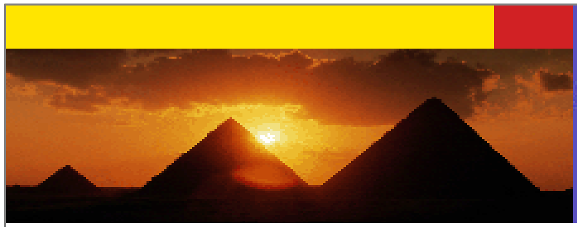
Java CertPath

Java CertPath Programming Model

```
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);
```

Java CertPath Model (Cont.)

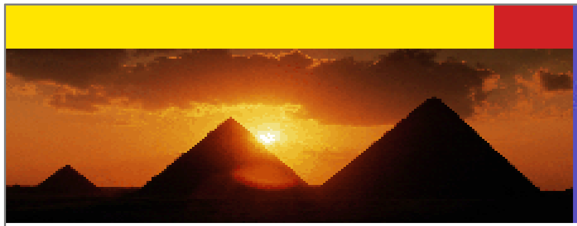
```
// Set the PKIX parameters
PKIXParameters params = new
   PKIXParameters(
      Collections.singleton(anchor));
// Revocation as false
params.setRevocationEnabled(false);

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



Java 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

Java™ Secure Socket Extension (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

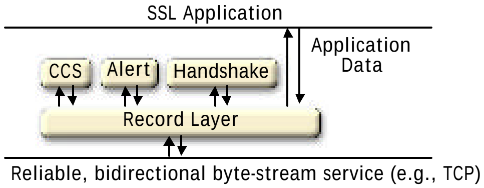
Secure Socket Layer (SSL)

- 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



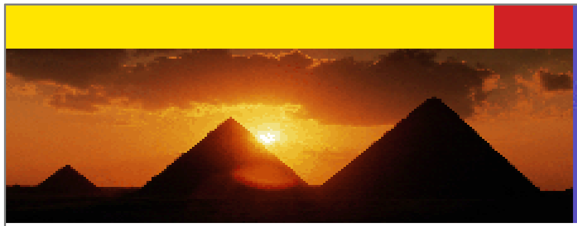
- Record layer offers bulk encryption/authentication using symmetric-key algorithms
- Cleartext flow until symmetric key is established
- Handshake protocol uses public-key algorithms to establish a "master-secret" used to derive MAC secrets, cipher keys/IVs

JSSE Programming: Server Side

```
import java.io.*;
import java.net.*;
import javax.net.ssl.*;

// Create server side SSL socket
SSLServerSocketFactory sslsrvfact =
    SSLServerSocketFactory.getDefault();
SSLServerSocket s =
    sslsrvfact.createServerSocket(port);
s.accept();
```

JSSE Programming: Client Side

```
import java.io.*;
import java.net.*;
import javax.net.ssl.*;

