# Techniques and Tips: Developing Secure Payment Applications, Using Java™ ME Technology

**Angela Caicedo**
**Doris Chen Ph.D.**

Technology Evangelists
Sun Microsystems

TS-1049

Copyright © 2006, Sun Microsystems, Inc., All rights reserved.

# Goal of This Talk

Learn how to develop Java™ Platform, Micro Edition secure payment applications using Secure And Trusted Services APIs (SATSA) and Payment API (PAPI)

# Agenda

Introduction to Mobile Payment

Java ME Security Model

Security and Trust Services API for J2ME™: SATSA (JSR 177)

Payment API: PAPI (JSR 229)

Demo: Putting Everything Together

# Agenda

**Introduction to Mobile Payment**

Java ME Security Model

Security and Trust Services API for J2ME™: SATSA (JSR 177)

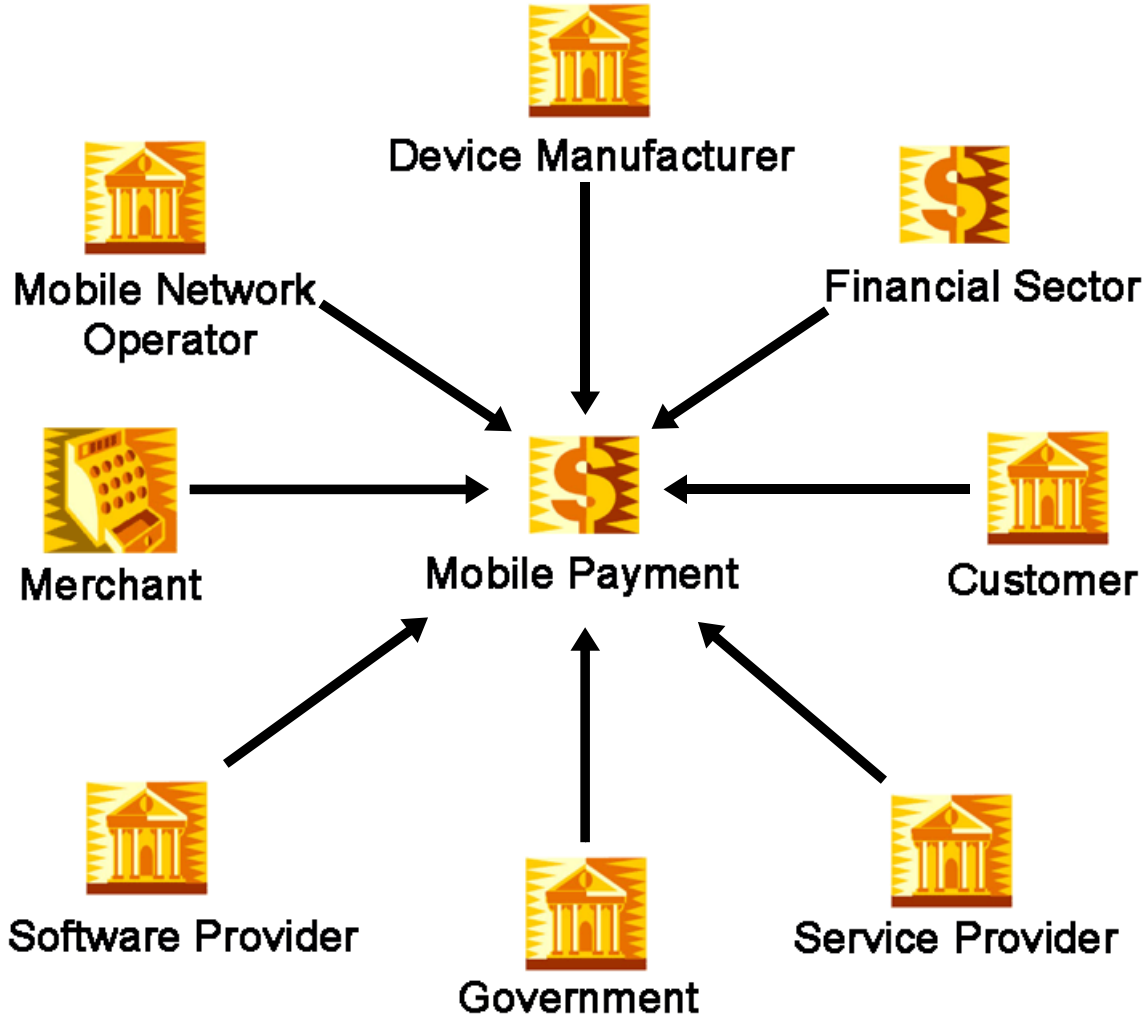
Payment API: PAPI (JSR 229)

Demo: Putting Everything Together

# Mobile Payment Motivation

- Computers are generally vulnerable and compromise security
  - You can easily cancel an ATM transaction if the user claims not to have authorized them
  - Smart card connected to the PC does not ensure security: Virus may send incorrect information to the smart card
  - Mobile personal devices, with built-in display and keyboard, provide technical solution for reducing fraud
    - Some security is already part of the authentication mechanism of existing cell phones as a way to prevent call theft
    - Inexpensive to incorporate additional mechanisms to ensure secure transaction authorization
- Convenience: Transactions anywhere

# Major Mobile Payment Players



# Transactions Categories

## By User's Location

- Remote transactions
  - Take place over the network of the user's mobile service provider
  - Examples: Downloading ring tones or video, online purchases
- Local transactions
  - Take place when the mobile device communicates with a nearby machine
  - Bluetooth instead of the mobile network
  - Examples: purchases at a store, withdrawals from a bank or payment for public transportation

# Technologies to Handle Local Transactions

- IrFM (infra-red) technology to allow Palm Pilots to act as a digital wallet; Palm and HP
- IrFM payment procedures; Verizon, Visa and many Asian companies
- “Wireless Wallet” technology, requires an always-on connection to the user’s wireless network
- Radio Frequency Identification-(RFID-) based payment procedures being developed
  - Small chip built into the cover of the phone is scanned,
  - Personal Identification Number (PIN) must be entered to authorise the payment
  - This was developed to be similar to existing credit card

# Inhibitors to the Growth of M-Commerce

| Obstacle | Phone (%) | PDA (%) |
|---|---|---|
| Credit Card Security | 52 | 47 |
| Fear of “Klunky” User Experience | 35 | 31 |
| Don't Understand How It Works | 16 | 16 |
| Never Heard of It Before | 10 | 12 |
| Other | 11 | 13 |

Source: Forrester Research

# Requirements for the Global Adoption of Mobile Payments

- Security
  - Minimizes fraud and hence reduces operating cost
  - Increase in consumer and merchant confidence
  - Increase in merchant and SP confidence
  - Security elements to be addressed
    - Authentication
    - Confidentiality
    - Data integrity
    - Non-repudiation
- Interoperability: ensuring that any participating payment product can be used at any participating merchant location
- Usability: Simplicity is required

# Agenda

Introduction to Mobile Payment

**Java ME Security Model**

Security and Trust Services API for J2ME: SATSA (JSR 177)

Payment API: PAPI (JSR 229)

Demo: Putting Everything Together

# Why Java ME for Secure Payment

- Java™ ME secure from the beginning
- Mobile-end-point-capability agnostic approach
- APIs under the JCP:
  - Payment API (PAPI) (JSR-229)
  - Secure And Trusted Services APIs (SATSA) (JSR-177)
- Netbeans Mobility Pack 5.0 +
  - Sun Java Wireless Toolkit 2.5

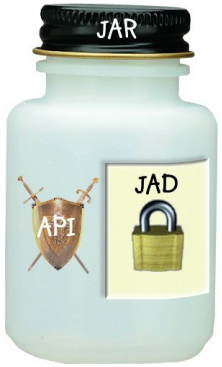
# MIDP 2.0 Security Model


Protected APIs

Setting Permissions


Protected APIs Specified

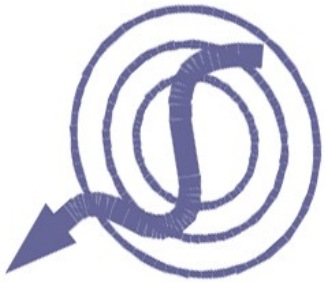
Signing




Certificate and Digital Signature

Security Policy

Application Management Software (AMS) Uses Security Policy

Installation

Over the Air

# What Is a Protection Domain?

- Defines a set of permissions (Allowed and User) that may be granted to a MIDlet suite in that domain
- Defines a set of rules that describe how MIDlet suites get into the domain
  - A signed Midlet suite should be in trusted domain
- Permission: Allow, User (blanket, session, oneshot)
- Vendor implementation issues
  - How many protection domains
  - How each protection domain is defined

# Protection Domain Config File in J2ME Wireless Toolkit

## $WTK_HOME/appdb/_policy.txt

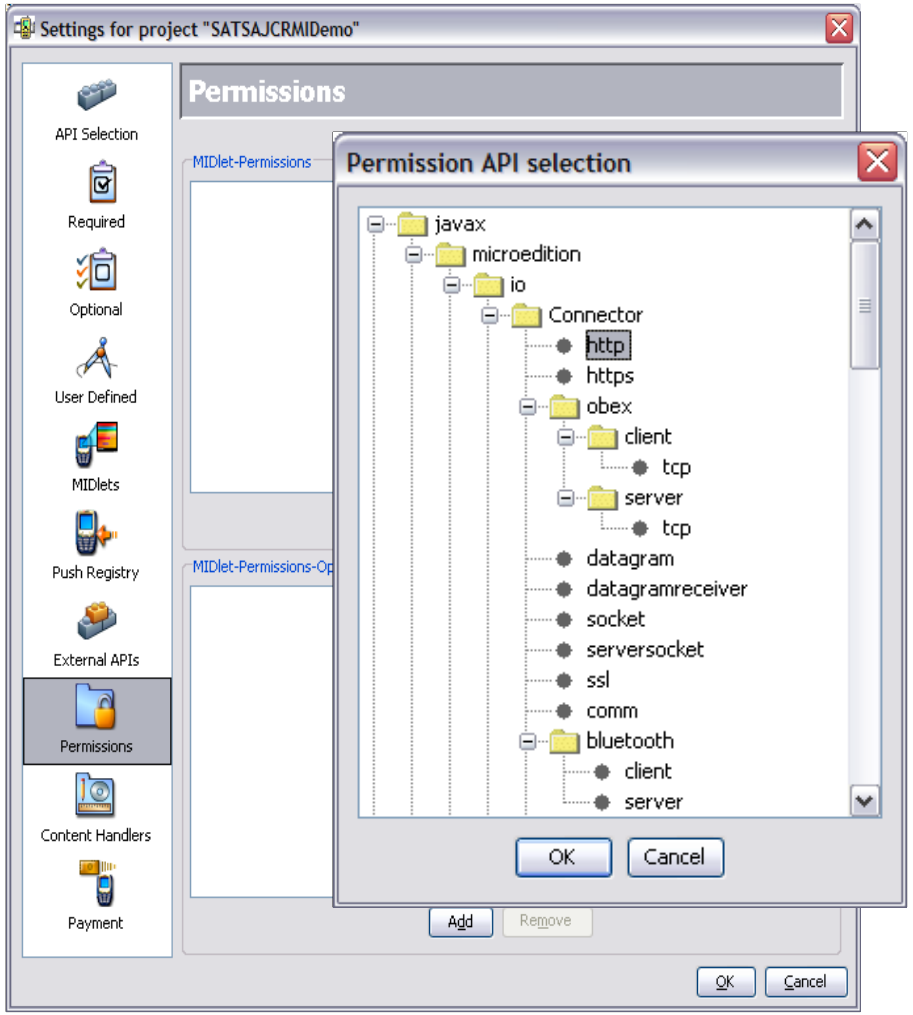
```
alias: net_access
  javax.microedition.io.Connector.http,
  javax.microedition.io.Connector.socket,
  ...
alias: application_auto_invocation
  javax.microedition.io.PushRegistry
  ...
alias: local_connectivity
  javax.microedition.io.Connector.comm

domain: minimum

domain: maximum
allow: net_access
allow: application_auto_invocation
allow: local_connectivity
...
```

