FORUM NOKIA

MIDP 2.0: Tutorial On Signed MIDlets

Version 1.0; May 19, 2004

NOKIA

Contents

Change History

May 19, 2004	Version 1.0	Initial document release

1 Introduction

This document is a tutorial on how to create signed MIDlet suites, following the recommendations of MIDP version 2.0 [MIDP 2.0] and the Java™ Technology for the Wireless Industry [JTWI] specifications. It includes a simple example as well as the procedure to sign and install the suite in a MIDP 2.0 device.

The aim of this document is to show the different requirements and possible problems to be found when signing a MIDlet suite. In particular, it is important to be aware of the necessary certificates and the protection domains used by the target device and emulator.

The new security model introduced in MIDP 2.0 is relevant to MIDlet developers who need to use functions or APIs that are considered sensitive. These are, for example, APIs for network connections, messaging, and push functionality. In addition, MIDP optional packages may contain additional restricted APIs.

This document assumes that you are familiar with Java programming. It also assumes that you understand the basics of MIDP programming, for example, by having read the Forum Nokia paper *MIDP 1.0: Introduction to MIDlet Programming* [MIDPPROG]. Knowledge of public key encryption and digital signatures is also recommended [PKCS] as well as their usage in the Java language [Java Certificates].

2 Security in MIDP 2.0

One of the main benefits of MIDP is the openness of the platform, which enables anybody to write software that can run on MIDP devices. MIDlet suites can be downloaded from the network in an anonymous fashion, and as such, there are some security and privacy issues that the user may be concerned about: Can a MIDlet read private data and send it to an unknown server? Can it make unauthorized calls that cost money to the user? Can rogue programs run on the device and potentially cause problems?

Besides the Java language safety features, such as garbage collection and array bounds checking, the MIDP specification adds some additional safety measures. In MIDP 1.0 [MIDP 1.0], the security restrictions are divided into low-level machine security and application-level machine security. Low-level security relates to the verification of the MIDlet suite's class files. The verification is partially done by the developer in the preverification step and partially on the device.

To further enhance security, MIDlet suites run in a "sandbox" that restricts the available APIs to a limited set. There are some additional restrictions, such as the absence of user-defined class loaders and the prohibition of adding new native functions. These restrictions are part of the application-level machine security [CLDC].

In MIDP version 2.0, the security model was enhanced to allow MIDlets access APIs that are considered sensitive. For example, making an HTTP connection is one of such sensitive operations because it may involve monetary costs for the user. MIDP 2.0 introduces the concept of *trusted* and *untrusted* MIDlets. An *untrusted* MIDlet suite has limited access to restricted APIs, requiring user approval depending on the security policy of the device. On the other hand, *trusted* MIDlet suites can acquire some permissions automatically depending on the security policy.

Permissions are used to protect APIs that are sensitive and require authorization. The MIDP 2.0 implementation has to check whether a MIDlet suite has acquired the necessary permission before invoking the API. Permissions have names starting with the package name in the same way as Java™ 2 Platform, Standard Edition (J2SE™) permissions. For instance, the permission to make an HTTP connection is called `javax.microedition.io.Connector.http`. Permissions are documented along the class or package documentation of the protected API.

2.1 Protection Domains

Protection domains are a key security concept in MIDP 2.0. A protection domain is a set of permissions and interaction modes. Those permissions can be either automatically granted or deferred until user approval. They are called *allowed* and *user* permissions respectively. When a MIDlet suite is installed, it is assigned to a given protection domain and acquires its permissions and interaction modes.

User permissions may require an explicit approval by the user. The user can either deny the permission or allow it. There are three interaction modes in which *user* permissions can be granted: *blanket*, *session*, and *oneshot*. When the *blanket* interaction mode is used, the MIDlet suite acquires the permission as long as the suite is installed, unless explicitly revoked by the user. The *session* mode requests the user authorization the first time the API is invoked and its validity is guaranteed while any of the MIDlets in the same suite are running. Finally, *oneshot* permissions request user approval every time the API is invoked. The protection domain determines which of the modes are available for each user permission as well as the default mode.

A MIDlet suite has to request permissions declaratively using the *MIDlet-Permissions* and *MIDlet-Permissions-Opt* attributes, either in the application descriptor or in the manifest file. *MIDlet-Permissions* contains permissions that are critical for the suite's functionality and *MIDlet-Permissions-Opt* indicates desired permissions, which are not so fundamental for the core functionality. For example, for an application's functionality it is critical to make HTTP connections to work. It may also use HTTPS connections for improved security, but it is not so vital. In this case, the application descriptor could look like this:

```
MIDlet-Permissions: javax.microedition.io.Connector.http
MIDlet-Permissions-Opt: javax.microedition.io.Connector.https
```

One of the requirements to install a MIDlet and assign it to a given domain is that the requested permissions should be a subset of the permissions given to the protection domain. As an example, Nokia Developer's Suite for J2ME™ (NDS) 2.1 includes a domain called *minimum*, which has no permissions at all. If a signed MIDlet that includes any permission request is to be installed in the *minimum* domain, the installation procedure will fail because the domain doesn't include any of those permissions. For the same reason MIDlet suites with misspelled *MIDlet-Permissions* or *MIDlet-Permissions-Opt* properties will fail to install.

Each protection domain, except for the *untrusted* domain, is associated to a set of root certificates. When signing a MIDlet suite, it is necessary to use a public key certificate that can be validated to one of those root certificates. This association will be used to assign the MIDlet suite to a given protection domain. The relationship between root certificates and protection domain is that a domain can be associated to many root certificates, whereas a root certificate is associated to only one domain.

