CMER

Centre for Mobile Education and Research

Security APIs for Mobile Devices

Week II

Overview

- SATSA
- Smart Card Communications
- Digital Signatures
- General Purpose Cryptographic Library
- MIDP
- Trusted/Untrusted MIDlet Suites
- Permissions
- Protection Domains
- Signing a MIDlet
- Creating the Signing Certificate
- Verifying Certificate
- Certification Expiration and Revocation

APIs

- There are various APIs available that deal with security, trust, and privacy.
- The following below deal with mobile devices:
  - Security and Trust Services API (SATSA) for J2ME
  - Mobile Information Device Profile (MIDP) for J2ME

SATSA

- Security and Trust Services API (SATSA) defines an API that provides various abilities for J2ME developed applications that may communication with various security related elements.
- It focuses on three specific areas
  - Smart Card Communication
  - Digital Signatures
  - General Purpose Cryptography Library

Smart Card Communications (Cont.)

- Based on the Generic Connection Framework (GCF), which is defined in the CLDC 1.0 specification
- Provide a secure programming environment
- Due to a smart card's ability to provide a wide range of security and trust services, they are the most commonly used security element.

Smart Card Communications (Cont.)

- It’s services can be frequently upgraded with new or better applications that can be installed on a smart card
- Two access methods are defined that allow a J2ME application to communicate with a smart card to control the security services deployed on it
  - ADPU protocol
  - Java Card RMI protocol

Digital Signatures

- Allows a J2ME application to generate digital signatures
- Used to authenticate end-users or to authorize transactions using public key cryptography.
- User’s identity is usually bound to the public key through a public key certificate.

Digital Signatures (Cont.)

- User credential management allows a J2ME application to manage user credentials on a user’s behalf.
  - i.e. certificates
- Digital signature service and credential management rely on a security element to provide secure storage of user credentials and cryptographic keys

Digital Signatures (Cont.)

- The security element is also responsible for implementing secure computation involving the cryptographic keys that are securely stored on the security element.

General Purpose Cryptography Library

- This library provides a similar version of the J2security element cryptography API.
- It supports basic cryptographic operations, such as message digest, digital signature verification (but not signing), encryption, and decryption.

General Purpose Cryptography Library (Cont.)

- Cryptographic operations allow a J2ME application to provide secure data communication, data protection, and content management.
- The subset is determined by including the minimum number of classes and methods from the J2security element API to address the use cases that are not targeted by other API in this specification.

General Purpose Cryptography Library (Cont.)

- Another criterion to select the subset is to expose the cryptographic functions that are already supported by many devices. To minimize the footprint, the API in the subset supports only a default Cryptographic Service Provider.

API Packages

- There are four packages that are a subset of this API
  - SATSA-APDU
    - defines an API to support communication with smart card applications using the APDU protocol.
  - SATSA-JCRMI
    - defines a Java Card RMI client API that allows a J2ME application to invoke a method of a remote Java Card object.

API Packages (Cont.)

- SATSA-PKI
  - defines an API to support application level digital signature signing (but not verification) and basic user credential management. To enable broader reuse, this API is independent of the type of security elements that are utilized by a J2ME device.
- SATSA-CRYPTO
  - defines a subset of the J2security element cryptography API. It provides basic cryptographic operations to support message digest, signature verification, encryption, and decryption.

API Packages (Cont.)

- SATSA-APDU and SATSA-JCRMI packages can be implemented on a J2ME device which does not have the smart card slots.
  - Such implementation of these optional packages is called the null implementation

SATSA-APDU

- Includes two components to support communication with smart cards using the APDU protocol
  - A subset of the java.lang package that supports the exception class UnsupportedOperationException
  - The javax.microedition.apdu package
    - Contains the interface APDUConnection to support APDU exchanges.

SATSA-APDU (Cont.)

- Allows a J2ME application to create a APDUConnection that can communicate with a smart card application identified by an Application Identifier.
- Each APDUConnection has a logical channel reserved exclusively for it.
  - Logical channel management is handled by the API implementation, which requests the smart card to allocate an unused logical channel.

SATSA-APDU (Cont.)

- More than one APDUConnection can be created to communicate simultaneously with smart card applications on one (via logical channels) or multiple smart cards.
- An implementation may support up to 20 logical channels for communication with the smart card.

SATSA-APDU (Cont.)

- An APDUConnection can be created to communicate with SIM application toolkit applications on channel 0 of an IC card or a SIM card.
- The APDUConnection has limited capabilities when communicating with a SIM application toolkit application.