# Requesting Permission Types for Midlet Suite (by MIDlet Developer)

- Specified in Java Application Descriptor (JAD) file indicating a MIDlet suite's dependence on certain permissions
- This MIDlet suite needs to make an HTTP connection and may also make socket connections
  - MIDlet-Permissions: javax.microedition.io.Connector.http
  - MIDlet-Permissions-opt: javax.microedition.io. Connector. socket
- Is a handy way to advise a device at installation time that your MIDlet suite will be attempting particular operations

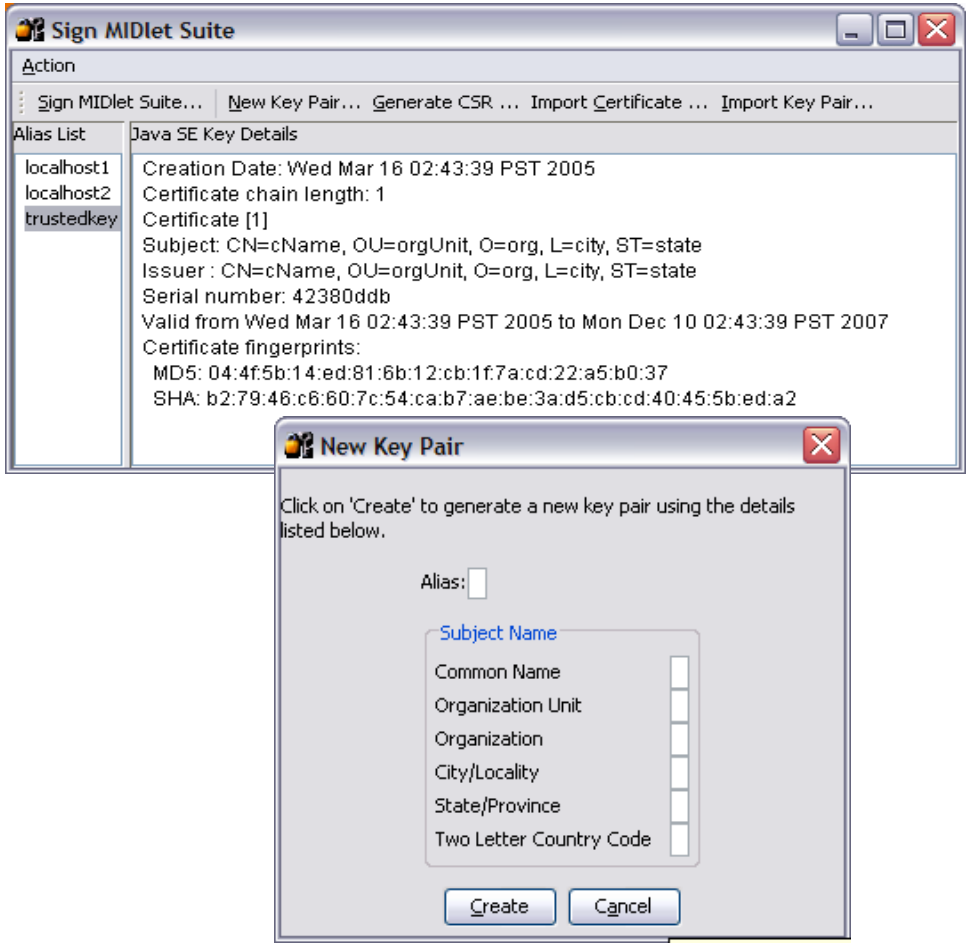
# Requesting Permission Types in Wireless Toolkit

- Set the `MIDlet-Permissions` and `MIDlet-Permissions-Opt` attributes from the permissions panel of the Project|Settings



# Sign MIDletSuite

- Select Project|Sign
- Sign MIDletSuite window opens
- Click New Key Pair button and Create, so A certificate is stored in the MEKeystore
- Select a security domain type to associate with this certificate
- Click “Sign MIDlet Suite”



# Agenda

Introduction to Mobile Payment

Java ME Security Model

**Security and Trust Services API for J2ME: SATSA (JSR 177)**

Payment API: PAPI (JSR 229)

Demo: Putting Everything Together

# Security and Trust Services API for J2ME

## JSR 177

- Provides security and trust services by integrating a Security Element (SE)
  - Secure storage to protect sensitive data: User's private keys, public key (root) certificates, service credentials, personal information
  - Cryptographic operations to support payment protocols, data integrity, and data confidentiality
  - A secure execution environment to deploy custom security features: User identification and authentication, banking, payment, loyalty applications

# The Security and Trust Services API

## Capabilities

- Smart Card Communication
  - Smart cards provide a secure programmable environment
  - Deliver a broad range of security and trust services
  - Continually upgraded with new or improved applications that can be installed on a smart card
  - Access methods based on the APDU protocol and the Java Card RMI protocol (SATSA-APDU and SATSA-Java Card RMI packages)
  - Allow a Java ME application to communicate with a smart card to leverage the security services deployed on it

# The Security and Trust Services API

## Capabilities

- Digital Signature Service and Basic User Credential Management(SATSA-PKI package)
  - Digital signature service generates digital signatures
  - Digital signatures used to authenticate end-users or to authorize transactions using public key cryptography
  - User's identity bound to a public key through a public key certificate
  - User credential management manage user credentials, such as certificates, on a user's behalf
  - Rely on a SE to provide secure storage of user credentials and cryptographic keys
  - Secure computation involving the cryptographic keys

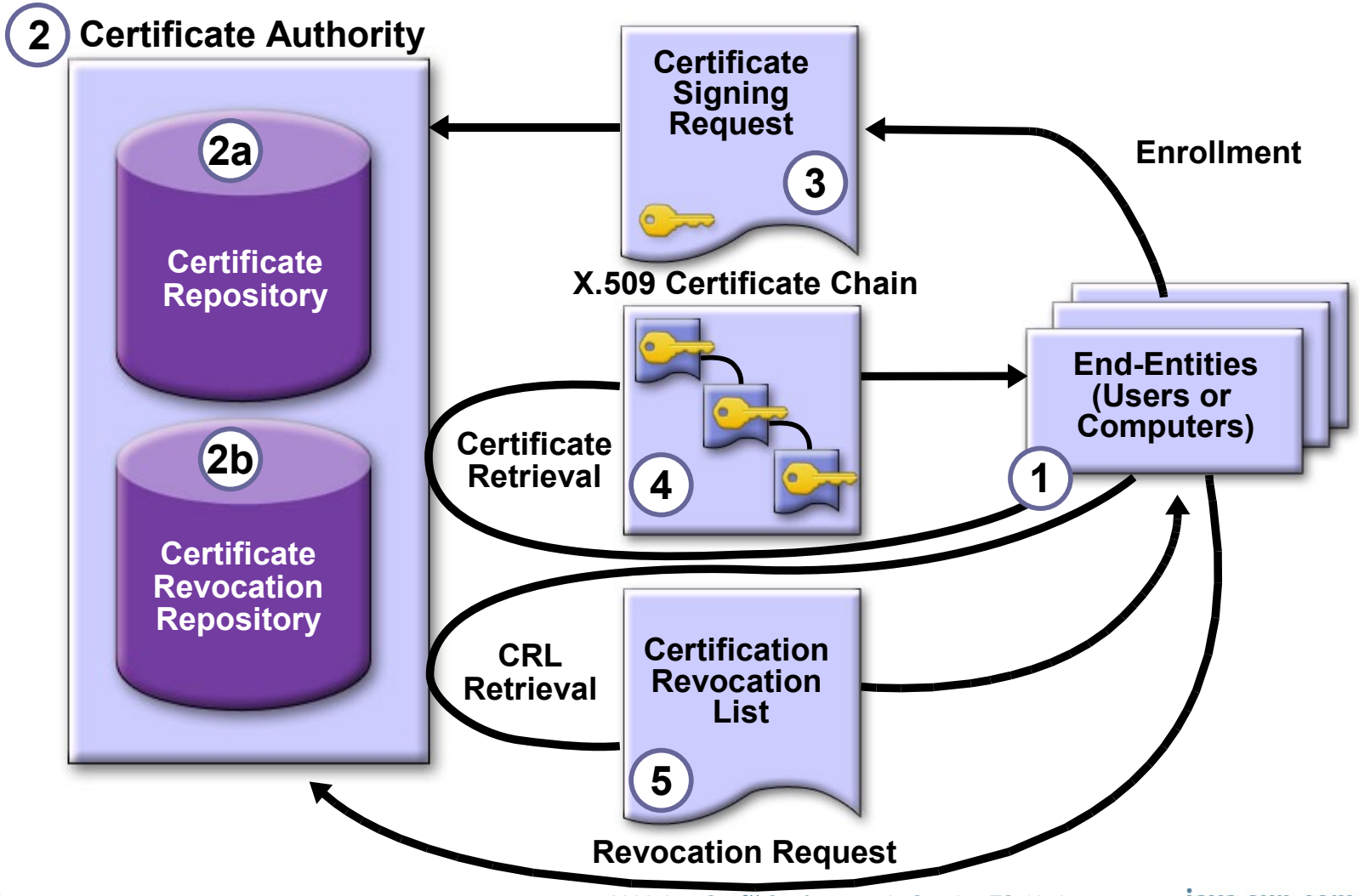
# JSR 177: Scope and Packaging

- SATSA-APDU Optional Package
  - Support communication with smart card using low-level protocol
- SATSA-JCRMI Optional Package
  - Support remote method invocation of Java Card based objects
- SATSA-PKI Optional Package
  - Support digital signature and user credential management
- SATSA-CRYPTO Optional Package
  - Support low-level cryptography operations
- Recommended practice: Access control policy

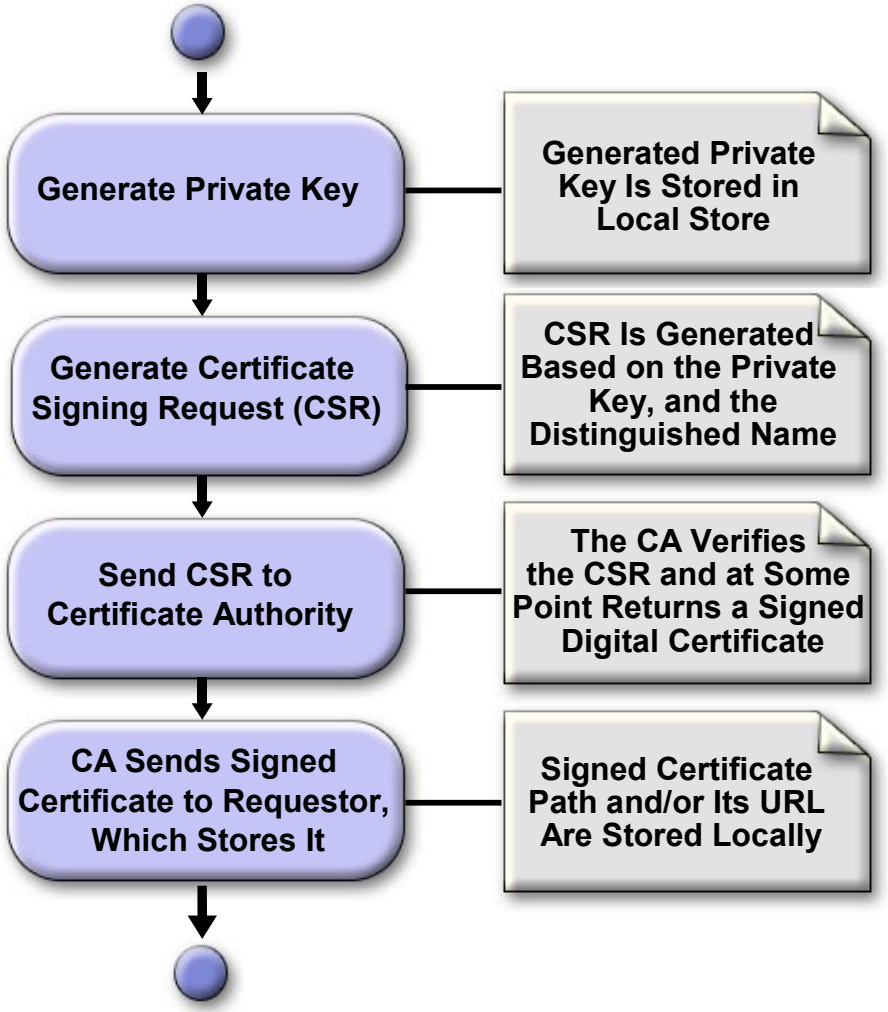
# Cryptography Goals Addressed With SATSA