// Create Client SSL socket
SSLSocketFactory sslfact =
    SSLSocketFactory.getDefault();
SSLSocket s =
    sslfact.createSocket(host, port);
```

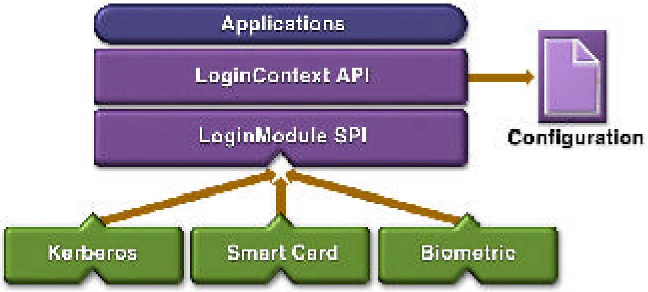


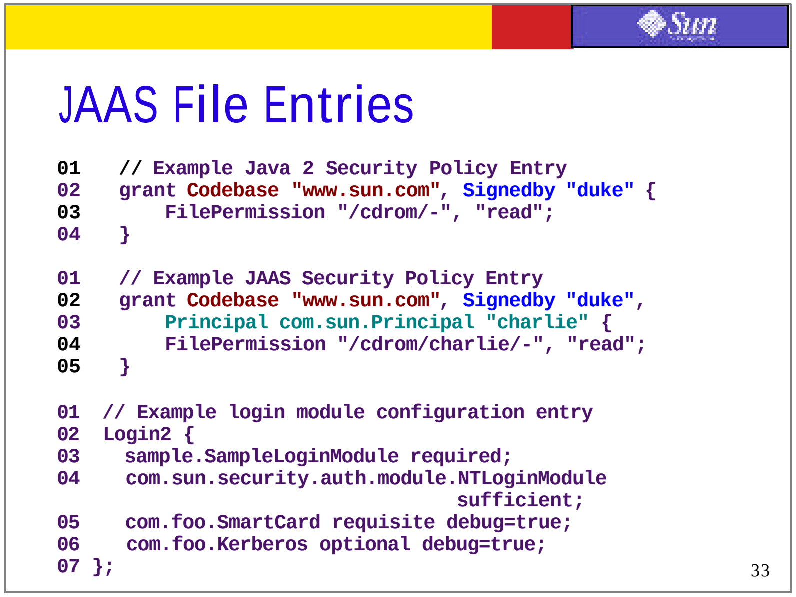
JAAS

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

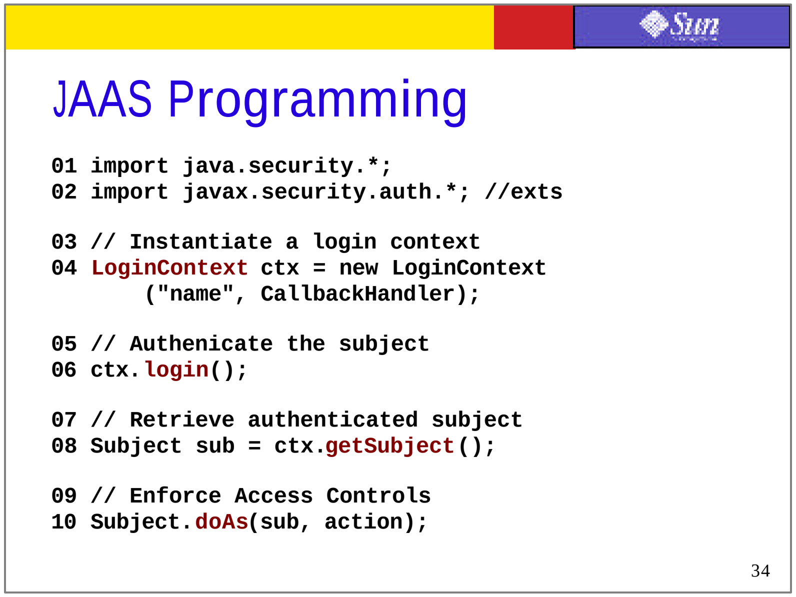
JAAS Pluggable Authentication





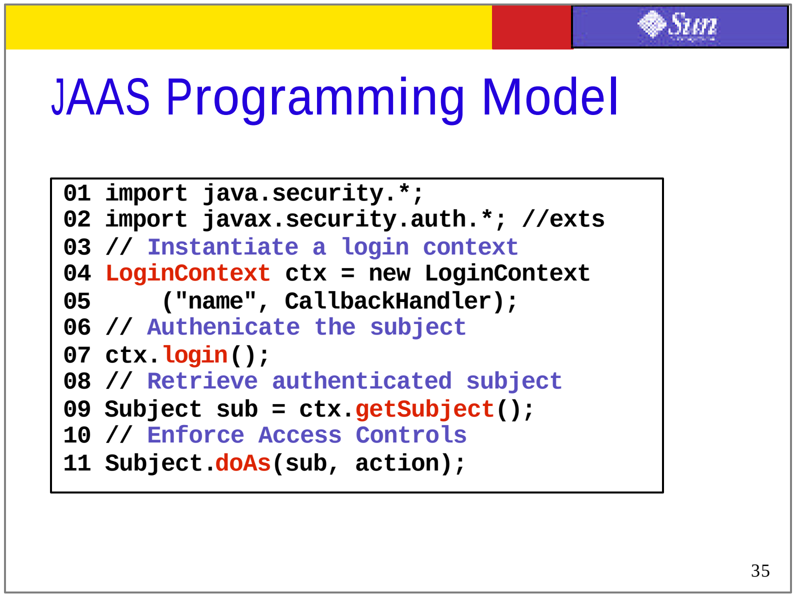
JAAS File Entries

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

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

 // 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