The MIDP 2.0 specification recommends four protection domains for GSM/UTMS devices: the *manufacturer*, *operator*, *trusted third party*, and *untrusted* domains. The *manufacturer* domain uses root certificates belonging to the device producer. The *operator* domain is used for the network operator MIDlets and may use root certificates available on storages such as SIM cards. The *trusted third party* domain will encompass well-known Certificate Authorities' (CA) root certificates. Finally, the mandatory *untrusted* domain has not an associated root certificate and is used for unsigned and MIDP 1.0 MIDlet suites.

Nokia's MIDP 2.0 devices carry the same set of *third party* certificates, and consequently a MIDlet signed for one Nokia device will be acceptable for any other Nokia device. Code signing certificates acquired from companies such as Verisign, Thwate, and others can be used to sign MIDlets that would be installed in the *trusted* domain.

Since the amount of domains and their associated permissions may vary among devices and networks, the suite's developer or the entity in charge of distributing it should be aware of them when deciding what certificate to use and what permissions to request.

2.2 Untrusted MIDlet

The MIDP 2.0 specification defines as *untrusted* a MIDlet suite for which the origin and integrity of the JAR file cannot be verified by the device. This doesn't mean that the MIDlet cannot be installed or executed; it just means that the access to restricted operations requires explicit user permission. All MIDP 1.0 MIDlets are by default *untrusted*.

Untrusted MIDlets can call any API without permission if it is not protected. That includes all classes in the following packages:

```
java.util
java.lang
java.io
javax.microedition.rms
javax.microedition.midlet
javax.microedition.lcdui
javax.microedition.lcdui.game
javax.microedition.media
javax.microedition.media.control
```

In case an untrusted MIDlet suite tries to invoke a protected API and doesn't have the permission, for example if the user rejects the permission's prompt, a `SecurityException` will be thrown. The MIDlet should catch those exceptions and handle them properly.

In Nokia's MIDP 2.0 devices, MIDP 1.0 MIDlets will get an `IOException` instead of a `SecurityException` when the MIDlet cannot acquire the permission. This is to ensure backward compatibility with MIDP 1.0 MIDlets that don't expect a `SecurityException` to be raised, for example, when opening an HTTP connection.

It is also worth noticing that the Nokia's UI API is not protected. This includes the classes in the `com.nokia.mid.sound` and `com.nokia.mid.ui` packages.

2.3 Trusted MIDlets

If the device can verify the authenticity and integrity of the MIDlet suite and assign it to a protection domain, the MIDlet suite is said to be *trusted*. A *trusted* MIDlet suite will have its requested permissions granted according to its protection domain. For example, if the `javax.microedition.io.Connector.http` permission was requested and the protection domain has set the permission as *allowed*, no user confirmation will be needed to open an HTTP connection.

Don't confuse the concept of *trusted* MIDlet suite with the *trusted* protection domain. A *trusted* MIDlet suite can be assigned to any protection domain.

To sign your MIDlet you will need a code-signing certificate conforming to the X.509 Public-Key Infrastructure (PKI) specification [X.509]. The device will use a set of root certificates to validate the MIDlet suite's certificate. Among them it is expected to find the manufacturer's root certificate as well as well-known CA's root certificates. Depending on the CA's policy, the certificate can include any number of intermediate certificates that should also be included in the MIDlet.

All the certificates used to sign the MIDlet are to be included in the suite's `JAD` file using the *MIDlet-Certificate-<n>-<m>* attributes. Besides the certificates, the SHA1 digest of the JAR file signed with the suite's certificates is stored in the JAD file on the *MIDlet-Jar-RSA-SHA1* attribute.

The process of verifying whether a MIDlet suite is trusted is done at installation time and/or before launching the MIDlet suite. When a MIDlet suite is verified, the device will perform the basic security checks, such as class verification and attribute checks. If it finds a *MIDlet-Jar-RSA-SHA1* attribute in the JAD file, it will initiate the authentication and authorization procedures.

During the authentication, it reads the chain of certificates in the JAD file written in the attributes *MIDlet-Certificate-<n>-<m>* (where *n* and *m* are numbers indicating the certificate chain), and tries to validate the certificate with one of the root certificates.

If the certificate chain can be validated to a root certificate, the device will extract the public key from the MIDlet's suite certificate and use it to decrypt the *MIDlet-Jar-RSA-SHA1* attribute. The resulting value will be the SHA1 digest of the MIDlet JAR. The MIDP implementation will then calculate the same digest value from the JAR file.

If both digests are equal, the MIDlet suite is authenticated and it will be allocated to the protection domain assigned to the root certificate. If one or more requested permissions on the *MIDlet-Permissions* attribute are not in the protection domain, the installation will not be allowed to continue. On the other hand, if some of the requested *MIDlet-Permissions-Opt* are not in the protection domain, the installation can proceed.

2.4 Certificate Expiration

During the authentication procedure described above, the certificate chain is verified by checking that all the certificates are valid and being used correctly. This includes certificate integrity checking, proper usage properties, and validity period. If any of the certificates in the chain fail, the MIDlet suite will not be installed.

With respect to the validity period, it is important to consider that purchased certificates normally have a restricted period, often one or two years. MIDlet suites can be signed and installed with a certificate only during that time. This means that suites signed with a certificate cannot be installed after or before this period. It is important to keep this in mind, and obtain new certificates when appropriate and sign again the MIDlet suites that need it.

However, any MIDlet suites already installed in a device continue to function normally after the signing certificate is expired.

2.5 Function Groups

Instead of making the user manage each individual permission requested by a MIDlet suite, permissions can be grouped by functionality in Function Groups. The user will then give permissions to Function Groups, for example the “Net Access” Function Group when using network features, rather than explicitly for the `javax.microedition.io.Connector.http` permission. Using a higher-level concept like Function Groups instead of single permissions is better suited for user interaction in small devices.