- Confidentiality: Only authorized recipients can access information → Data encryption
- Data integrity: Detect if information has changed → Digital signatures
- Non-repudiation: Ensure that a transaction can't be denied → Non-repudiation type of signatures
- Authentication: Verify the source of information → Data encryption and digital signatures

# Public Key Infrastructure (PKI) Functional Elements



# Public Key Certificate Enrollment Process



# Generating a Private Key and Certificate Signing Request

```
byte[] csr = null; // Buffer for generated CSR
String distinguishedName = "CN=eortiz@j2medeveloper.com,
O=J2MEDeveloper.com,UID=eortiz,C=USA"; // The DN
int rsaKeyLength = 1024;
String securityElementID = null; //Use default SE
String securityElementPrompt = null; // No prompt
boolean forceKeyGen = true; // Generate private key
try {
    csr = UserCredentialManager.generateCSR(
            distinguishedName,
            UserCredentialManager.ALGORITHM_RSA,
            rsaKeyLength,
            UserCredentialManager.KEY_USAGE_AUTHENTICATION,
            securityElementID,
            securityElementPrompt,
            forceKeyGen);
} catch (Exception e) {
    /* Handle IllegalArgumentException or UserCredential
       ManagerException or SecurityException or
       CMSMessageSignatureServiceException  */...
}
```

# Requesting the Signed Certificate (Verifying the CSR)

```
/* Send the generated CSR to the CA enrollment server,
   possibly over a secure TCP (SecureConnection) or
   HTTPS (HttpsConnection). Wait for response (the
   signed X.509 certificate chain)  */
String url = "www.j2medeveloper-ca.com:443";
byte[] response = secureSend(url, csr);
...
```

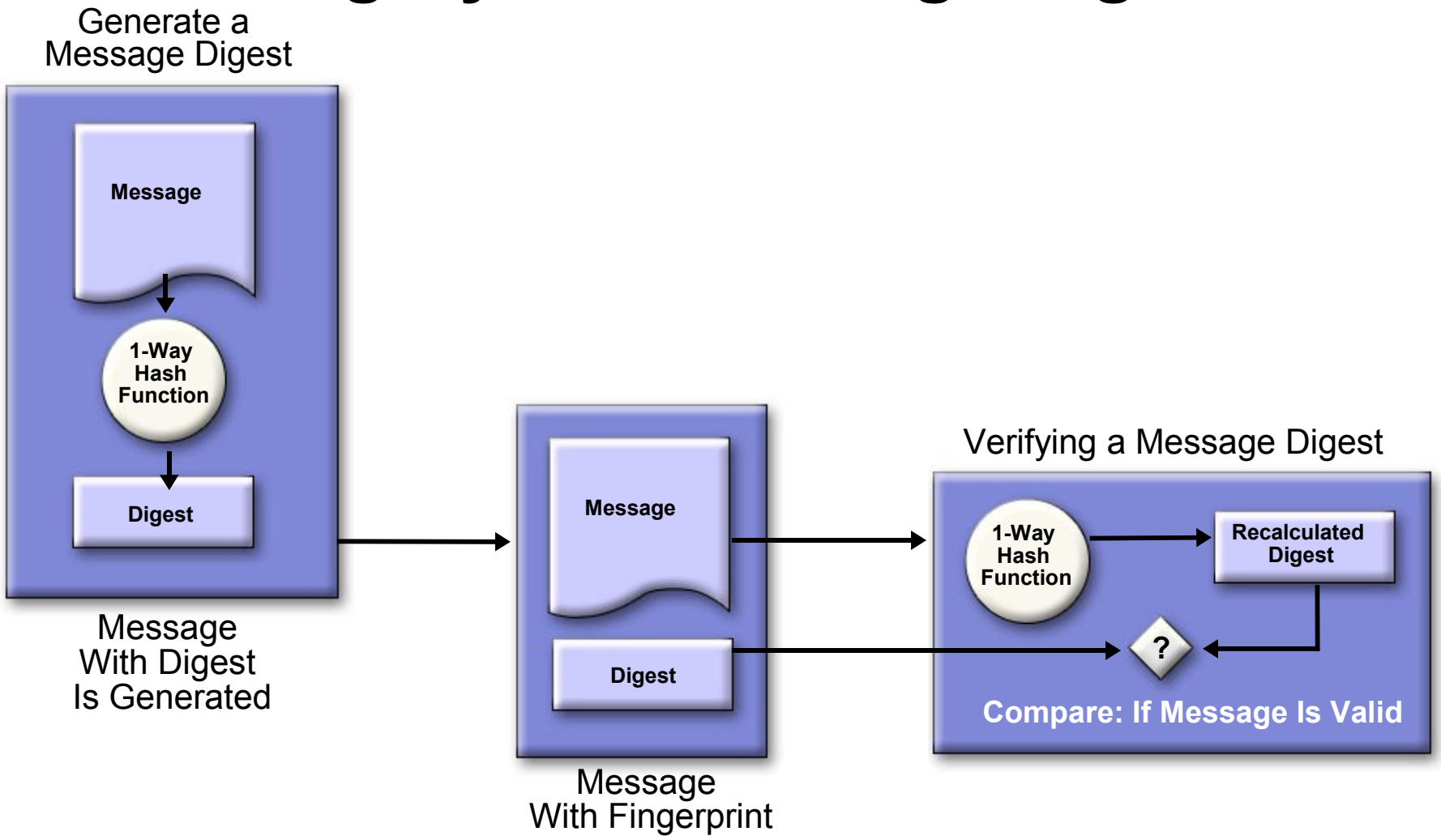
# Storing the Certificate

```
/* Parse response, extracting the signed X.509
   certificate information. Store the received
   certificate on the security element. */
UserCredentialManager.addCredential(certDisplayName,
        pkiPath,
        Uri);   // from the enrollment response
```

# Managing Certificate's Local Store

```
// The certificate friendly name
String certDisplayName = new String("MyCertificate");
// The certificate path and URI.
byte[] pkiPath = "..."; // from the enrollment response
String uri = "..."; // from the enrollment response

try {
      // Store the received certificate on
      // the security element.The pkiPath and URI are
      // extracted from the message received from the CA.
    boolean added;
    added = UserCredentialManager.addCredential(
        certDisplayName,
        pkiPath,
        uri);
} catch (Exception e) {
    // Handle IllegalArgumentException or
    // UserCredentialManagerException or
    // SecurityException
    ...
}
```

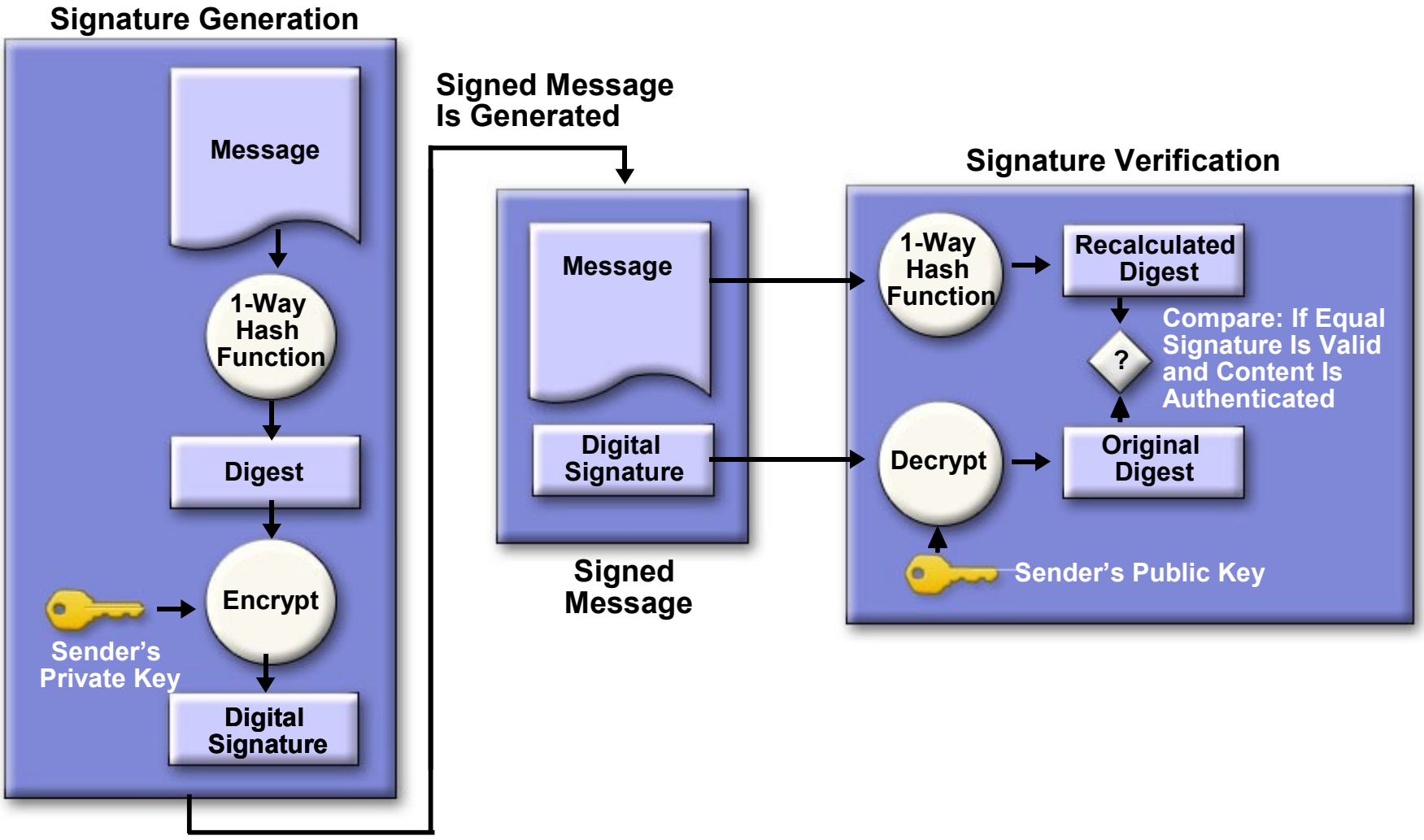
# Data Integrity with Message Digest



# Generating a Message Digest

```
static String digestAlgo = "SHA-1";
static int shaDigestLen = 20;
byte[] message = "..."; // original message
byte[] newDigest = new byte[shaDigestLen];

try {
    MessageDigest md;
    md = MessageDigest.getInstance(digestAlgo);
    md.update(message, 0, message.length);
    md.digest(newDigest, 0, shaDigestLen);
} catch (Exception e) {
    // Handle NoSuchAlgorithmException or DigestException
    ...
}
```