SATSA-APDU (Cont.)

- Only ENVELOPE APDUs may be sent by J2ME applications to trigger a SIM application toolkit application.
- Proactive sessions and commands are not supported by the APDUConnection.

SATSA-APDU (Cont.)

- J2ME application is responsibility for making sure that it must not send an envelope to the SIM application toolkit application that would result in a proactive session being initiated via the APDUConnection interface.

SATSA-JCRMI

- Contains four components
  - A subset of the java.lang package that supports the exception class UnsupportedOperationException
  - A subset of the java.rmi package that provides the basic RMI client interfaces.
  - The javax.microedition.jcrmi package
    - Contains the JavaCardRMIConnection interface used to initiate a JCRMI session and interfaces used by the stubs generated by the JCRMI stub compiler

SATSA-JCRMI (Cont.)

– A subset of the Java Card API. The subset includes the Exception classes defined in the packages javacard.framework, javacard.framework.service, and javacard.security
  - These exceptions may be thrown during a method invocation of a Java Card object because of cryptographic errors (the javacard.security package), card framework access errors (the javacard.framework package) or errors detected in accessing the card services (the javacard.framework.service package)

SATSA-JCRMI (Cont.)

- This package is used to create a JavaCardRMIConnection to initialize and initiate a Java Card RMI session with a Java Card applet.
- A JavaCardRMIConnection allows a J2ME application to access the initial remote reference.
  - This then allows the application to invoke methods of the remote object on the card and obtain the references to other remote objects.

SATSA-JCRMI (Cont.)

- Each JavaCardRMIConnection has a logical channel reserved exclusively for it.
  - Logical channel management is handled by the API implementation, which requests the smart card to allocate an unused logical channel.
- More than one JavaCardRMIConnection can be created to communicate simultaneously with Java Card applets on one or multiple smart cards.

SATSA-JCRMI (Cont.)

- An implementation may support up to 20 logical channels for communication with the smart card.
- Platforms that do not support the standard J2security element RMI, must have a static stub for each remote object generated off the device and downloaded together with a J2ME application.

SATSA-JCRMI (Cont.)

- The stub interfaces that a JCRMI compiler must use to create the appropriate stubs are defined in this optional package

SATSA-PKI

- Contains two components
  - The javax.microedition.pki package
    - supports generation of certificate requests and local registration of the user credentials.
    - user credentials are used in conjunction with other parameters to compute formatted digital signatures
  - The javax.microedition.securityservice package.
    - It supports generation of application-level digital signatures that conform to Cryptographic Message Syntax format

SATSA-PKI (Cont.)

- This package is provided to generate digital signatures and basic user credential management
  - Eg, an X.509 certificate is a user credential that includes a public key
- Allows the use of CMSMessageSignatureService to sign messages with a private key.
- Messages may be signed for authentication or non-repudiation.

SATSA-PKI (Cont.)

- Authorization of the use of a key in a security element is governed by the policy of the security element, for example, PIN entry required.
- A J2ME application uses UserCredentialManager to perform the following tasks:
  – **Formulate a certificate enrollment request, which may be sent to a certificate registration authority.**

SATSA-PKI (Cont.)

– **Add a certificate or a certificate URI to a certificate store.**
– **Remove a certificate or a certificate URI from a certificate store.**

SATSA-CRYPTO

- Contains three components
  - The java.security and java.security.spec packages.
    - provide the basic support for accessing public keys, computing digests, and verifying digital signatures
  - The javax.crypto and javax.crypto.spec packages.
    - provide the basic support for encryption and decryption of data

SATSA-CRYPTO (Cont.)

- A subset of the java.lang package
  - Contains the exception class IllegalStateException

- A subset of the J2SE cryptography API
- Allows the use of the MessageDigest class to access the functionality of a message digest algorithm.
- Allows use the Signature class to access the functionality of a digital signature algorithm for verifying a digital signature.

SATSA-CRYPTO (Cont.)

- A J2ME application uses the Cipher class to access the functionality of a cryptographic cipher for encryption and decryption.
- The SATSA-CRYPTO package does not include an API to create a private key object.
  - Asymmetric cipher using a private key is not supported.

SATSA-CRYPTO (Cont.)

- The KeyFactory class can be used to build an opaque public key object from a given key specification (transparent representations of the underlying key material).

SATSA-CRYPTO (Cont.)

- A J2ME application uses the SecretKeySpec class to construct an opaque secret key object from the key material that can be represented as a byte array and have no key parameters associated with them.