The MIDP 2.0 and JTWI specifications have defined the following Function Groups:

- *Net Access*: Contains permissions related to network data connections.
- *Messaging*: Set of permissions related to sending or receiving messages like SMS.
- *Auto Invocation*: Permissions related to automatically starting a MIDlet, for example by Push Registration.
- *Local Connectivity*: Permissions related to connection via local ports like IrDA or Bluetooth.
- *Multimedia Recording*: Permissions that allow to record images, audio, video, and so on.
- *Read User Data*: Set of permissions to read user’s data like phone book or calendar entries.
- *Write User Data*: Permissions related to writing user’s data.

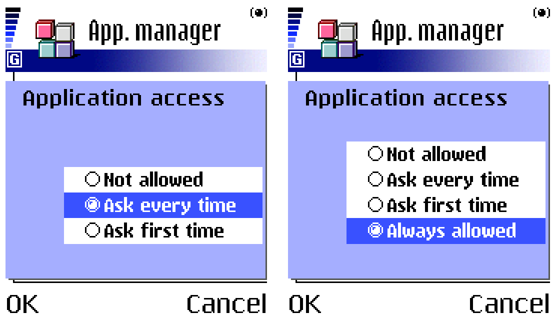
The availability of these Function Groups depends on the device’s capabilities. For instance, “Multimedia Recording” wouldn’t be available for devices without media capturing facilities or for those where the Media API is not available.

The list of Function Groups presented in this document may be extended in the future when new optional APIs are developed.

Function Groups also determine which interaction modes are available for the *trusted* and *untrusted* domains. For example, in the *untrusted* domain, “Net Access” can be set as *session* or *denied*, with blanket being disabled. On the other hand in the *trusted* domain, *oneshot*, *blanket*, and *denied* are allowed.

Another use of Function Groups is to define some combinations of permissions that are not allowed simultaneously. For instance, it wouldn’t be desirable to have *blanket* permission for “Net Access” and “Auto Invocation” at the same time. If this were the case, a MIDlet would set itself for periodical restart and make network connections without user permission, creating unsolicited charges for the user.

An example of Function Groups usage can be seen in the Nokia 6600 mobile phone screenshots shown in Figure 1. They show the 'Network Access' Function Group found in the Application Manager application. The screenshot on the left is for an unsigned MIDlet suite, and the one on the right is for the same MIDlet signed with a Verisign certificate. Clearly the difference is that with the signed MIDlet the user can allow the MIDlet to open network connections without user's prompts.


App. manager

Application access

Not allowed
[✓ Ask every time]
Ask first time

OK Cancel

App. manager

Application access

Not allowed
Ask every time
Ask first time
[✓ Always allowed]

OK Cancel

Figure 1: Network Access Function Group for an unsigned and signed MIDlet suite (Nokia 6600 screenshots)

3 Example of a Signed MIDlet

3.1 Description

A very simple example is described here to show how to request permissions and how the signing procedure works. The MIDlet simply loads a document using an HTTP connection. The actual content of the document is not relevant. The important point is that the MIDlet needs to call a sensitive operation. This chapter will examine both the scenario of building an *untrusted* MIDlet as well as the scenario of building a *trusted* MIDlet. The MIDlet has only one class, `SignedMIDlet`, which sets a textbox and adds two commands for closing the MIDlet and loading a page. The page's URL is set in the application descriptor. When the page is loaded, the amount of bytes read is displayed on the screen. In case of an error, the error message is displayed. The network operation is done in a separate thread.

The example was developed using Nokia Developer's Suite for J2ME™ 2.1 (NDS 2.1) and Series 60 MIDP Concept SDK Beta 0.3.1, Nokia Edition (Concept SDK) emulator.

3.2 SignedMIDlet.java

This is the only class of the MIDlet suite, and it sets a form to be displayed. The `Load` command will attempt to open an HTTP connection and count the target page size in bytes. Any errors will also be displayed.

```
import java.io.*;
import javax.microedition.io.*;
import javax.microedition.lcdui.*;
import javax.microedition.midlet.*;

public class SignedMIDlet
  extends MIDlet
  implements Runnable, CommandListener
{
  private final TextField text
    = new TextField("", "", 256, TextField.ANY);
  private final Form form = new Form("HTTP Result");
  private final Command exitCommand = new Command("Exit", Command.EXIT,
1);
  private final Command okCommand = new Command("Load", Command.OK, 1);


  public SignedMIDlet() {}


  public void startApp()
  {
    if (Display.getDisplay(this).getCurrent()==null) {
      form.setCommandListener(this);
      form.addCommand(exitCommand);
      form.addCommand(okCommand);
      form.append(text);
      Display.getDisplay(this).setCurrent(form);
    }
  }


  public void pauseApp() {}


  public void destroyApp(boolean unconditional) {}


  public void run()
```

```
    {
      // do an action that requires permission, e.g. HTTP connection
      try
      {
        String url = getAppProperty("url");
        HttpConnection connection = (HttpConnection) Connector.open(url);
        InputStream in = connection.openInputStream();
        int counter = 0;
        int ch;
        while ((ch = in.read()) != -1)
        {
          counter++;
        }
        in.close();
        connection.close();
        text.setString("Bytes read: " + counter);
      }
      catch (IOException e)
      {
        text.setString("IOException: " + e.getMessage());
      }
      catch (SecurityException e)
      {
        text.setString("SecurityException: " + e.getMessage());
      }
    }


    public void commandAction(Command command, Displayable displayable)
    {
      if (command == exitCommand)
      {
        notifyDestroyed();
      }
      else if (command == okCommand)
      {
        new Thread(this).start();
      }
    }

  }
```

3.3 Permissions

The MIDlet needs to inform which permissions it wants to acquire. In the case of `SignedMIDlet`, the `javax.microedition.io.Connector.http` permission will be requested. In NDS 2.1, you can add the permissions under the **Create Application Package** option in the **Permissions** tab, as shown in Figure 2 (You may need to select a MIDP 2.0 emulator as default to enable this option.).