# The Signing Process



# Agenda

Introduction to Mobile Payment

Java ME Security Model

Security and Trust Services API for J2ME: SATSA (JSR 177)

**Payment API: PAPI (JSR 229)**

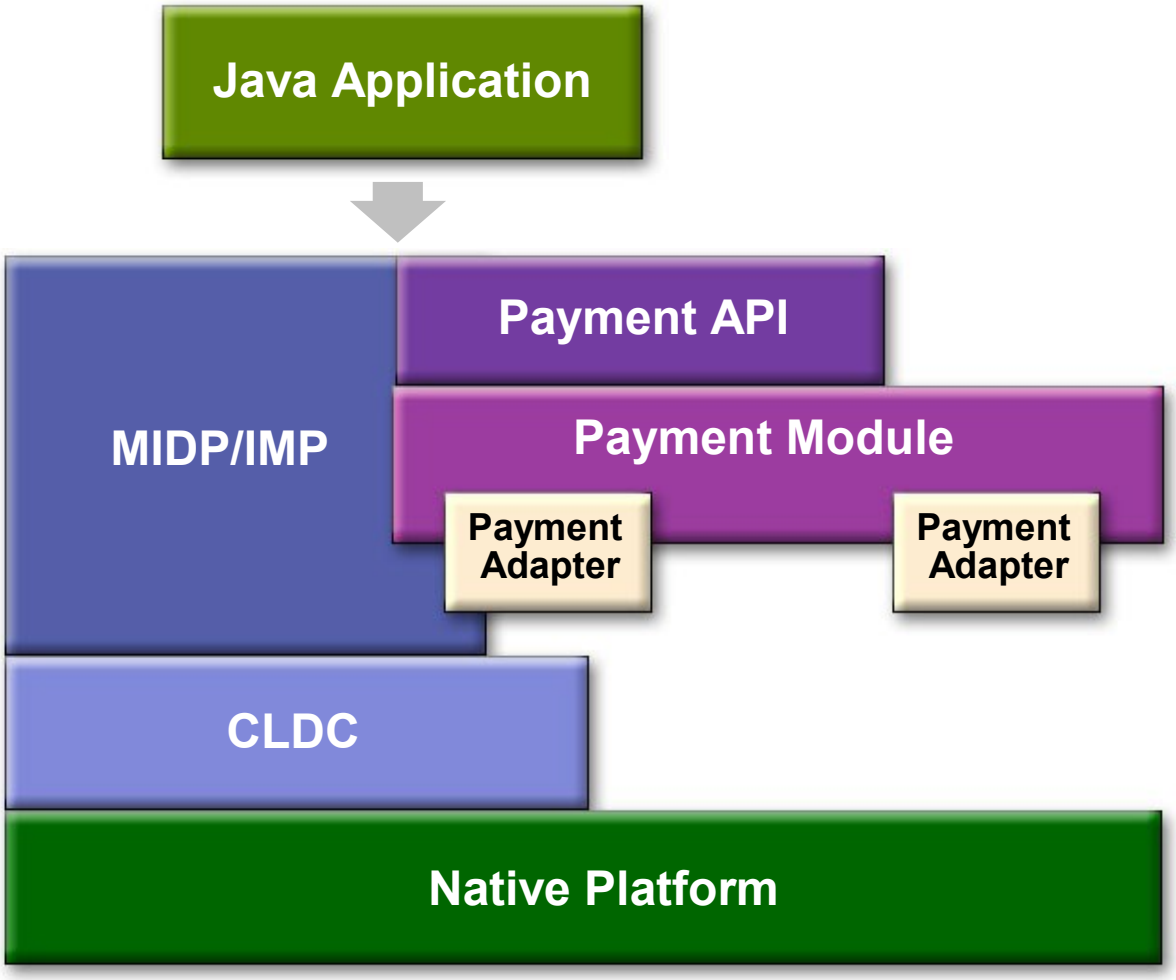
Demo: Putting Everything Together

# Payment API—JSR 229

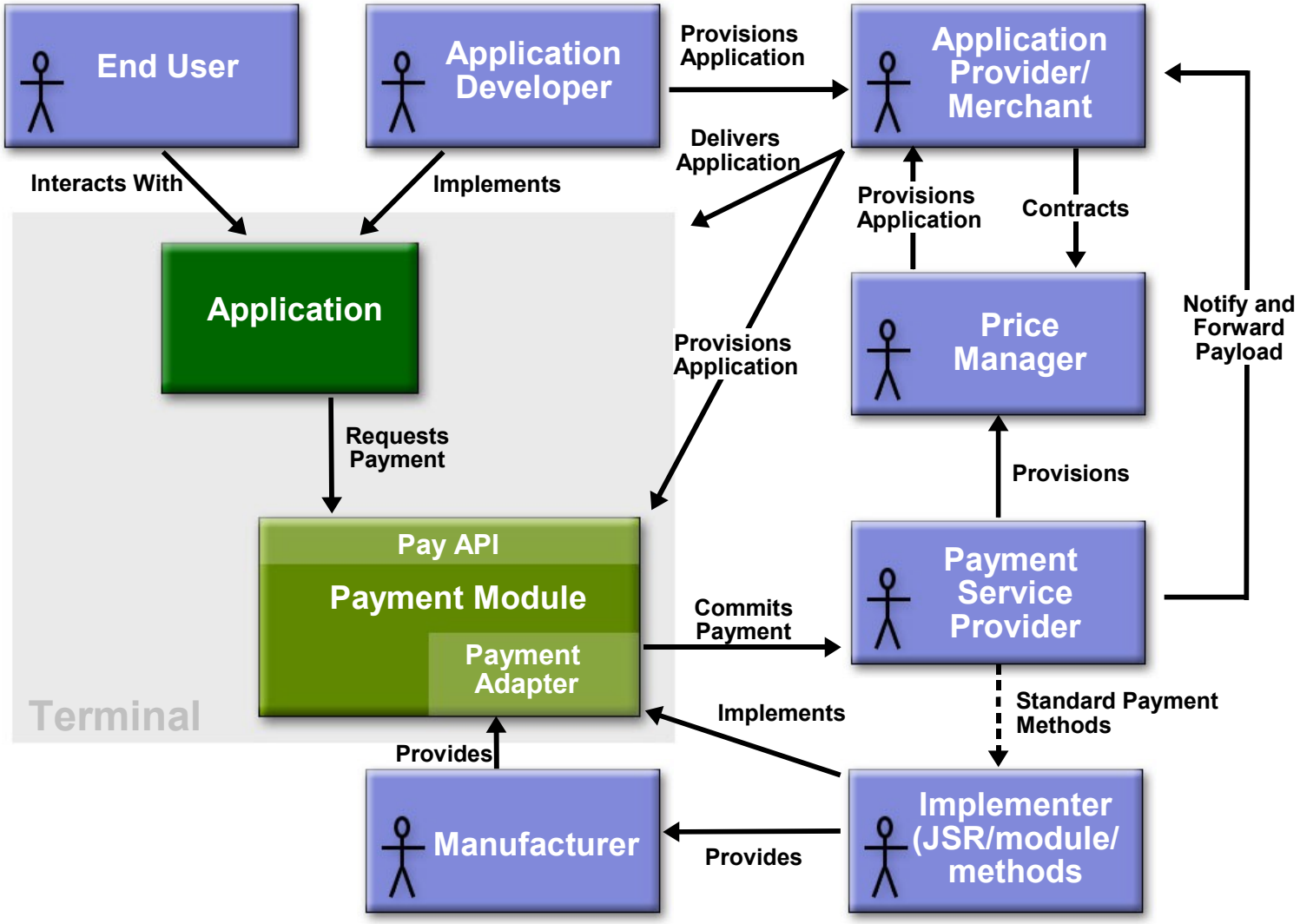
## What Does It Allow You to Do?

- Initiate payment transactions in a secured manner to transparently expedite the chargeable service requests
  - Requesting a payment transaction
  - Requesting feature and service price management
  - Payment service availability
- Provide a generic payment initiation mechanism that hides the actual payment architecture and complexity from the developers
- Does not define and imply any concrete payment implementation and mechanism

# General Architecture



# Functional Overview



# Payment Module Responsibilities

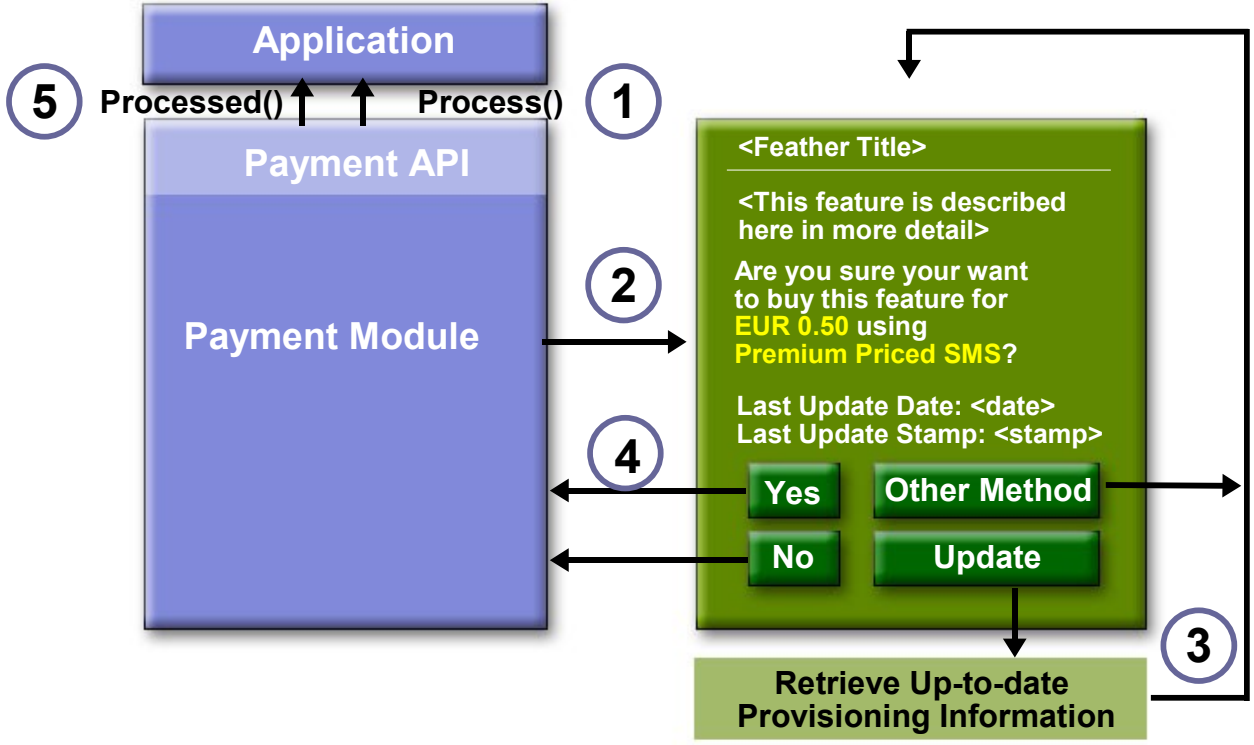
## Provisioning Mapping

- Applications include provisioning data for the different payment adapters
- Payment module load the payment provisioning data from the JAR-Manifest Resource file
- Find possible syntax errors
- Map the provisioning data to the appropriate payment adapter
- When to map? Up to the implementation to decide

# Payment Module Responsibilities

## Pricing Updates

- Price information delivered with application
- How often or when price updates should be conducted?