Security

- A J2ME application must be granted with a permission to use the privileged API in SATSA-APDU, SATSA-JCRMI, and SATSA-PKI packages.
- Permissions are checked by the platform prior to the invocation of the protected methods in the API.

Security (Cont.)

- Based on the security framework implemented by the underlying platform, an implementation of a SATSA package must support either the MIDP 2.0 permissions applicable to that package or the functional equivalent J2security element style permission classes.

Security (Cont.)

- Along with protecting the usage of the resources in a security element, a recommended access control model is defined that allows the security element to specify access control policies.

Security (Cont.)

- Within the scope of this specification, the access control model is intended for the API in the SATSA-APDU and SATSA-JCRMI packages, when it is implemented on GSM/UMTS-complaint devices.

Security (Cont.)

- Based on the access control policy defined in a smart card, the device determines whether the J2ME application is allowed to access any function of the smart card, using theAPDUConnection or the JavaCardRMIConnection.

MIDP

- MIDP v2.0 stands for Mobile Information Device Profile for Java ME.
- This specification was created to define an enhanced architecture and the associated APIs required to enable an open, third-party, application development environment for mobile information devices, or MIDs

MIDP (Cont.)

- Designed to operate on top of the Connected, Limited Device Configuration (CLDC)
- Deals with the following APIs
  - Application delivery and billing
  - Application lifecycle (i.e., defining the semantics of a MIDP application and how it is controlled)
  - Application signing model and privileged domains security model

MIDP (Cont.)

- End-to-end transactional security (https)
- MIDlet push registration (server push model)
- Networking
- Persistent storage
- Sound
- Timers
- User interface (UI) (including display and input, as well as the unique requirements for games).

Responsibility

- MIDP is not responsible for
  - System-level APIs:
    - MIDP is intented for enabling application programmers, rather than enabling system programming.
    - low-level APIs that specify a system interface to, for example, a MID's power management or voice CODECs are beyond the scope of this specification.
  - Low-level security:
    - The MIDP specifies no additional low-level security features other than those provided by the CLDC.

Requirements

- In order to use MIDP the following criteria must be met.
  - MUST support MIDP 1.0 and MIDP 2.0 MIDlets and MIDlet Suites.
  - MUST include all packages, classes, and interfaces described in this specification.
  - MUST implement the OTA User Initiated Provisioning specification.

Requirements (Cont.)

- MUST give the user a visual indication of network usage generated when using the mechanisms indicated in this specification.
- MUST provide support for accessing HTTP 1.1 servers and services either directly, or by using gateway services such as provided by WAP or i-mode.
- MUST provide support for secure HTTP connections either directly, or by using gateway services such as provided by WAP or i-mode.

Requirements (Cont.)

- MUST support PNG image transparency.
- MUST support ISO/IEC JPEG together with JFIF.
- MUST support Tone Generation in the media package.
- MUST support 8-bit, 8 KHz, mono linear PCM wav format IF any sampled sound support is provided.

Requirements (Cont.)

- **MUST support Scalable Polyphony MIDI (SP-MIDI) and SP-MIDI Device 5-to-24 Note Profile IF any synthetic sound support is provided.**
- **MUST implement the mechanisms needed to support "Untrusted MIDlet Suites".**
- **MUST implement "Trusted MIDlet Suite Security" unless the device security policy does not permit or support trusted applications.**

Requirements (Cont.)

- MUST support Scalable Polyphony MIDI (SP-MIDI) and SP-MIDI Device 5-to-24 Note Profile IF any synthetic sound support is provided.
- MUST implement the mechanisms needed to support "Untrusted MIDlet Suites".
- MUST implement "Trusted MIDlet Suite Security" unless the device security policy does not permit or support trusted applications.

Requirements (Cont.)

- MUST implement "Trusted MIDlet Suites Using X.509 PKI" to recognize signed MIDlet suites as trusted unless PKI is not used by the device for signing applications.
- MUST implement "MIDP x.509 Certificate Profile" for certificate handling of HTTPS and SecureConnections.

Requirements (Cont.)

- MUST enforce the same security requirements for I/O access from the Media API as from the Generic Connection framework, as specified in the package documentation for javax.microedition.io.
- MUST support at least the UTF-8 character encoding for APIs that allow the application to define character encodings.
- SHOULD NOT allow copies to be made of any MIDlet suite unless the device implements a copy protection mechanism.

Requirements (Cont.)