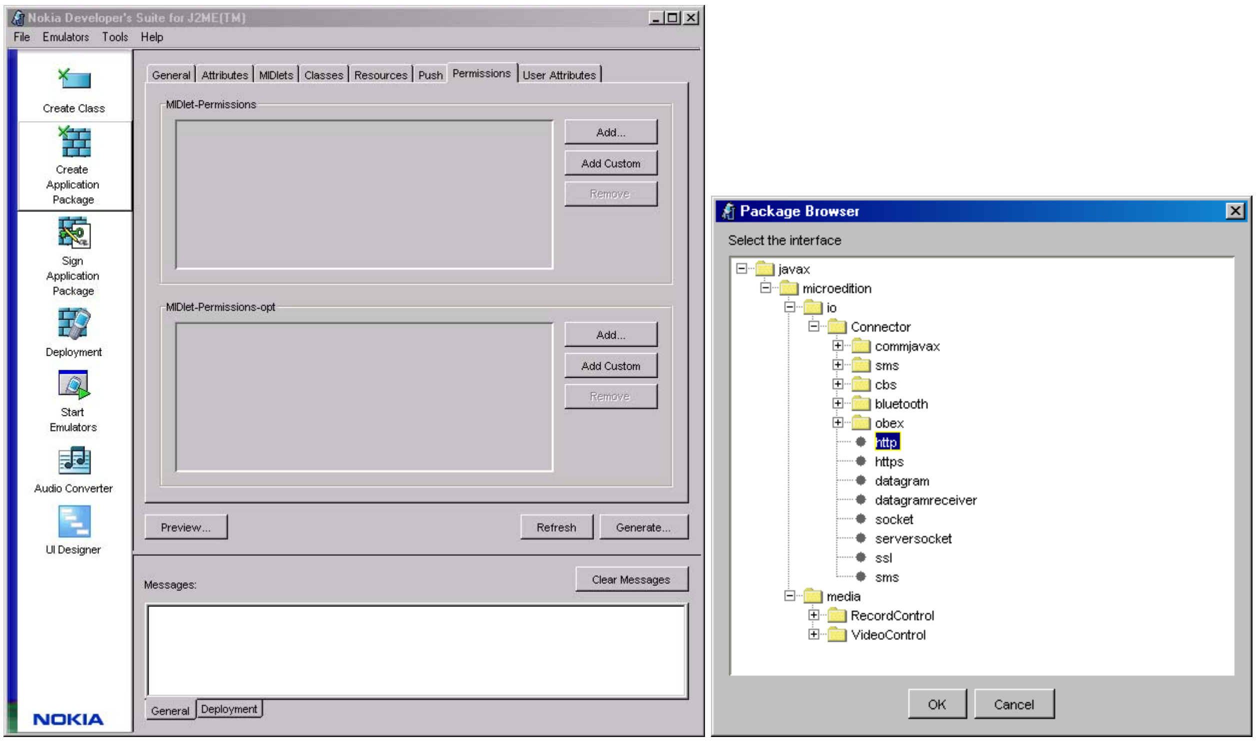


Figure 2: The Create Application Package and Permissions dialog

Alternatively, the requested permissions can be written directly to the JAD application descriptor.

3.4 Signing Procedure

To sign your MIDlet's JAR you need to obtain a code-signing certificate (Note that SSL certificates are not acceptable to sign code). As a developer, you have to be aware what certificates will be acceptable for your target device. For example Nokia devices can accept certificates signed by companies such as Verisign and Thwate. In this example, the JAR will be signed by using a certificate signed by Verisign and purchased through the normal channels.

When signing the JAR file, the certificate will include an identifier of you as the developer or your company, so that the user can identify who made the MIDlet. This is also an important input for technical non-repudiation.

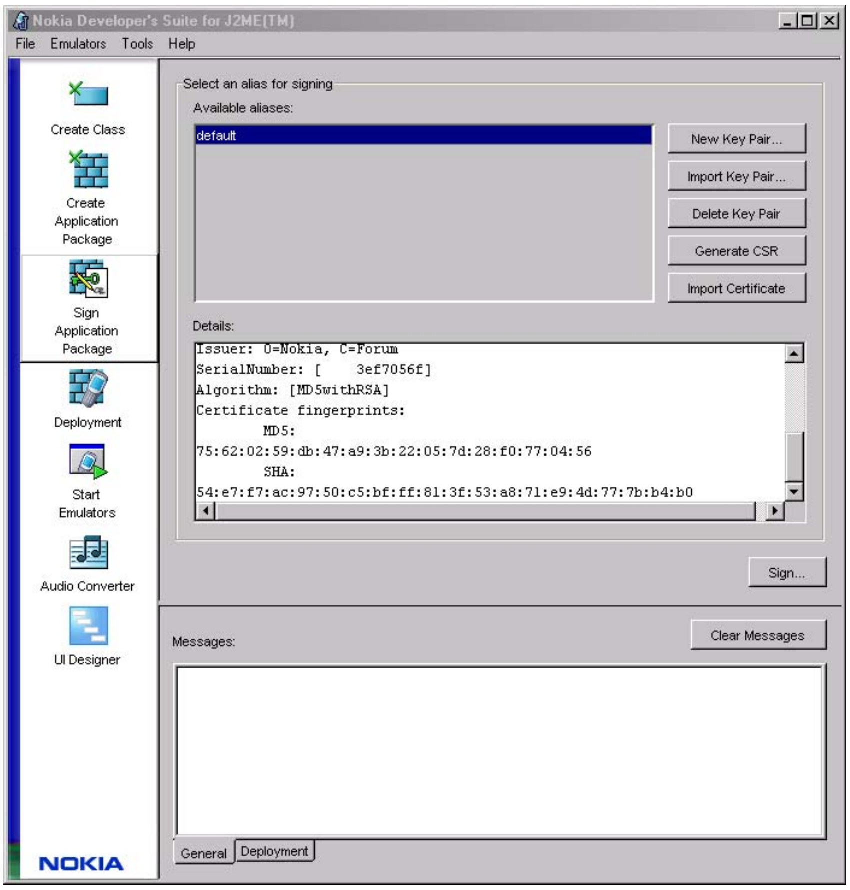
For experimentation purposes, it is possible create your own “self-signed” certificate and load it to the emulator. However, this approach won't work in actual Nokia devices since the set of root certificates is closed.