# Payment Module Responsibilities

- Method selection
  - Payment API is payment adapter agnostic
  - Show all operational payment methods available to the user
- Transaction and update history
  - Keep history of the latest provisioning update as well as all missed transactions and a reasonable number of past transactions
- Dynamic payment adapter management (optional)
  - Responsible for downloading, checking, installing and registering that particular payment adapter plug-in

# Payment Adapter Responsibilities

- Conducts payment transactions
  - Focus on at least one particular payment method
  - All payment methods MUST involve an interaction between this adapter and servers in the network
- Payment authentication
  - Should include non-repudiation mechanisms and user authentication
  - It's up to the payment adapter implementation
  - Use SATSA

# Payment API Overview

## javax.microedition.payment

### `TransactionModule`

- Represents the communication interface between the application and the payment module
- Support asynchronous payment handling
- `process()`
  - Return immediately after passing the values to the payment adapter
- An event is generated as a result of the payment transaction, and the corresponding record is passed through the `processed()` method of the `TransactionListener`

# Payment API Overview

## javax.microedition.payment

**`TransactionListener`**

- Receives notifications of transaction records that have been generated by the payment module once a transaction has been processed
- `processed()` indicate that a transaction-related event has occurred: `TRANSACTION_SUCCESSFUL, TRANSACTION_FAILED, TRANSACTION_REJECTED`

**`TransactionRecord`**

- Represents an atomic payment transactions
- `GetFeatureID(), getTransactionID(), getState(), getFinishedTimestamp(), wasMissed()`

# Payment Example (1 of 2)

```
import javax.microedition.payment.*;
...

public class MyGame extends MIDlet implements
   TransactionListener, CommandListener {

private TransactionModule myTransactionModule;

  public MyGame(){
    ...
    try {
      myTransactionModule = new TransactionModule(this);
    } catch(TransactionModuleException e) {
    // print error messages
    }
    try {
      myTransactionModule.setListener(this);
    }catch(Exception e) {...}
  }
```

# Payment Example (2 of 2)

```
public void startApp(){
  ...
  try {
    myTransactionModule.process(featureID, "Feature Title",
                                "Feature description");
    synchronized(this) {
    try {
      wait(); // wait until callback is called
    }catch (InterruptedException ie) {// Handle exception}
    }
  }catch (Exception e) { // Handle exception}...
}

public void processed(TransactionRecord myRecord){
  switch(myRecord.getState()) {
    case TransactionRecord.TRANSACTION_SUCCESSFUL:
      // Payment transaction successful
      break;
    case TransactionRecord.TRANSACTION_REJECTED:
      // Payment rejected
      break;
    case TransactionRecord.TRANSACTION_FAILED:
    default:
      // Technical problem - try again!
    break;
   }...
}
```

# Payment Example

JAD

```
Pay-Version: 1.0
Pay-Adapters: PPSMS, X-TEST
MIDlet-Permissions: javax.microedition.payment.process.jpp
MIDlet-Certificate-<n>-<m>: <base64 encoding of a certificate>
MIDlet-Jar-RSA-SHA1: <base64 encoded Jar signature>
```

JAR-Manifest

```
Pay-Version: 1.0
Pay-Update-Stamp: 2004-11-15 02:00+01:00
Pay-Providers: SMS1, Test1Card
Pay-Update-URL: http://<update-site>/thisgame.manifest.jpp
Pay-Cache: no
Pay-Feature-0: 0
Pay-Feature-1: 0
Pay-Feature-2: 1
Pay-SMS1-Info: PPSMS, EUR, 928, 99
Pay-SMS1-Tag-0: 1.20, 9990000, 0x0cba98765400
Pay-SMS1-Tag-1: 2.50, 9990000, 0x0cba98765401, 2
Pay-Test1Card-Info: X-TEST8, EUR, c4d21, soap://<soap-site-1>/
Pay-Test1Card-Tag-0: 1.21
Pay-Test1Card-Tag-1: 2.46
```

# Agenda

Introduction to Mobile Payment

Java ME Security Model

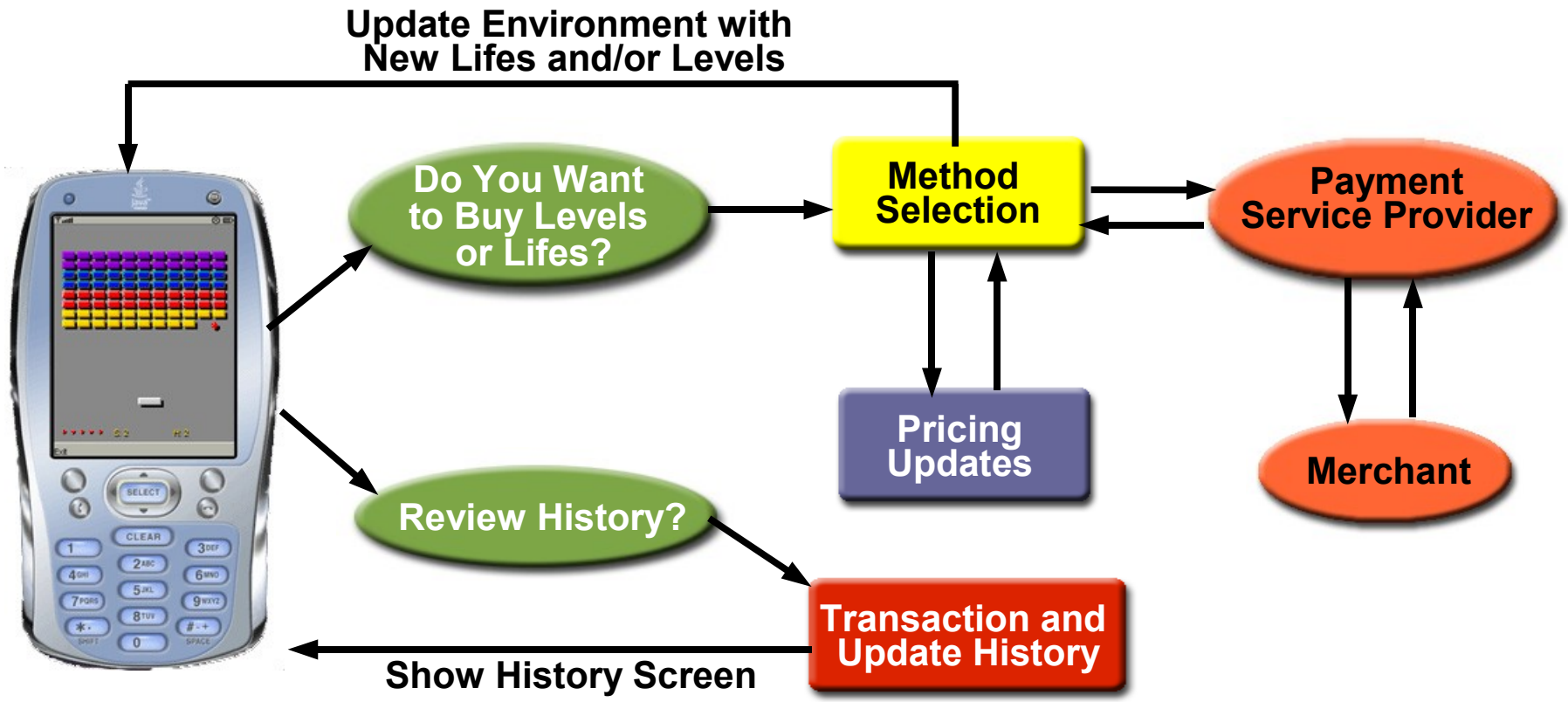
Security and Trust Services API for J2ME: SATSA (JSR 177)

Payment API: PAPI (JSR 229)

**Demo: Putting Everything Together**

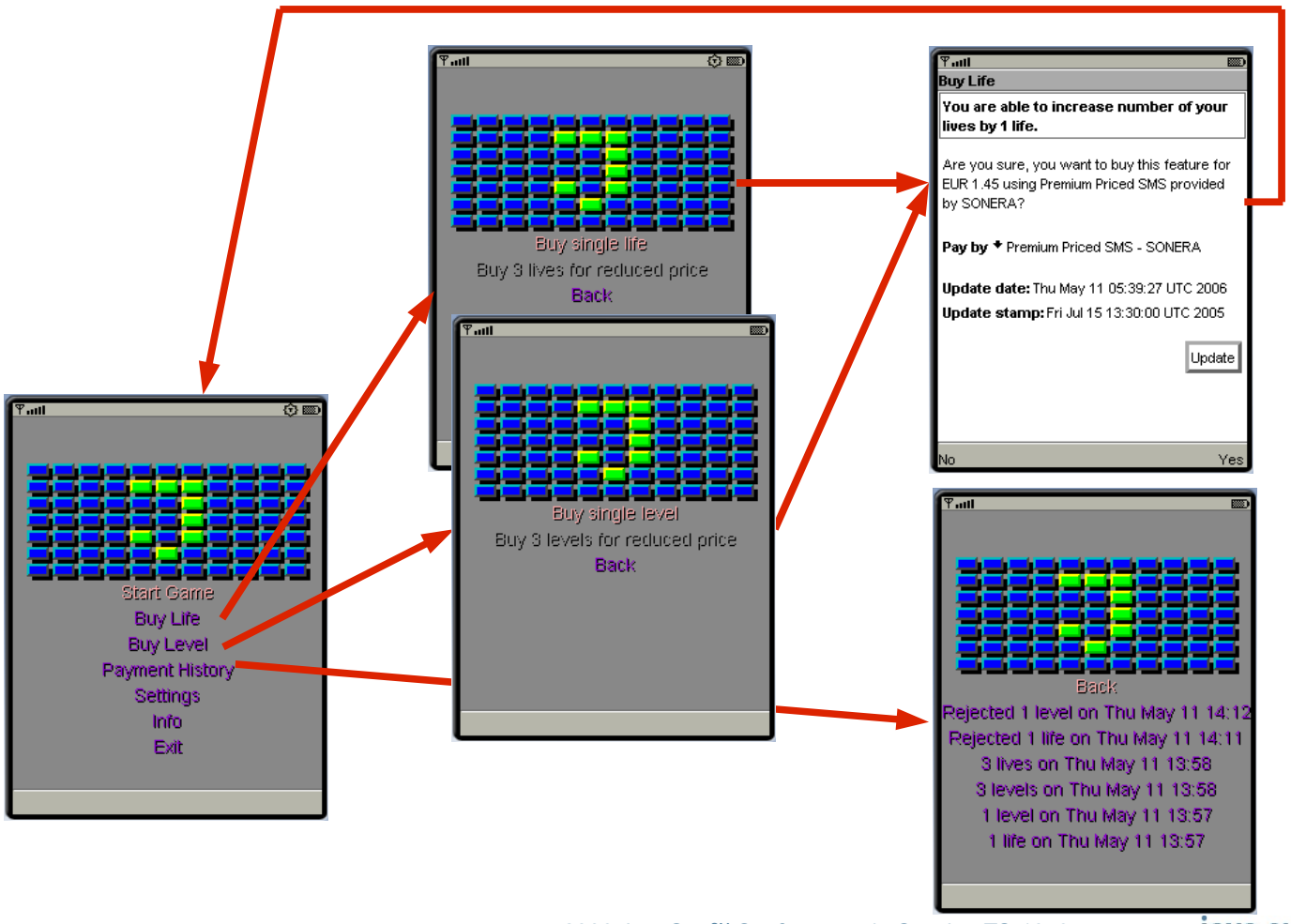
# Designing Your Application

## Buying Lifes and Levels for Your Game



# Designing Your Application

## Buying Lifes and Levels for Your Game



# Getting Ready for the Game

```
public Main() {
    screen = new Screen(this);
    engine = new Engine(screen);

    screen.setCommandListener(this);

    try {
        txModule = new TransactionModule(this);
        txModule.setListener(this);
        txModule.deliverMissedTransactions();
        restoreBoughtFeatures();
    } catch (TransactionListenerException tle) {
        tle.printStackTrace();
    } catch (TransactionModuleException tme) {
        tme.printStackTrace();
    }
}
```