- SHOULD provide support for datagram connections.
- SHOULD provide support for server socket stream connections.
- SHOULD provide support for socket stream connections.
- SHOULD provide support for secure socket stream connections.

Requirements (Cont.)

- MAY support other character encodings.
- MAY incorporate zero or more supported protocols for push.
- MAY provide support for accessing any available serial ports on their devices through the CommConnection interface.
- MAY include support for additional sampled sound formats.
- MAY include support for additional MIDI formats.
- MAY include support for additional image formats.

Security - MIDP

- MIDP 1.0
  - constrains each MIDlet suite to operate in a sandbox wherein all of the APIs available to the MIDlets would prevent access to sensitive APIs or functions of the device.
  - This concept is used in this specification and all untrusted MIDlet suites are subject to its limitations. Every implementation of this specification MUST support running untrusted MIDlet suites.

Security – MIDP (Cont.)

- MIDP 2.0
  - launched the concept of trusted applications that may be permitted to use APIs that are considered sensitive and are restricted.
  - If and when a device determines that a MIDlet suite can be trusted then access is allowed as indicated by the domain policy.
  - devices are expected to operate using standard Internet and wireless protocols and techniques for transport and security.

Security – MIDP (Cont.)

- The current mechanisms for securing Internet content is based on existing Internet standards for public key cryptography:
  - [RFC2437] - PKCS #1 RSA Encryption Version 2.0
  - [RFC2459] - Internet X.509 Public Key Infrastructure
  - [RFC2560] - Online Certificate Status Protocol
  - [WAPCERT] - WAP-211-WAPCert-20010522-a - WAP Certificate Profile Specification

Trusted or Untrusted MIDlet suite?

- Any MIDlet suite that is not trusted by the device MUST be run as untrusted.
- If errors occur in the process of verifying that a MIDlet suite is trusted then the MIDlet suite MUST be rejected.

Untrusted MIDlet Suites

- An untrusted MIDlet suite is a MIDlet suite for which the origin and the integrity of the JAR file can NOT be trusted by the device.
- Untrusted MIDlet suites MUST execute in the untrusted domain using a restricted environment where access to protected APIs or functions either is not allowed or is allowed with explicit user permission.

Untrusted MIDlet Suites (Cont.)

- Any MIDP 1.0 compliant MIDlet suite MUST be able to run in an implementation of this specification as untrusted.
- Any APIs or functions of this specification which are not security sensitive, having no permissions defined for them, are implicitly accessible by both trusted and untrusted MIDlet suites.

Untrusted MIDlet Suites (Cont.)

- Untrusted MIDlet suites do not request permissions explicitly in the JAR manifest or application descriptor.
- Allow access to the following APIs WITHOUT explicit confirmation from the user
  - RMS APIs
    - javax.microedition.rms
  - MIDlet Lifecycle APIs
    - javax.microedition.midlet

Untrusted MIDlet Suites (Cont.)

- User Interface APIs
  - javax.microedition.lcdui
- The Game APIs
  - javax.microedition.lcdui.game
- The multi-media APIs for playback of sound
  - javax.microedition.media
  - javax.microedition.media.control

Untrusted MIDlet Suites (Cont.)

- Allow access to the following protected APIs or functions WITH explicit confirmation from the user
  - http
    - javax.microedition.io.HttpConnection
  - https
    - javax.microedition.io.HttpsConnection

Trusted MIDlet Suite Security

- A MIDlet suite for which the authentication and the integrity of JAR file can be trusted by the device and bound to a protection domain
- Based on protection domains.
  - Protected domain is a set of Allowed and User permissions that may be granted to a MIDlet suite

Trusted MIDlet Suite Security (Cont.)

- Each protection domain defines the permissions that may be granted to a MIDlet suite in that domain.
- The protection domain owner specifies how the device identifies and verifies that it can trust a MIDlet suite and bind it to a protection domain with the permissions that authorize access to protected APIs or functions.

Trusted MIDlet Suite Security (Cont.)

- Using X.509 PKI, the trusted MIDlet Suites describes a mechanism for identifying trusted MIDlet suites though signing and verification.
- If an implementation of this specification will recognize MIDlet suites signed using PKI as trusted MIDlet suites they must be signed and verified according to the formats and processes specified in Trusted MIDlet Using X.509 PKI.

Permissions

- Permissions are the means to protect access to APIs or functions which require explicit authorization before being invoked.
- Permissions are checked by the implementation prior to the invocation of the protected function.
- The names of permissions have a hierarchical organization similar to Java package names.