The signing procedure can be done using the NDS 2.1 GUI or command line utilities. Both procedures are explained below.

3.4.1 Signing procedure using the NDS 2.1 GUI

When using the GUI to sing the MIDlet's JAR, you would proceed as follows (Steps 2 to 5 are done only once per each certificate):

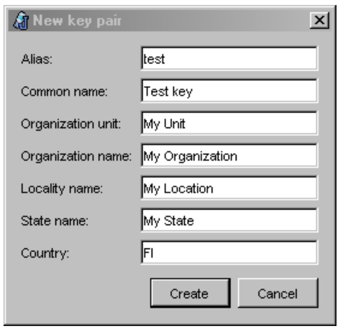
1. Go to the **Create Application Package** tool, add the needed permissions, and package your application.
2. Go to the **Sign Application Package** tool. This will display the screen shown in Figure 3.



```
Issuer: O=Nokia, C=Forum
SerialNumber: [    3ef7056f]
Algorithm: [MD5withRSA]
Certificate fingerprints:
         MD5:
75:62:02:59:db:47:a9:3b:22:05:7d:28:f0:77:04:56
         SHA:
54:e7:f7:ac:97:50:c5:bf:ff:81:3f:53:a8:71:e9:4d:77:7b:b4:b0
```

Figure 3: The Sign Application Package screen

3. Here you can create a new key using the **New Key Pair...** button. This will display the window shown in Figure 4 requesting values for the key's attributes alias, domain, and company name.



Alias:	test
Common name:	Test key
Organization unit:	My Unit
Organization name:	My Organization
Locality name:	My Location
State name:	My State
Country:	FI

Figure 4: Create a new key pair

4. You can use this key as your self-signed certificate. In this example case, the **Generate CSR** button was used in NDS 2.1 to generate a Certificate Signing Request (CSR) file. This file was sent to Verisign to purchase a code-signing certificate. Normally all CAs will proceed to verify your identity by different means and then issue the certificate.
5. Once the CA returns the certificate, it can be imported to NDS 2.1 using the **Import Certificate** button.
6. Now that the certificate has been imported, you can press the **Sign...** button and select the JAD file to be signed.

7. Once this is done you can open the JAD file that now includes three new attributes encoded using the base 64 method. These include two certificates and the JAR's signature. Two certificates are obtained due to Verisign's practice of including an intermediate certificate.

```
MIDlet-Certificate-1-1:
MIIEHjCCA4egAwIBAgIQce1yiTZcQTou7Hh+fx0kEzANBgkqhkiG9w0BAQUFADCBpzEXMBUG
A1UEChMOVmVyaVNpZ24sIEluYy4xHzAdBgNVBAsTFlZlcmlTaWduIFRydXN0IE5ldHdvcmsx
OzA5BgNVBAsTMlRlcm1zIG9mIHVzZSBhdCBodHRwczovL3d3dy52ZXJpc2lnbi5jb20vcnBh
IChjKTAxMS4wLAYDVQQDEyVWZXJpU2lnbiBDbGFzcyAzIENvZGUgU2lnbmluZyAyMDAxIENB
MB4XDTAzMDgxNDAwMDAwMFoXDTA…

MIDlet-Certificate-1-2:
MIIDpjCCAw+gAwIBAgIQbaJ66Skutt3AqAAdR247aTANBgkqhkiG9w0BAQUFADBfMQswCQYD
VQQGEwJVUzEXMBUGA1UEChMOVmVyaVNpZ24sIEluYy4xNzA1BgNVBAsTLkNsYXNzIDMgUHVi
bGljIFByaW1hcnkgQ2VydGlmaWNhdGlvbiBBdXRob3JpdHkwHhcNMDExMjAzMDAwMDAwWhcN
MTExMjAyMjM1OTU5WjCBpzEXMBUGA1UEChMOVmVyaVNpZ24sIEluYy4xHzAdBgNVBAsTFlZl
cmlTaWduIFRydXN0IE5ldHdvcmsx…

MIDlet-Jar-RSA-SHA1:
2dQkKhAfMmy/WVIUdh4/O80R1RFWFuA6qi3ImzxolQfi+PnX4cU0PWonLdB86G/WQ/aRdqlc
6/Z0ibi+JYmZUek5zoFCp7nijxoP…
```

3.4.2 Signing procedure using command line utilities

It is possible to do the same signing procedure using command line utilities (Steps 1 to 4 are done only once per each certificate):

1. The key is created using the keytool utility (included in J2SE) with a command such as the following:

```
keytool -genkey -alias SignedMIDlet -keyalg RSA -keystore midlets.sks
Enter keystore password:  midlets
What is your first and last name?
  [Unknown]:  Test Key
What is the name of your organizational unit?
  [Unknown]:  My Unit
What is the name of your organization?
  [Unknown]:  My Company
What is the name of your City or Locality?
  [Unknown]:  My Location
What is the name of your State or Province?
  [Unknown]:  My State
What is the two-letter country code for this unit?
  [Unknown]:  FI
Is CN=Test Key, OU= My Unit, O="My Company", L=My Location, ST=My State,
C=FI correct?
  [no]:  yes

Enter key password for <SignedMIDlet>
        (RETURN if same as keystore password)
```

This will create a new keystore *midlets.sks* with the password *midlets* and a new key with the given distinguished name fields. There will be a new *midlets.sks* file in your current directory. Alternatively, you could omit the `–keystore` command and store the key in the default java keystore.