# Restoring Previously Bought Features

```
private void restoreBoughtFeatures() {
    TransactionRecord[] record =
            txModule.getPastTransactions(10);

    if (record != null) {
     for (int i = 0; i < record.length; i++){
        processed(record[i]);
     }
    }
}
```

# Processing Transaction Records

```
public void processed(TransactionRecord record) {
    switch (record.getState()) {
     cas e TransactionRecord.TRANSACTION_SUCCESSFUL:
            switch (record.getFeatureID()) {
                case FEATURE_1_LIFE:
                    engine.increaseNumOfLives(1);
                    break;
                case FEATURE_3_LIVES:
                    engine.increaseNumOfLives(3);
                    break;
                case FEATURE_1_LEVEL:
                    engine.increaseNumOfLevels(1);
                    break;
                case FEATURE_3_LEVELS:
                    engine.increaseNumOfLevels(3);
                    break;
            }
            break;
        default:
    }
}
```

# Processing Transactions

```
public void run() {
    try {
        txModule.setListener(this);
        txModule.process(feature, title, description);
        if (enableTranListenerNull) {
            txModule.setListener(null);
        }
    } catch (TransactionListenerException tle) {
        System.err.println("Transaction Listener not
                                    set");
    } catch (Exception e) {
        e.printStackTrace();
    }
}
```

```
title = "Buy Life";
feature = FEATURE_1_LIFE;
description = "You are able to increase number of your
                lives by 1 life.";
```

# Getting Transaction History

```
TransactionRecord[] record =
                                txModule.getPastTransactions(6);
String[] stringRecord = null;
...
stringRecord = new String[record.length];
for (int i = 0; i < record.length; i++) {
    switch (record[i].getState()) {
        case TransactionRecord.TRANSACTION_FAILED:
            feature = "Failed ";
            break;
        case TransactionRecord.TRANSACTION_REJECTED:
            feature = "Rejected ";
            break;
    }
    switch (record[i].getFeatureID()) {
        case FEATURE_1_LIFE:
            feature += "1 life";
            break;
   ...}
date.setTime(record[i].getFinishedTimestamp());
when = date.toString();
stringRecord[i] = feature + " on " + when.substring(0,
                  when.lastIndexOf(':'));
```

# Provisioning (1 of 2)

JAD

```
Pay-Version: 1.0
Pay-Adapters: PPSMS,X-CCARD
MIDlet-Permissions: javax.microedition.payment.process,
                    javax.wireless.messaging.sms.send,
                    javax.microedition.io.Connector.http,
                    javax.microedition.io.Connector.https,
                    javax.microedition.io.Connector.sms
MIDlet-Certificate-<n>-<m>: <base64 encoding of a certificate>
MIDlet-Jar-RSA-SHA1: <base64 encoded Jar signature>
```

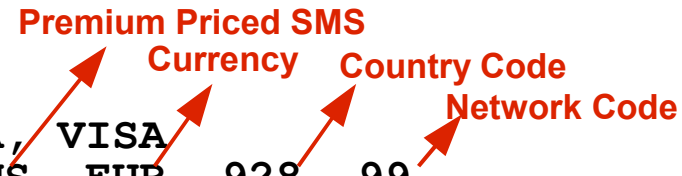
JAR

```
Pay-Version: 1.0
Pay-Update-Stamp: 2004-08-12T13:30:00Z
Pay-Update-URL: http://localhost/jbricks/bin/jbricks.jpp
Pay-Providers: SONERA, VISA, RADIOG, DNSDNA, MASTERCARD, AMEX
Pay-Cache: no
Pay-Feature-0: 0
Pay-Feature-1: 1
Pay-Feature-2: 2
Pay-SONERA-Info: PPSMS, EUR, 928, 99
Pay-SONERA-Tag-0: 1.40, 5550000, 1_LIFE
Pay-SONERA-Tag-1: 2.80, 5550000, 3_LIVES, 2
Pay-SONERA-Tag-2: 2.10, 5550000, 1_LEVEL
Pay-SONERA-Tag-3: 4.20, 5550000, 3_LEVELS, 2
...
```

# Provisioning (1 of 2)

jBricks.jpp

```
Pay-Adapters: X-CCARD,PPSMS
Pay-Cache: yes
Pay-Feature-0: 0
Pay-Feature-1: 1
Pay-Feature-2: 2
Pay-Feature-3: 3
Pay-Providers: SONERA, VISA
Pay-SONERA-Info: PPSMS, EUR, 928, 99
Pay-SONERA-Tag-0: 1.45, 5550000, Lives-1
Pay-SONERA-Tag-1: 2.85, 5550000, Lives-3, 2
Pay-SONERA-Tag-2: 2.15, 5550000, Levels-1
Pay-SONERA-Tag-3: 4.25, 5550000, Levels-3, 2
Pay-Update-Stamp: 2005-07-15T13:30:00Z
Pay-Update-URL: http://localhost/jbricks/bin/jbricks.jpp
Pay-VISA-Info: X-CCARD, EUR, VISA, https://localhost
Pay-VISA-Tag-0: 1.55, LIVES-1
Pay-VISA-Tag-1: 3.05, LIVES-3
Pay-VISA-Tag-2: 2.25, LEVELS-1
Pay-VISA-Tag-3: 4.45, LEVELS-3
Pay-Version: 1.0
Pay-Certificate-1-1: MIICEDCCAXkCBEI4DdswDQY...a4vfSg==
Pay-Signature-RSA-SHA1: EnJZ8oBsQTxLjVvOd..KP2Fv9eJP1s=
```



# Provisioning (1 of 2)

jBricks.jpp

```
Pay-Adapters: X-CCARD,PPSMS
Pay-Cache: yes
Pay-Feature-0: 0
Pay-Feature-1: 1
Pay-Feature-2: 2
Pay-Feature-3: 3
Pay-Providers: SONERA, VISA
Pay-SONERA-Info: PPSMS, EUR, 928, 99
Pay-SONERA-Tag-0: 1.45, 5550000, Lives-1
Pay-SONERA-Tag-1: 2.85, 5550000, Lives-3, 2
Pay-SONERA-Tag-2: 2.15, 5550000, Levels-1
Pay-SONERA-Tag-3: 4.25, 5550000, Levels-3, 2
Pay-Update-Stamp: 2005-07-15T13:30:00Z
Pay-Update-URL: http://localhost/jbricks/bin/jbricks.jpp
Pay-VISA-Info: X-CCARD, EUR, VISA, https://localhost
Pay-VISA-Tag-0: 1.55, LIVES-1
Pay-VISA-Tag-1: 3.05, LIVES-3
Pay-VISA-Tag-2: 2.25, LEVELS-1
Pay-VISA-Tag-3: 4.45, LEVELS-3
Pay-Version: 1.0
Pay-Certificate-1-1: MIICEDCCAXkCBEI4DdswDQY...a4vfSg==
Pay-Signature-RSA-SHA1: EnJZ8oBsQTxLjVvOd..KP2Fv9eJP1s=
```

Phone number to which the payment SMS will be sent

Prefix prepended to any payment SMS

Number of payment SMS that MUST be sent

# How Is Security Involved?

## Security for Deployment

- PAPI 1.0 takes advantage of security features and control mechanisms provided by MIDP2.0
- The MIDlet MUST be protected through the signature of the JAR application by an authority recognized by the platform certificate authority
- “debug mode” does not required signature nor certification
- “debug mode” only for development platforms such as the Java Wireless Toolkit

# How Is Security Involved?

## Security Used in the Adapters Implementation

- Particular payment adapters involve non-repudiation mechanisms, user authentication and other security features
- This task is outside this payment specification
- **SATSA great solution!**
- Scenarios:
  - Credit-cardbased adapter may ask the user for the card number and expiration date every time it is used
  - It may also need an X.509 certificate or access to a smartcard to authenticate itself to the credit card operator in the network

# How Is Security Involved?

## Security Storage

- RMS API alone:
  - The simplest to use
  - Potential security holes if RMS records are located on a file in the file system
  - Not recommended alone
- FileConnection API of JSR 75 “PDA Optional Packages for the Java ME Platform”
- JSR 177 “Security and Trust Services API (SATSA)”
  - Mechanisms to store information securely (i.e. RMS with encryption and Security Element)

# DEMO

See Everything in Action

# Summary

- Mobile payments are simply the next step in the evolution of how transactions are made
- Java ME and it's optional APIs are a great solution
- Payment API provides you with a payment agnostic API
- Secure And Trusted Services API give you all the security you required for developing a great Secure Payment Applications

# Q&A

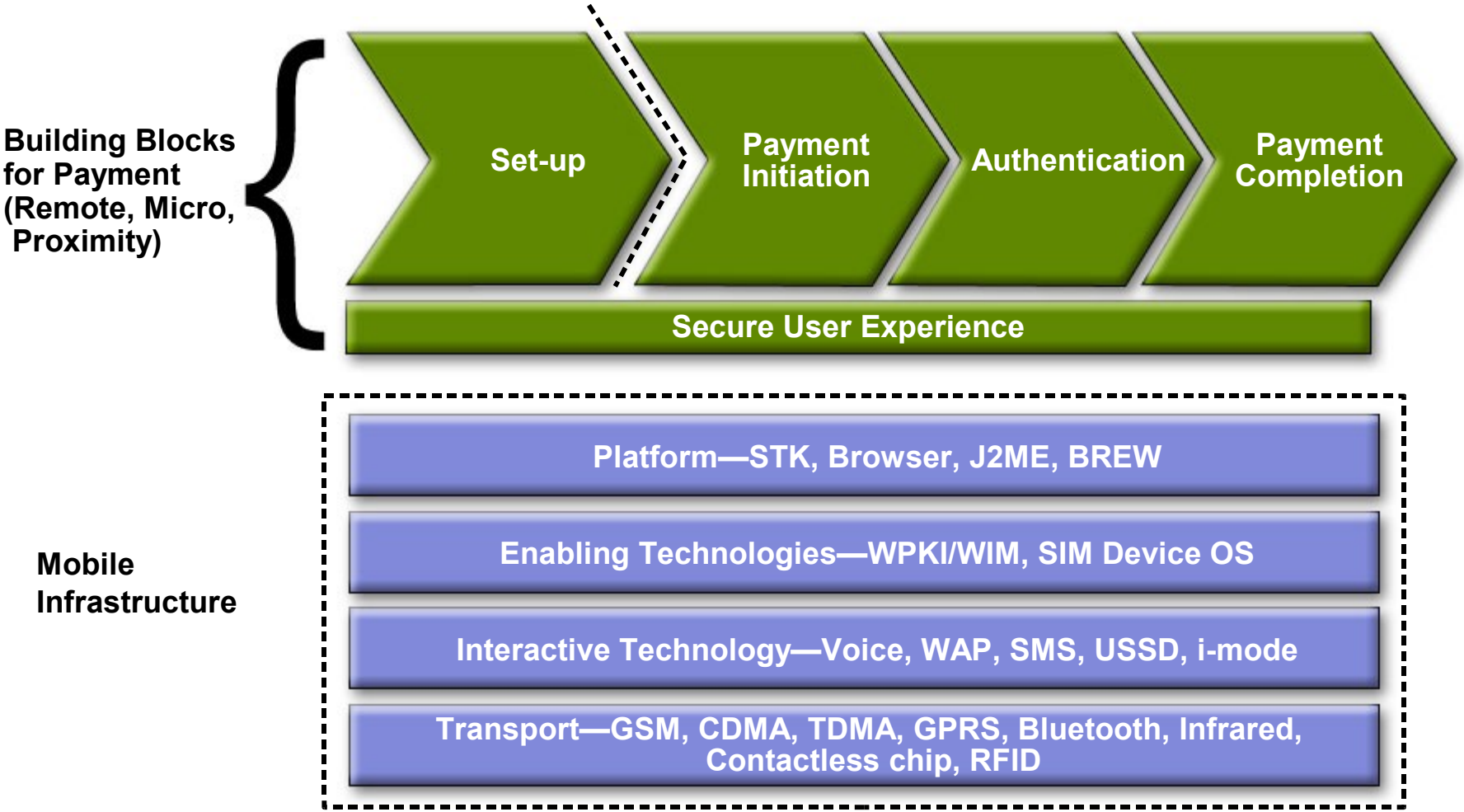
# Techniques and Tips: Developing Secure Payment Applications, Using Java™ ME Technology