Permissions (Cont.)

- The names of permissions are case sensitive.
- All of the permissions for an API MUST use the prefix that is the same as the package name of the API.
- If the permission is for a function of a specific class in the package then the permission MUST include the package and classname.

Permissions (Cont.)

- Each API in this specification that provides access to a protected function will define the permissions.
- For APIs defined outside of MIDP 2.0 there must be a single document that specifies any necessary permissions and the behavior of the API when it is implemented on MIDP 2.0.

Permissions for Protected Functions

- Each function (or entire API) which was identified as protected must have its permission name defined in the class or package documentation for the API.
- Refer to the documentation of the javax.microedition.io package for permissions on all Generic Connection schemes defined in this specification.

Permissions for Protected Functions (Cont.)

- All APIs and functions within this specification that do not explicitly define permissions MUST be made available to all trusted and untrusted MIDlet suites.

Requesting Permissions for a MIDlet Suite

- A MIDlet suite that requires access to protected APIs or functions must request the corresponding permissions.
- Permissions requested can be required by listing the permissions in the attribute MIDlet-Permissions.
- These permissions are critical to the function of the MIDlet suite and it will not operate correctly without them.

Requesting Permissions for a MIDlet Suite (Cont.)

- If the MIDlet suite can function correctly with or without particular permission(s) it should request them using the MIDlet-Permissions-Opt attribute.
- The MIDlet suite is able to run with reduced functionality (for example, as a single player game instead of a net game) without these non-critical permissions and MUST be installed and run.

Requesting Permissions for a MIDlet Suite (Cont.)

- The MIDlet-Permissions and MIDlet-Permissions-Opt attributes contain a list of one or more permissions.
- Multiple permissions are separated by a comma.
- Leading/trailing whitespace and tabs are ignored.

Permissions on the Device

- Each device that implements this specification and any other Java APIs will have a total set of permissions referring to protected APIs and functions.

Protection Domains

- **A protection domain defines a set of permissions and related interaction modes.**
- **A protection domain consists of:**
  - **a set of permissions that should be allowed (*Allowed)***
  - **a set of permissions that the user may authorize (*User) each with its user interaction mode***

Protection Domains (Cont.)

- **Within a protection domain each permission may be either *allowed or user* _*but NOT both*_*.***

Allowed Permission

- The *Allowed* permissions are any permissions which explicitly allow access to a given protected API or function on the basis of MIDlet suite being associated with the protection domain.
- *Allowed* permissions do not require any user interaction.

User Permission

- The *User* permissions are any permissions for a protected API or function on the basis of MIDlet suite being bound to the protection domain and will allow access to protected API or function after the prompt given to the user and explicit user permission being granted.

User Permission Interaction Modes

- A User Permission is defined to allow the user to deny permission or to grant permission to a specific API with one of the following interaction modes:
  - blanket
    - valid for every invocation of an API by a MIDlet suite until it is uninstalled or the permission is changed by the user.

User Permission Interaction Modes (Cont.)

- Session
  - valid from the invocation of a MIDlet suite until it terminates. "session" mode MUST prompt the user on or before the first invocation of the API or function which is protected. When the user re-invokes the MIDlet suite the prompt MUST be repeated.
- Oneshot
  - MUST prompt the user on each invocation of the API or function which is protected.

User Permission Interaction Modes (Cont.)

- The choice of user permission interaction modes is driven by the security policy and the device implementation.
- Each user permission has a default interaction mode and a set of other available interaction modes.
- The user SHOULD be presented with a choice of interaction modes.

User Permission Interaction Modes (Cont.)

- The default interaction mode may be offered if it is supplied.
- The user MUST always be able to deny permission.
- If and when prompted, the user SHOULD be provided with a user friendly description of the requested permissions sufficient to make a well-informed choice.

User Permission Interaction Modes (Cont.)

- The range of blanket to one shot action permission modes represents a tradeoff between usability and user notification and should behave smoothly and consistently with the human interface of the device.

Granting Permissions to Trusted MIDlet Suites

- **Authorization of trusted MIDlet suites uses protection domain information, permissions on the device, and permissions requested in the MIDlet suite.**
- **Permissions in the domain are *Allowed* or *User.***
- **Permissions requested by the application are either critical or non-critical.**

Granting Permissions to Trusted MIDlet Suites (Cont.)