You can list all the stored keys with the following command:

```
keytool -list -keystore midlets.sks
Enter keystore password:  midlets
Keystore type: jks
Keystore provider: SUN

Your keystore contains 1 entry

signedmidlet, Dec 6, 2003, keyEntry,
Certificate fingerprint (MD5):
C7:8C:F1:63:17:62:0A:43:6A:F7:F1:5F:E1:EC:66:73
```

2. The CSR file can be generated using the following command:

```
keystore –certreq -alias SignedMIDlet -keystore <keystore
path>\midlets.sks –keypass midlets –file request.csr
```

3. The generated CSR file can then be used to purchase a code-signing certificate from a CA.
4. The certificate returned by the CA can then be imported. Notice that the keystore needs to contain already the certificate used to generate the CSR; otherwise it cannot be imported.

```
keytool -import -file <cert_file> -alias SignedMIDlet -keystore
<keystore path>\midlets.sks –keypass midlets
```

5. The JadTool.jar provided with NDS 2 can perform the actual JAR signing with the following command:

```
java -jar <NDS2.1 path>\bin\lib\JadTool.jar -addjarsig -keypass midlets
-alias SignedMIDlet -keystore <keystore path>\midlets.sks -inputjad <jad
path>\Signed.jad -outputjad <jad path>\Signed.jad -jarfile <jar
path>\Signed.jar
```

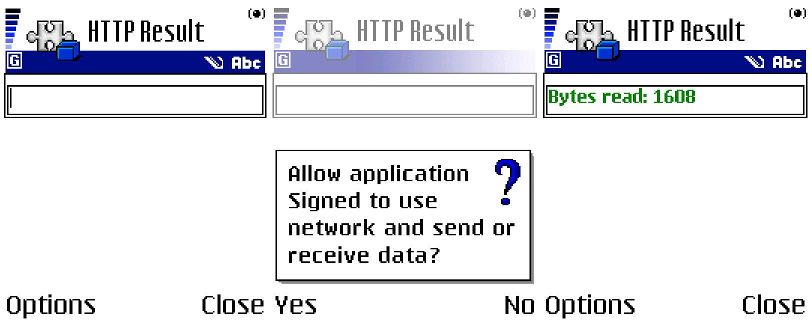
8. Now Signed.jad contains the original properties plus the *MIDlet-Jar-RSA-SHA1* attribute with the digital signature. The next step is to add the certificate using again JadTool.jar with the following command:

```
java -jar <NDS2.1 path>\bin\lib\JadTool.jar -addcert -alias SignedMIDlet
-keystore <keystore path>\midlets.sks -inputjad <jad path>\Signed.jad -
outputjad <jad path>\Signed.jad
```

4 Study Cases

4.1 Untrusted MIDlet

To run the example as an *untrusted* MIDlet, simply proceed as usual with the **Start Emulators / Emulate** command, without signing it. The default protection domain in the Concept SDK is *untrusted*. When the MIDlet tries to open an HTTP connection, the user is prompted for permission. If the user grants permission, the result will look as shown in Figure 5.


HTTP Result

HTTP Result

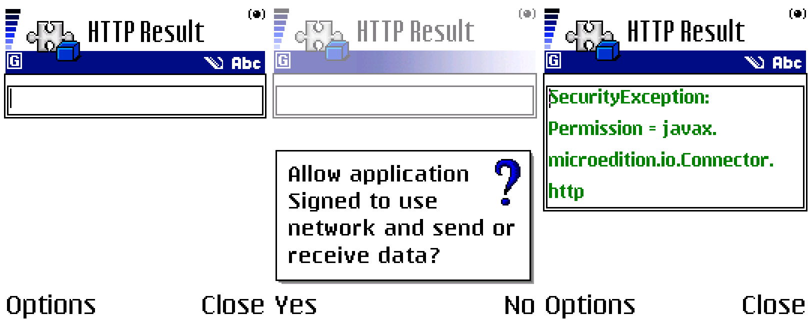
HTTP Result

Bytes read: 1608

Allow application Signed to use network and send or receive data?

Figure 5: Untrusted MIDlet with user permission granted (Nokia 6600 screenshots)

If the user denies access to the sensitive API, a `SecurityException` will be thrown as show in Figure 6.


HTTP Result

HTTP Result

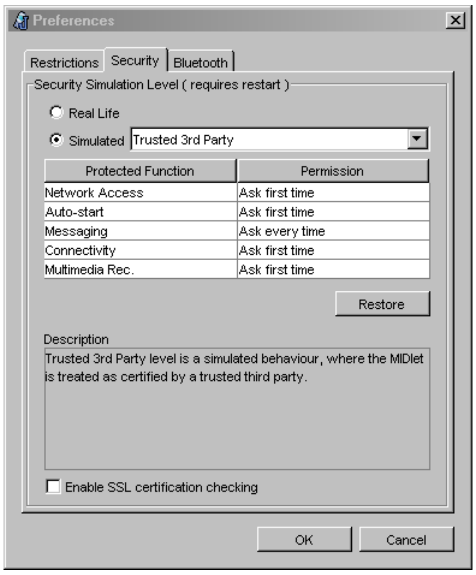
HTTP Result

SecurityException:
Permission = javax.
microedition.io.Connector.
http

Allow application Signed to use network and send or receive data?

Figure 6: Untrusted MIDlet with user permission denied

Notice that the Function Group “Net Access” is by default set to “Ask every time” for the *untrusted* domain. This means that every time the *Load* command is invoked, the user will be prompted for access. It is possible to change this behavior in the Emulator preferences or directly change the default domain to, for example, *Trusted 3rd Party*. In this case, the user will be prompted only once while the MIDlet suite is running.