**Angela Caicedo**
**Doris Chen Ph.D.**

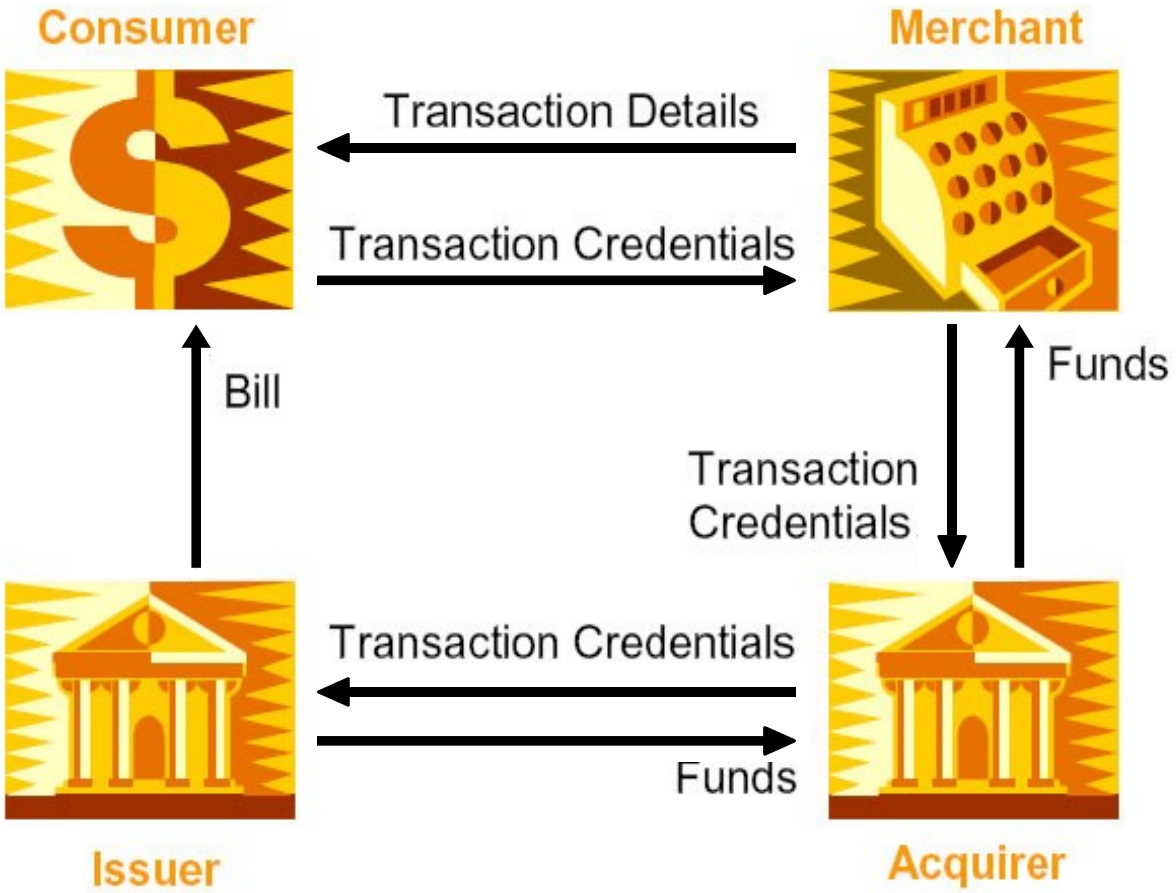
Technology Evangelists
Sun Microsystems

TS-1049

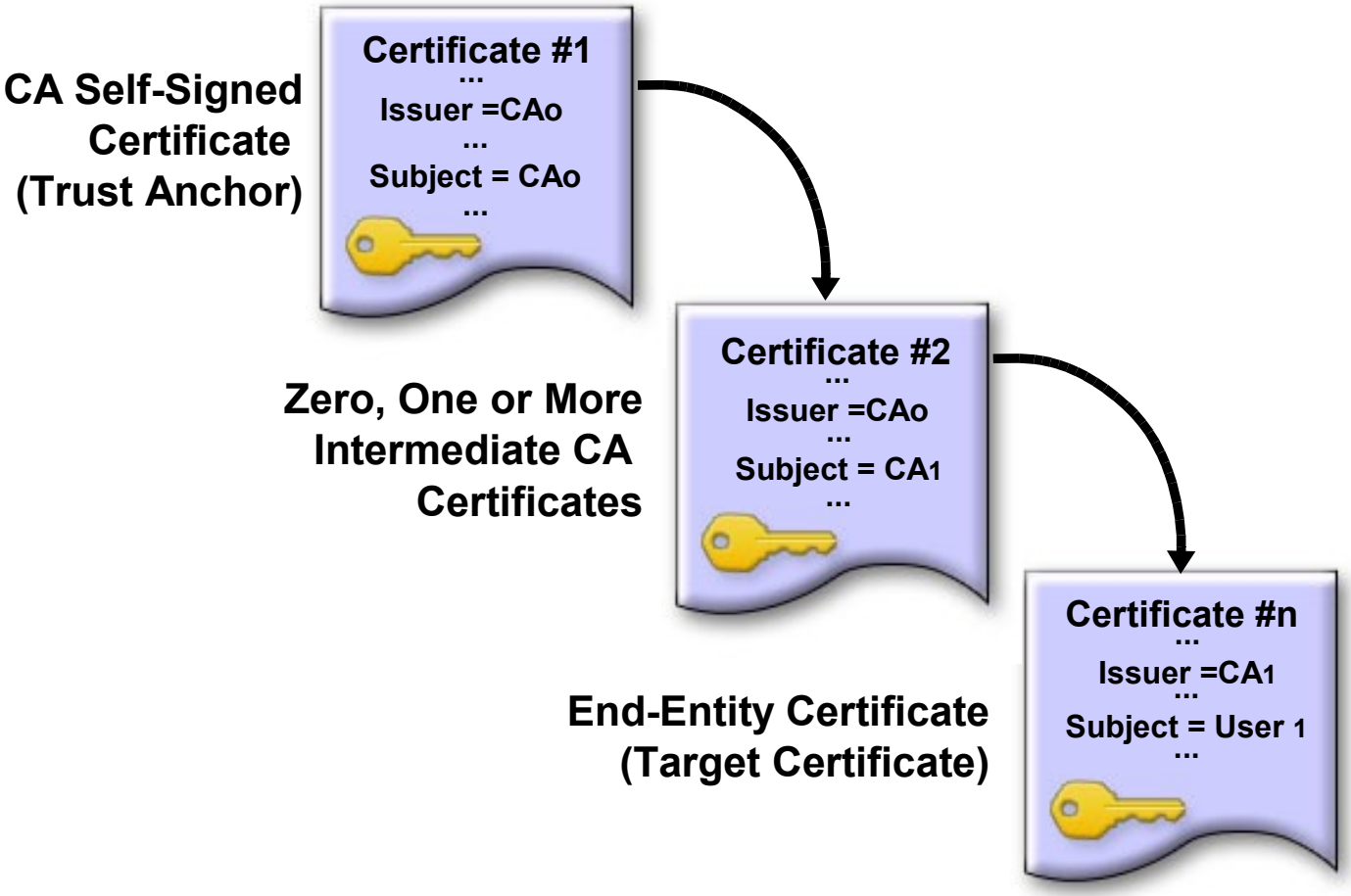
# Payment Lifecycle



# Payment Transaction



# A Certificate Path or PkiPath



# Signing a String for Authentication Purposes

```
String stringToSign = "...";
String securityElementPrompt = null; // Don't prompt
byte[] signature;
String myCaDN = "..."; // The CA DN, from the certificate
String[] caNames = new String[] { myCaDN };
try {
    // Sign the specified string. Include the certificate
    // and content as well.
    signature = CMSMessageSignatureService.authenticate(
        stringToSign,
        CMSMessageSignatureService.SIG_INCLUDE_CERTIFICATE
        |CMSMessageSignatureService.SIG_INCLUDE_CONTENT,
        CaNames, securityElementPrompt);
} catch (Exception e) {...
}
```

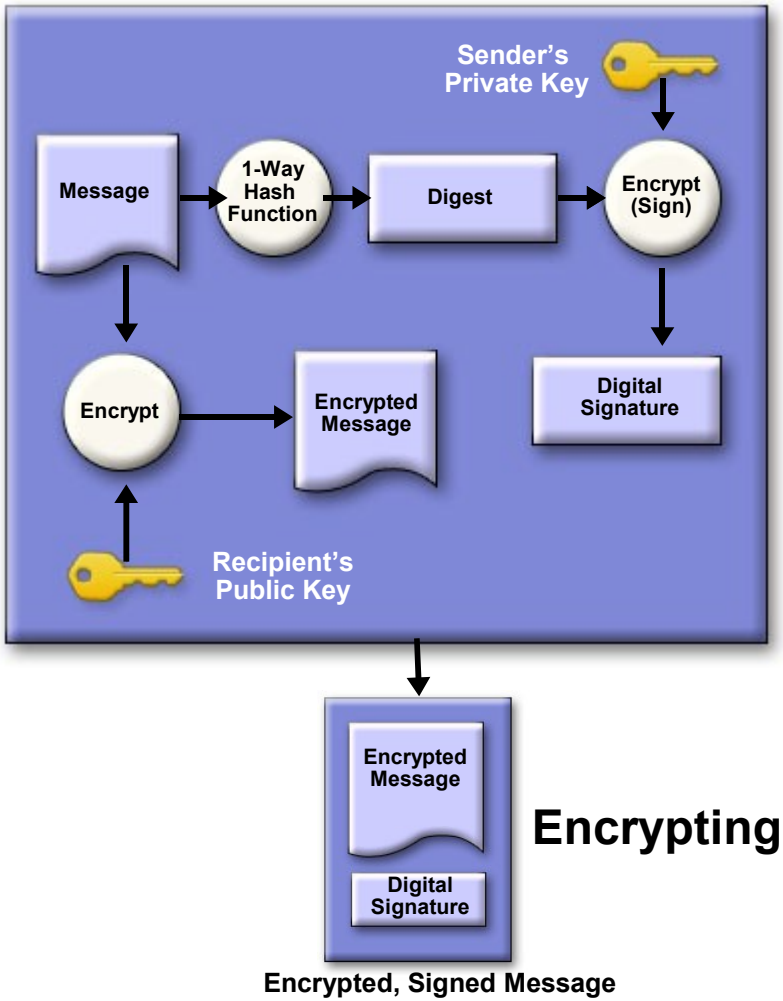
**NOTE: Signing for Non-Repudiation Purposes:**