- To establish the permissions granted to a trusted MIDlet suite when it is to be invoked, all of the following MUST be true:
  - The MIDlet suite must have been bound to a protection domain.
  - The requested critical permissions are retrieved from the attributes MIDlet-Permissions and non-critical permissions from MIDlet-Permissions-Opt.

Granting Permissions to Trusted MIDlet Suites (Cont.)

- If these attributes appear in the application descriptor they MUST be identical to corresponding attributes in the manifest. If they are not identical, the MIDlet suite MUST NOT be installed or invoked.
- If any of the requested permissions are unknown to the device and are not marked as critical then they are removed from the requested permissions.

Granting Permissions to Trusted MIDlet Suites (Cont.)

- If any of the requested permissions are unknown to the device and marked as critical, the MIDlet suite MUST NOT be installed or invoked.
- If any of the requested permissions are not present in the protection domain (*Allowed or User)* permission sets and the requested permission was marked as critical then the MIDlet suite does not have sufficient authorization and MUST NOT be installed or invoked.

Granting Permissions to Trusted MIDlet Suites (Cont.)

– **If any of the requested permissions are not present in the protection domain (*Allowed or User)* permission sets, and the requested permissions are not marked as critical, the application MUST still be installed and MUST be able to be invoked by the user.**
– **If any of the requested permissions match the *User* permissions of the protection domain then the user MUST explicitly provide authorization to grant those permissions to the MIDlet suite.**

Granting Permissions to Trusted MIDlet Suites (Cont.)

  - The implementation is responsible for making the request to the user and getting the response to allow or deny the request.
- – During execution, any protected APIs MUST check for the appropriate permissions and throw a SecurityException if the permission has not been granted.

- The successful result of authorization is that the MIDlet suite is granted access to protected APIs or functions for which it requested permissions.

Trusted MIDlet Suites using X.509 PKI

- Signed MIDlet suites may become trusted by authenticating the signer of the MIDlet suite and binding it to a protection domain that will authorize the MIDlet suite to perform protected functions by granting permissions allowed in the protection domain.

Trusted MIDlet Suites using X.509 PKI (Cont.)

- This allows signing and authentication of MIDlet suites based on X.509 Public Key Infrastructure so the device can verify the signer and trust the MIDlet suite.
- If an implementation of this specification will recognize MIDlet suites signed using PKI as trusted MIDlet suites they MUST be signed and verified according to the formats and processes below.

Trusted MIDlet Suites using X.509 PKI (Cont.)

- The MIDlet suite is protected by signing the JAR.
  - The signature and certificates are added to the application descriptor as attributes.
  - The device uses them to verify the signature.
  - The device completes the authentication using a root certificate bound to a protection domain on the device.

Signing a MIDlet Suite

- Zero or more root certificates will need to be on the device. Additionally, root certificates may be present in removable media such as SIM(WIM) card/USIM module.
- Implementations MUST support X.509 Certificates and corresponding algorithms.
- Devices MAY support additional signing mechanisms and certificate formats.

Signing a MIDlet Suite (Cont.)

- The signer of the MIDlet suite may be the developer or some entity that is responsible for distributing, supporting, and perhaps billing for its use.
- The signer will need to have a public key certificate that can be validated to one of the protection domain root certificates on the device.
- The public key is used to verify the signature on the MIDlet suite.

Signing a MIDlet Suite (Cont.)

- The public key is provided as a RSA X.509 certificate included in the application descriptor.
- Attributes defined within the manifest of the JAR are protected by the signature.
- Attributes defined within the application descriptor are not secured.
  - When an attribute appears in the manifest it *MUST NOT be overridden* by a different value from the application descriptor.

Signing a MIDlet Suite (Cont.)

- For trusted MIDlet suites the value in the application descriptor must be equal to the value of the corresponding attribute in the manifest. If not, the MIDlet suite MUST NOT be installed.
- The MIDlet.getAppProperty method must return the attribute value from the manifest if one is defined. If not, the value from the application descriptor (if any) is returned.

Signing a MIDlet Suite (Cont.)

- Note that the requirement that attributes values be equal differs from MIDP 1.0 and must be used for applications that are signed and verified by these procedures.
- For untrusted application descriptors, the MIDP 1.0 rule giving priority to application descriptor attributes over manifest attributes must be followed.

Signing a MIDlet Suite (Cont.)

- The signer of the MIDlet suite is responsible to its protection domain root certificate owner for protecting the protection domain stake holder's assets and capabilities and, as such, must exercise due-diligence in checking the MIDlet suite before signing it.

Signing a MIDlet Suite (Cont.)