```
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

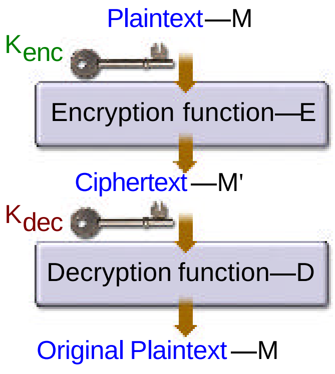
```
import java.security.*;
import javax.security.auth.*; //exts
// Instantiate a login context
LoginContext ctx = new LoginContext
    ("name", CallbackHandler);
// Authenicate the subject
ctx.login();
// Retrieve authenticated subject
Subject sub = ctx.getSubject();
// Enforce Access Controls
Subject.doAs(sub, 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



Plaintext—M

K_{enc}

Encryption function—E

Ciphertext—M'

K_{dec}

Decryption function—D

Original Plaintext—M

M is the original message
K_{enc} is encryption key
M' is the scrambled message
K_{dec} 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

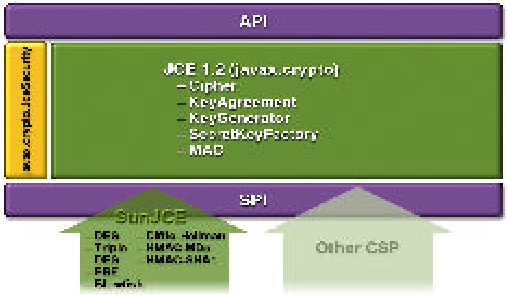
Private Key Cryptography