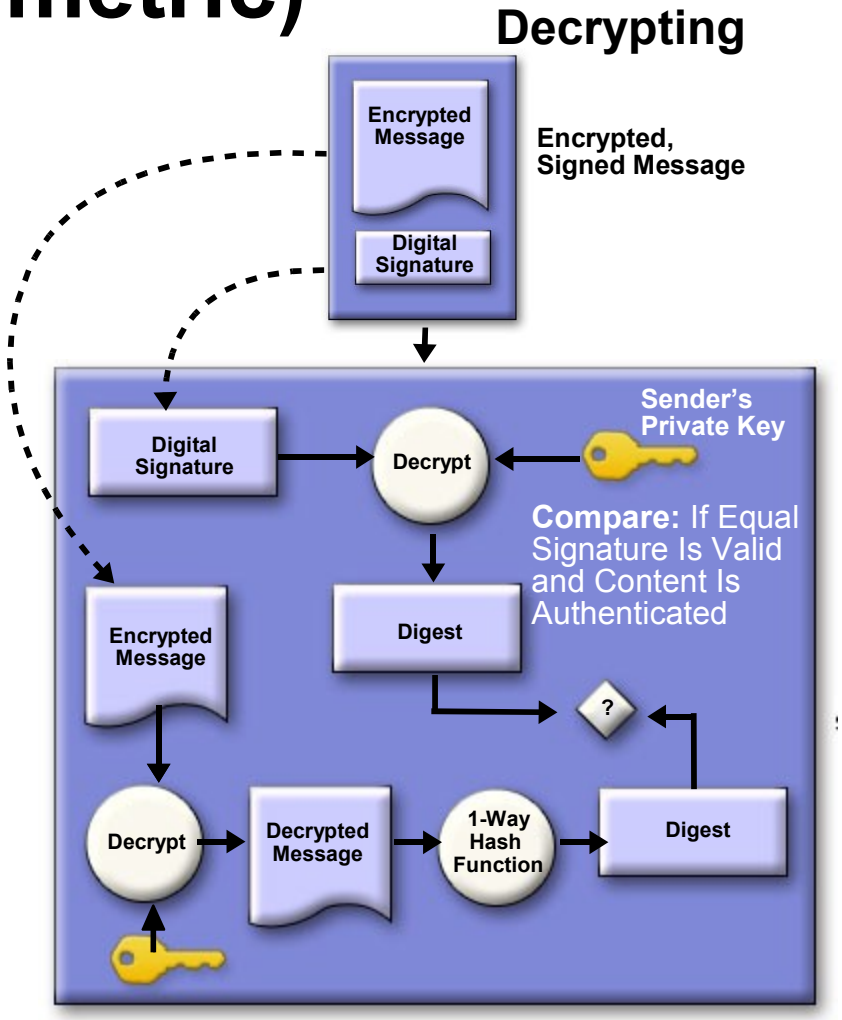
```
CMSMessageSignatureService.sign(...)
```

# Verifying a Digital Signature

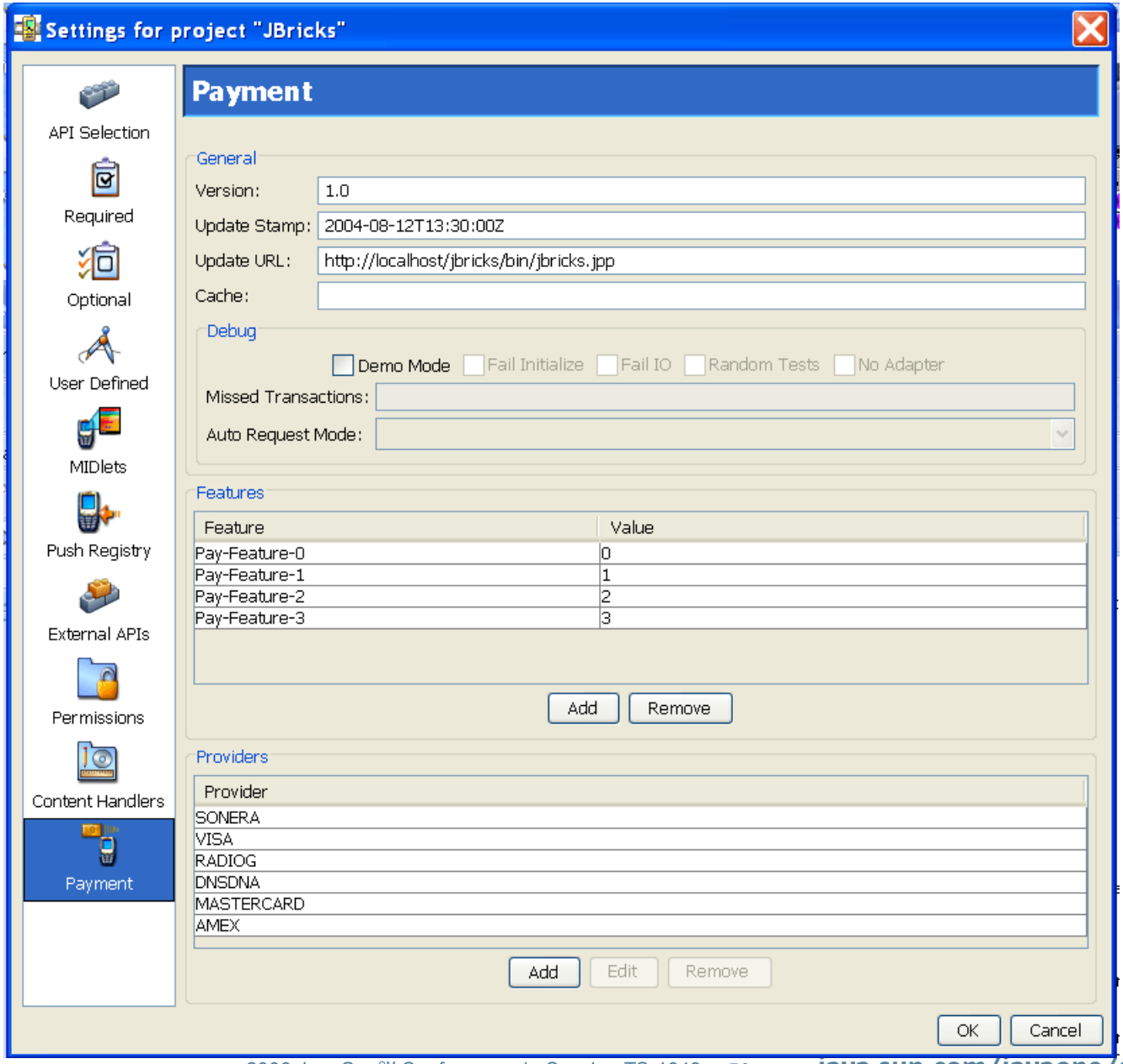
```
byte[] signedMessage = "..."; // sent by sender
byte[] messageSignature = "..."; // sent by sender
String sendersPublicKeyAlgo = "RSA";
byte[] sendersEncodedPublicKey = "..."; // sent by sender
// Create X.509 encoded Key from encoded public key.
X509EncodedKeySpec pks = new
                X509EncodedKeySpec(sendersEncodedPublicKey);
try {
  KeyFactory kf;
  kf = KeyFactory.getInstance(sendersPublicKeyAlgo);
  PublicKey sendersPublicKey = kf.generatePublic(pks);
  Signature signature;
  signature = Signature.getInstance
                        (sendersPublicKey.getAlgorithm());
  signature.initVerify(sendersPublicKey);
  signature.update(signedMessage, 0, signedMessage.length);
  boolean signatureValid;
  signatureValid = signature.verify(messageSignature);
  if (signatureValid = false){/*Signature didn't verify*/}
} catch (Exception e) {...}
```

# Signed Messaging Using Public Key Cryptography (Asymmetric)

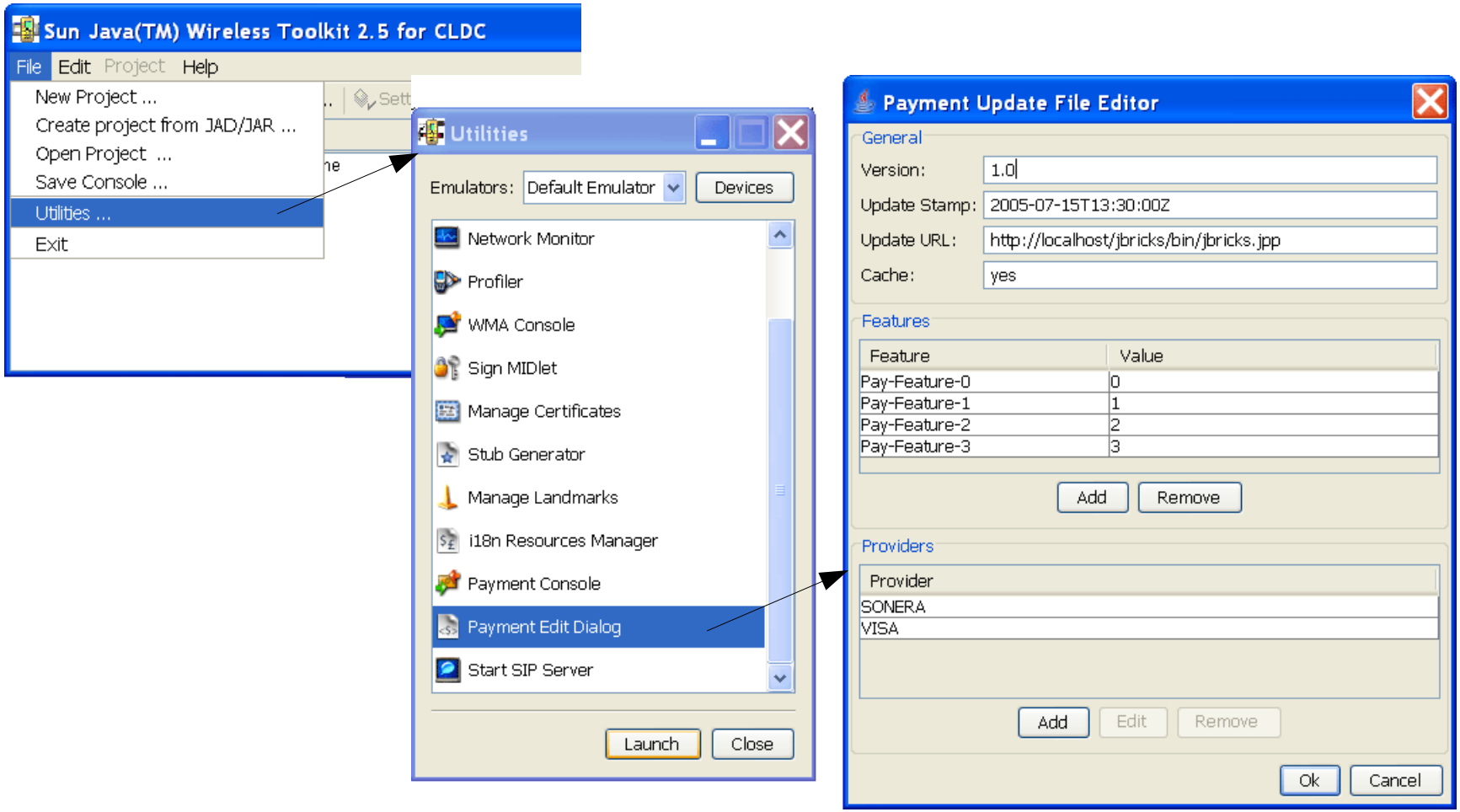


# Project Setting for Payment



# Utilities


Sun Java(TM) Wireless Toolkit 2.5 for CLDC
File Edit Project Help
New Project ...
Create project from JAD/JAR ...
Open Project ...
Save Console ...
Utilities ...
Exit
Utilities
Emulators: Default Emulator
Devices
Network Monitor
Profiler
WMA Console
Sign MIDlet
Manage Certificates
Stub Generator
Manage Landmarks
i18n Resources Manager
Payment Console
Payment Edit Dialog
Start SIP Server
Launch
Close
Payment Update File Editor
General
Version: 1.0
Update Stamp: 2005-07-15T13:30:00Z
Update URL: http://localhost/jbricks/bin/jbricks.jpp
Cache: yes
Features
Feature
Value
Pay-Feature-0
0
Pay-Feature-1
1
Pay-Feature-2
2
Pay-Feature-3
3
Add
Remove
Providers
Provider
SONERA
VISA
Add
Edit
Remove
Ok
Cancel

# Payment Transactions