- In the case where there is a trusted relationship (possibly bound by legal agreements), a protection domain root certificate owner may delegate signing MIDlet suites to a third-party and in some circumstances, the author of the MIDlet.

Creating the Signing Certificate

- The signer will need to be aware of the authorization policy for the device and contact the appropriate certificate authority.
  - For example, the signer may need to send its distinguished name (DN) and public key (normally, packaged in a certificate request) to a certificate authority.
- The CA creates a RSA X.509 (version 3) certificate and returns it to the signer.

Creating the Signing Certificate (Cont.)

- If multiple CA's are used then all the signer certificates in the application descriptor MUST contain the same public key.

Creating the RSA SHA-1 signature of the JAR

- The signature of the JAR is created with the signers private key according to the EMSA-PKCS1-v1_5 encoding method of PKCS #1 version 2.0 standard[RFC2437].
- The signature is base64 encoded, formatted as a single MIDlet-Jar-RSA-SHA1 attribute without line breaks and inserted in the application descriptor.
  - **MIDlet-Jar-RSA-SHA1: <base64 encoding of Jar signature>**

Authenticating a MIDlet Suite

- When an MIDlet suite is downloaded, the device MUST check if authentication is required.
- If the attribute MIDlet-Jar-RSA-SHA1 is present in the application descriptor then the JAR MUST be authenticated by verifying the signer certificates and JAR signature as below.

Authenticating a MIDlet Suite (Cont.)

- Application descriptors without the MIDlet-Jar-RSA-SHA1 attribute are not authenticated but are installed and invoked as untrusted MIDlet suites.

Verify Signer Certificate

- The certification path consists of the signer certificate from the application descriptor and other certificates as needed up to but not including the root certificate.
- Get the certification path for the signer certificate from the application descriptor attributes MIDlet-Certificate-1-<m> where <m> starts at 1 and is incremented by 1 until there is no attribute with the given name.

Verify Signer Certificate (Cont.)

  - The value of each attribute is a base64 encoded certificate that will need to be decoded and parsed.
- Validate the certification path using the basic path validation processes described using the protection domains as the authoritative source of protection domain root certificates.

Verify Signer Certificate (Cont.)

- Bind the MIDlet suite to the protection domain that contains the protection domain root certificate that validates the first chain from signer to root and proceed with installation.

Verify Signer Certificate (Cont.)

- If attributes MIDlet-Certificate-<n>-<m> with <n> greater than 1 are present and full certification path could not be established after verifying MIDlet-Certificate-<1>-<m> certificates, repeatedly perform prior steps for the value <n> greater by 1 than the previous value.

Actions upon completion of signer certificate verification

- Attempted to validate <n> paths. No public keys of the issuer for the certificate can be found or none of the certification paths can be validated
  - **Authentication fails, JAR Installation is not allowed.**
- More than one full certificate path established and validated

Actions upon completion of signer certificate verification (Cont.)

  - Implementation proceeds with the signature verification using the first successfully verified certificate path is used for authentication and authorization.
- Only one full certificate path established and validated
  - Implementation proceeds with the signature verification

Verify the MIDlet Suite JAR

1. Get the public key from the verified signer certificate (above).
2. Get the MIDlet-Jar-RSA-SHA1 attribute from the application descriptor.
3. Decode the attribute value from base64 yielding a PKCS #1 signature.
4. Use the signer's public key, signature, and SHA-1 digest of the JAR, to verify the signature.

Verify the MIDlet Suite JAR (Cont.)

  - If the signature verification fails, reject the application descriptor and MIDlet suite. The implementation MUST NOT install the JAR on failure or allow MIDlets from the MIDlet suite to be invoked.
- Once the steps of verifying the certificate, verifying the signature and verifying the JAR all succeed then the MIDlet suite contents are known to be intact and the identity of the signer is known.

Verify the MIDlet Suite JAR (Cont.)

- This process must be performed during installation.
- It is essential that steps be performed to verify the digital signature in order to identify the MIDlet suite signer.
- The results of the verification have a direct impact on authorization.

Summary of Verification

- The following present a summary of the state and result
- JAD not present, JAR downloaded
  - **Authentication can not be performed, may install JAR. MIDlet suite is treated as untrusted**
- JAD present but is JAR is unsigned
  - **Authentication can not be performed, may install JAR. MIDlet suite is treated as untrusted**

Summary of Verification (Cont.)