Security Simulation Level (requires restart)

Real Life [✓ Simulated] Trusted 3rd Party

Protected Function	Permission
Network Access	Ask first time
Auto-start	Ask first time
Messaging	Ask every time
Connectivity	Ask first time
Multimedia Rec.	Ask first time

Description

Trusted 3rd Party level is a simulated behaviour, where the MIDlet is treated as certified by a trusted third party.

☐ Enable SSL certification checking

Figure 7: Default protection domain

If you change the security domain to *minimum*, a `SecurityException` will be thrown without a user prompt as shown in Figure 6.

Using the *Real Life* mode, the Concept SDK will attempt to perform the authorization and authentication procedure. Since the MIDlet suite was not signed, the emulator will run the suite under the *untrusted* protection domain.

4.2 Trusted MIDlet

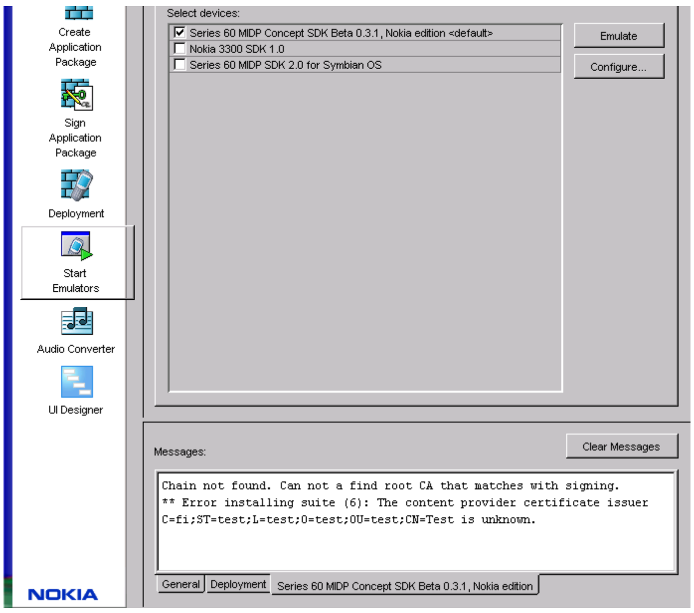
To make a *trusted* MIDlet suite, you first have to sign the MIDlet suite as explained in Section 3.4, “Signing Procedure.” The Concept SDK will execute the authentication and authorization process only if the security domain setting is set as *Real Life*. In that case, the Application Management Service is in charge of the process of verifying the *MIDlet-Jar-RSA-SHA1* attribute and run the suite in the protection domain associated with the root certificate.

To test this, sign the MIDlet suite with an acceptable certificate and run the Concept SDK in *Real Life* mode. If the suite can be authenticated, it will be assigned to the domain corresponding to its root certificate.

4.3 Potential Problems and Mismatches

In the process of signing and installing a MIDlet suite, some things can occur that prevent installation. The most obvious problem is that the MIDlet is signed with a certificate that cannot be authenticated. This would happen, for instance, if you would actually sign the suite with a self-signed certificate. It may also happen if the device doesn’t include the certificate of your CA.

In that case, NDS 2.1 will display an error message as show in Figure 8.

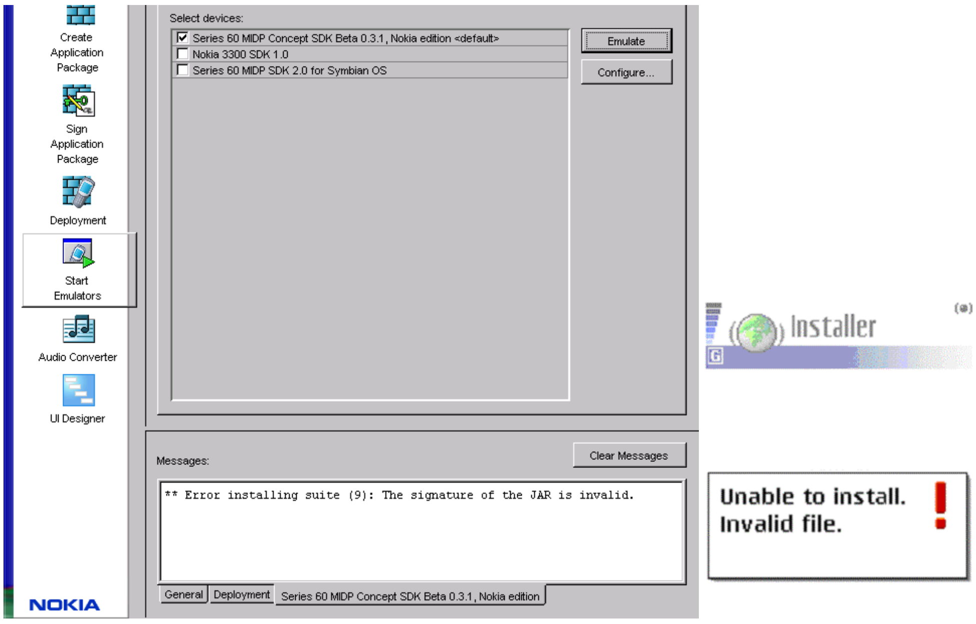

Create Application Package
Sign Application Package
Deployment
Start Emulators
Audio Converter
UI Designer
NOKIA
Select devices:
Series 60 MIDP Concept SDK Beta 0.3.1, Nokia edition <default>
Nokia 3300 SDK 1.0
Series 60 MIDP SDK 2.0 for Symbian OS
Emulate
Configure...
Messages:
Clear Messages
General
Deployment
Series 60 MIDP Concept SDK Beta 0.3.1, Nokia edition

```
Chain not found. Can not a find root CA that matches with signing.
** Error installing suite (6): The content provider certificate issuer
C=fi;ST=test;L=test;O=test;OU=test;CN=Test is unknown.
```

Figure 8: Unknown certificate

Note: With signed applets, the behavior when encountering an unknown certificate depends on the browser and local settings. In the end, the users can accept to trust an applet signed with an unknown certificate if they wish to. In MIDP, this is not allowed. If the device cannot recognize the MIDlet suite, it won't be installed at all. This highlights the need for a certificate issued by a Certificate Authority rather than a self-signed one.