- 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 seperate
- Key distribution is not an issue
- Best of both approaches
- Examples: SSL

JCE 1.2 Release



API

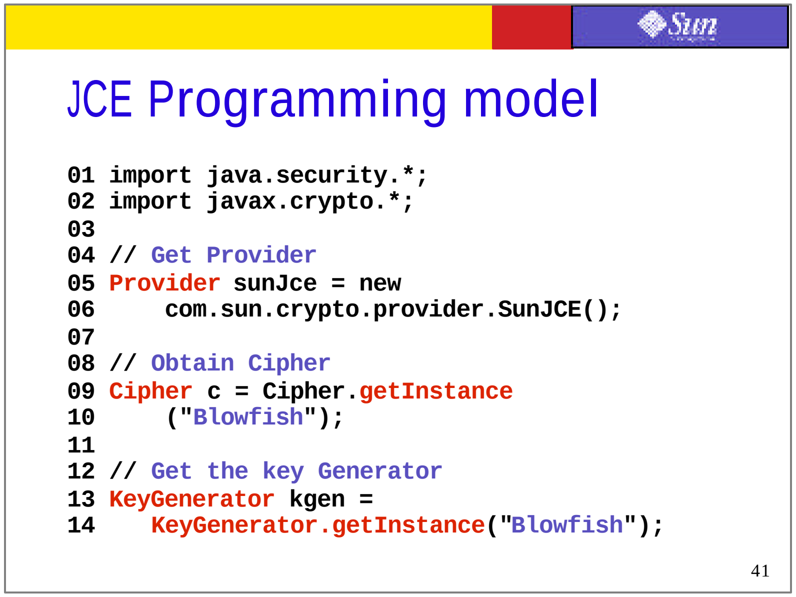
javax.crypto.JceSecurity

JCE 1.2 (javax.crypto)
– Cipher
– KeyAgreement
– KeyGenerator
– SecretKeyFactory
– MAC

SPI

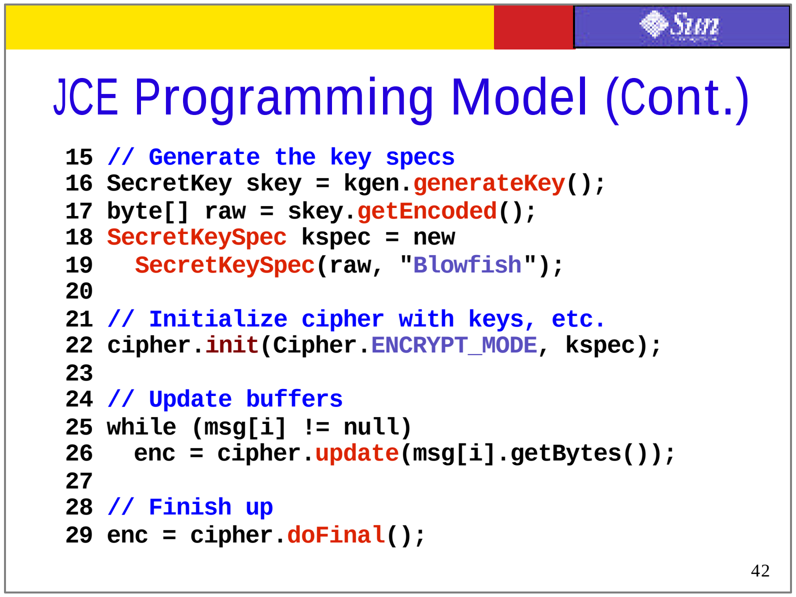
SunJCE
DES
Triple DES
PBE
Blowfish
Diffie-Hellman
HMAC-MD5
HMAC-SHA1

Other CSP



JCE Programming model

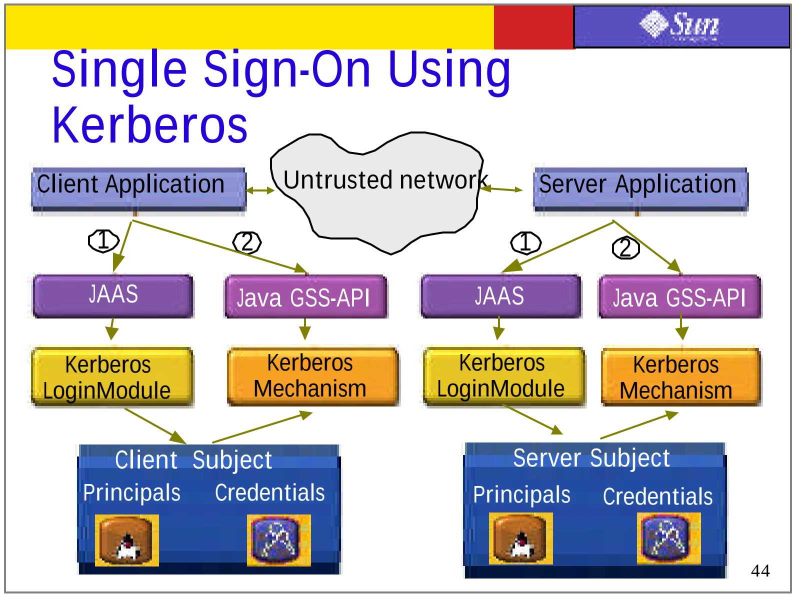
```
import java.security.*;
import javax.crypto.*;

// Get Provider
Provider sunJce = new
    com.sun.crypto.provider.SunJCE();

// Obtain Cipher
Cipher c = Cipher.getInstance
    ("Blowfish");

// Get the key Generator
KeyGenerator kgen =
   KeyGenerator.getInstance("Blowfish");
```



JCE Programming Model (Cont.)