- JAR signed but no root certificate present in the keystore to validate the certificate chain
  - Authentication can not be performed, JAR installation is not allowed
- JAR signed, a certificate on the path is expired
  - Authentication can not be completed, JAR installation is not allowed

Summary of Verification (Cont.)

- JAR signed, a certificate rejected for reasons other than expiration
  - JAD rejected, JAR installation is not allowed
- JAR signed, certificate path validated but signature verification fails
  - JAD rejected, JAR installation is not allowed
- JAR signed, certificate path validated, signature verified
  - JAR installation is allowed

Caching of Authentication and Authorization Results

- The implementation of the authentication and authorization process may store and transfer the results for subsequent use and MUST ensure that the cached information can not be tampered with or otherwise compromised between the time it is computed from the JAR, application descriptor, and authentication information and the authorization information is used.

Caching of Authentication and Authorization Results (Cont.)

- It is essential that the MIDlet suite and security information used to authenticate and authorize a MIDlet suite is not compromised.
  - for example, by use of removable media or other access to MIDlet suite storage that might be corrupted.

MIDP X.509 Certificate Profile for Trusted MIDlet Suites

- Secured trusted MIDlet suites utilize the same base certificate profile as does HTTPS.
- The profile is based on the WAP Certificate Profile [WAPCert] which is based on RFC2459 Internet X.509 Public Key Infrastructure Certificate and CRL Profile [RFC2459].

Certificate Processing for OTA

- Devices MUST recognize the key usage extension and when present verify that the extension has the digitalSignature bit set.
- Devices MUST recognize the critical extended key usage extension and when present verify that the extension contains the id-kp-codeSigning object identifier
- The application descriptor SHOULD NOT include a self-issued root certificate in a descriptor certificate chain.

Certificate Processing for OTA (Cont.)

- MIDP devices SHOULD treat the certificate as any other in a chain and NOT explicitly reject a chain with a X.509v3 self-issued CA certificate in its chain.

Certificate Expiration and Revocation

- Expiration and revocation of certificates supplied in the application descriptor is checked during the authorization procedure, specifically during certificate path validation.
- Certificate expiration is checked locally on the device as such information is retrievable from the certificate itself.

Certificate Expiration and Revocation (Cont.)

- Certificate expiration verification is a mandatory part of certificate path validation.
- Certificate revocation is a more complex check as it requires sending a request to a server and the decision is made based on the received response.
- Certificate revocation can be performed if the appropriate mechanism is implemented on the device.

Certificate Expiration and Revocation (Cont.)

- Such mechanisms are not part of MIDP implementation and hence do not form a part of MIDP 2.0 security framework.
- If certificate revocation is implemented in the device, it SHOULD support Online Certificate Status protocol (OCSP).

Certificate Expiration and Revocation (Cont.)

- If other certificate revocation protocols are supported, support for these other protocols may indicate that a certificate has been revoked; in this case, it is permissible to consider the certificate as revoked regardless of the result returned by the OCSP protocol.

Examples

- Developer Owns Signing Certificate
- Protection Domain Stakeholder Owns Signing Certificate

Developer Owns Signing Certificate

- This encodes the origin of the MIDlet suite into the JAD (via the identity of the signer). If the certificate is revoked, all of the developer's signed MIDlets on every device for every user will have their execution permissions revoked.

1. Developer creates MIDlet network application

Developer Owns Signing Certificate (Cont.)

2. Developer encodes permissions into JAR manifest and creates final MIDlet JAR
3. Developer generates a private-public key pair with a signing certificate and has the certificate signed by one or more protection domain root certificates
4. The developer's certificate is used to sign the MIDlet JAR and create the associated JAD entries

Developer Owns Signing Certificate (Cont.)

5. MIDlet JAR can be distributed with a suitably populated JAD and run on a MIDP 2.0 compliant device with the appropriate protection domain root certificate

Protection Domain Stakeholder Owns Signing Certificate

- This encodes the signers identity (not the MIDlet suite developer) into the JAD. If the certificate is revoked, all MIDlets signed with this particular certificate will have their execution permissions revoked.

1. Developer creates MIDlet network application
2. Developer encodes permissions into JAR manifest and creates final MIDlet JAR

Protection Domain Stakeholder Owns Signing Certificate (Cont.)

3. The protection domain stakeholder's signing certificate (not necessarily the root cert) is used to sign the MIDlet JAR and create the associated JAD entries
4. MIDlet JAR can be distributed with a suitably populated JAD and run on a MIDP 2.0 compliant device with the appropriate protection domain root certificate