A further problem, which the security framework is especially designed to detect, is the tampering or corruption of a MIDlet JAR file. If for any reason the JAR file has been modified, for example during the OTA transmission or while stored in a device, the integrity check depicted in Section 2.3, "Trusted MIDlets" will fail. This protects the user and the developer of maliciously or accidentally modified MIDlets. This is emulated by manually modifying the manifest file, and therefore making the SHA1 digest incorrect. The emulator shows an error message as show in Figure 9.

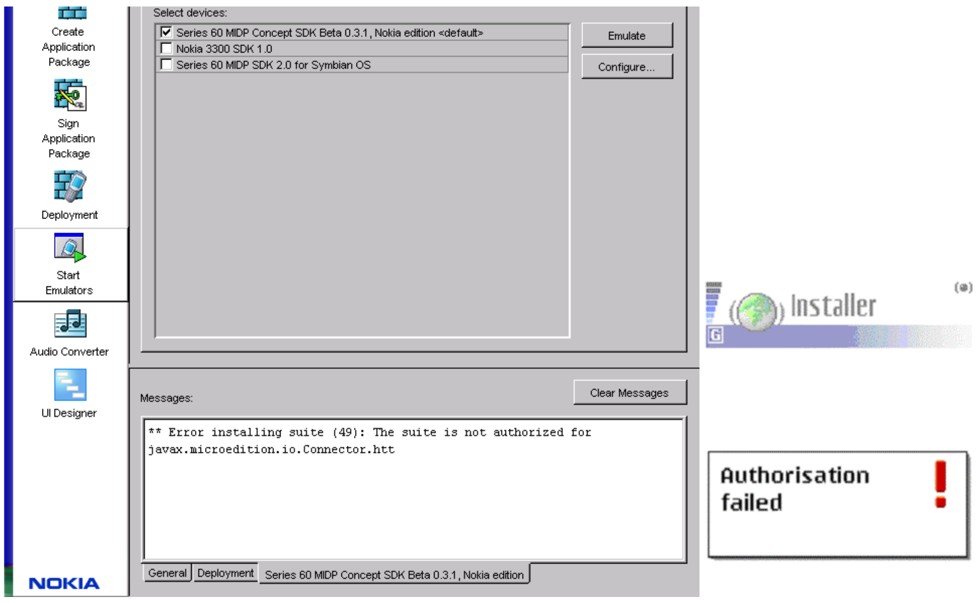


```
** Error installing suite (9): The signature of the JAR is invalid.
```

Unable to install. Invalid file.

Figure 9: Tampered JAR file (Emulator and Nokia 6600 screenshots)

Another potential problem is a mismatch between the permissions requested by the MIDlet suite and the protection domain's permissions. This can be tested in the emulator for example by misspelling the requested permission to `javax.microedtion.io.Connector.htt`. In this case, even tough the application certificate is recognized and the MIDlet suite's integrity accepted, the requested permissions cannot be fulfilled, and the installation is aborted, as shown in Figure 10.



- [✓] Series 60 MIDP Concept SDK Beta 0.3.1, Nokia edition <default>
- [ ] Nokia 3300 SDK 1.0
- [ ] Series 60 MIDP SDK 2.0 for Symbian OS

```
** Error installing suite (49): The suite is not authorized for
javax.microedition.io.Connector.htt
```

Authorisation failed

Figure 10: Not enough permission (Emulator and Nokia 6600 screenshots)

In the case of optional permissions, the MIDlet can be installed even if the domain does not contain those permissions (or they are misspelled).

5 Release Notes

The present tutorial was tested in the Nokia 6600 mobile phone using the maintenance release software version 4.09.1 and higher.

Notice also that the signing procedure depicted in the tutorial is meant to work with NDS 2.1 and it is not guaranteed to work in older versions or in Sun's WTK.

6 Terms and Abbreviations

Term or abbreviation	Meaning
CA	Certificate Authority
Concept SDK	Series 60 MIDP Concept SDK Beta 0.3.1, Nokia Edition
CSR	Certificate Signing Request
J2SE™	Java™ 2 Platform, Standard Edition
MIDP	Mobile Information Device Profile
NDS	Nokia Developer’s Suite for J2ME™

7 References

[MIDP 2.0] Mobile Information Device Profile 2.0, Java Community Process, 2002, http://jcp.org/aboutJava/communityprocess/final/jsr118/index.html

[JTWI] *Java™ Technology for the Wireless Industry*, Java Community Process, 2003, http://www.jcp.org/en/jsr/detail?id=185

[MIDPPROG] *MIDP 1.0: Introduction to MIDlet Programming*, Forum Nokia, 2004, http://www.forum.nokia.com

[PKCS] *Public-Key Cryptography Standards*, RSA, http://www.rsasecurity.com/rsalabs/pkcs/index.html

[Java Certificates] *X.509 Certificates and Certificate Revocation Lists (CRLs)*, JavaSoft, 2001 http:java.sun.com/j2se/1.4.1/docs/guide/security/cert3.html

[MIDP 1.0] *Mobile Information Device Profile (MIDP)*, Java Community Process, 2000, http://jcp.org/aboutJava/communityprocess/final/jsr037/index.html

[CLDC] *J2ME Connected, Limited Device Configuration*, Mobile Information Device Profile 2.0, Java Community Process, 2000, http://jcp.org/aboutJava/communityprocess/final/jsr030/index.html

[X.509] *Internet X.509 Public Key Infrastructure Certificate and CRL Profile*, IETF, January 1999, http://www.ietf.org/rfc/rfc2459.txt