```
// Generate the key specs
SecretKey skey = kgen.generateKey();
byte[] raw = skey.getEncoded();
SecretKeySpec kspec = new
  SecretKeySpec(raw, "Blowfish");

// Initialize cipher with keys, etc.
cipher.init(Cipher.ENCRYPT_MODE, kspec);

// Update buffers
while (msg[i] != null)
  enc = cipher.update(msg[i].getBytes());

// Finish up
enc = cipher.doFinal();
```

Kerberos



Single Sign-On Using Kerberos

Client Application

Untrusted network

Server Application

1

2

JAAS

Java GSS-API

Kerberos LoginModule

Kerberos Mechanism

Client Subject

Principals

Credentials

1

2

JAAS

Java GSS-API

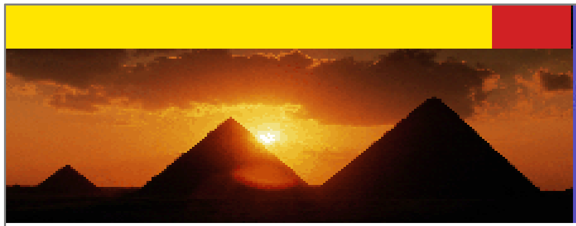
Kerberos LoginModule

Kerberos Mechanism

Server Subject

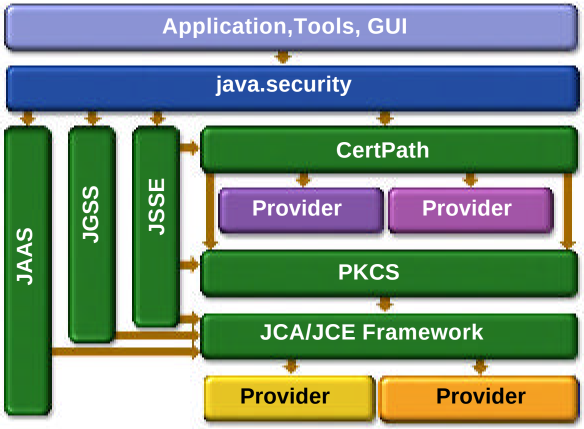
Principals

Credentials



Summary & Roadmap

Roadmap: J2SE Security Big Picture



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

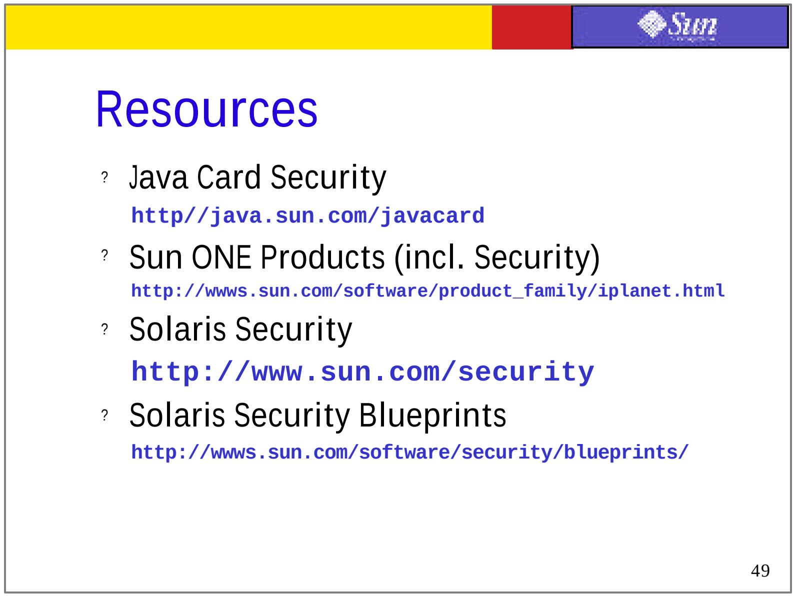
? The Java 2 SDK supports simple security tools for code signing

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)

http://wwws.sun.com/software/product_family/iplanet.html

? Solaris Security

http://www.sun.com/security

? Solaris Security Blueprints

http://wwws.sun.com/software/security/blueprints/