

A Brief Introduction to Secure SMS Messaging in MIDP

Version 1.0; September 23, 2003

Java

NOKIA

Contents

1	Introduction.....	5
1.1	Purpose.....	5
1.2	Bouncy Castle.....	5
2	EncryptedSMS MIDlet.....	6
2.1	User Interface	7
2.2	EncryptedSMSMIDlet.java	8
2.3	SendScreen.java	12
2.4	ReceiveScreen.java.....	14
2.5	MessageCodec.java	15
2.6	ErrorScreen.java	18
2.7	ReportScreen.java	19
	2.7.1 EncryptedSMS.jad.....	20
3	References	20
	Appendix A. Bouncy Castle and Obfuscation	21
	Appendix B. Cipher Comparison	22

Change History

September 23, 2003	V1.0	Initial document release
--------------------	------	--------------------------

Copyright © 2003 Nokia Corporation. All rights reserved.

Nokia and Nokia Connecting People are registered trademarks of Nokia Corporation. Java and all Java-based marks are trademarks or registered trademarks of Sun Microsystems, Inc. Other product and company names mentioned herein may be trademarks or trade names of their respective owners.

Disclaimer

The information in this document is provided “as is,” with no warranties whatsoever, including any warranty of merchantability, fitness for any particular purpose, or any warranty otherwise arising out of any proposal, specification, or sample. Furthermore, information provided in this document is preliminary, and may be changed substantially prior to final release. This document is provided for informational purposes only.

Nokia Corporation disclaims all liability, including liability for infringement of any proprietary rights, relating to implementation of information presented in this document. Nokia Corporation does not warrant or represent that such use will not infringe such rights.

Nokia Corporation retains the right to make changes to this specification at any time, without notice.

License

A license is hereby granted to download and print a copy of this specification for personal use only. No other license to any other intellectual property rights is granted herein.

A Brief Introduction to Secure SMS Messaging in MIDP

Version 1.0; September 23, 2003

1 Introduction

1.1 Purpose

This document presents an example that demonstrates how to communicate securely between MIDlets using SMS. It assumes familiarity with Java™ programming and the basics of MIDP programming, gained, for example by having read the Forum Nokia document *Brief Introduction to MIDP Programming* [MIDPPROG]. Familiarity with the *Wireless Messaging API* [WMA] is recommended as well — this can be achieved by reading the Forum Nokia document *A MIDlet Example Using the Wireless Messaging API and the Nokia SMS API: Chat* [CHAT]. Knowledge of the basics of cryptography is highly recommended. Developers should also be familiar with some basic properties of SMS messaging [GSM0338] [GSM0340] [NOKSMS].

In this example, a secure channel is established between two MIDP devices, using SMS as the transmission medium. To ensure that the SMS remains confidential, a password-based cipher is used to encrypt the message's content.

When using SMS for secure communication, developers should be mostly interested in the confidentiality and integrity of the message. Authentication is achieved by simply looking at the message itself, which includes the sender's phone number. Encrypting the message provides confidentiality, and a message digest can be used to ensure integrity. It should be noted that the encryption process and the digest add extra bytes, which correspondingly reduce payload capacity. If a higher payload is required, the digest can be removed.

1.2 Bouncy Castle

Several third-party packages are currently available to supply the lack of encryption libraries in the MIDP specification [MIDP1.0] [MIDP2.0]. One of the best known is Bouncy Castle [BC]. The Legion of Bouncy Castle is a Web community that produces an open-source Java implementation of cryptographic algorithms. Bouncy Castle includes a clean-room implementation of the Java Cryptography Extension (JCE) and, more significantly, a version targeted to the Java 2 Platform, Micro Edition (J2ME™).

Bouncy Castle for J2ME has a lightweight API that includes a simple cryptographic framework and a large number of algorithms such as block ciphers, public-key ciphers, message digests, etc. It also contains generalized interfaces like BlockCipher, Digest, etc., making it easy to switch algorithms and find the right balance between strength and speed.

The version used in this example is Release 1.18, which is approximately 530 KB in size. The size of the library makes it indispensable to reduce the size of the MIDlet by means of obfuscation. The obfuscator should remove all unused classes and methods. In addition, Bouncy Castle uses a few Java classes not available in MIDP, such as `java.security.SecureRandom`. The obfuscator must rename those classes to avoid conflicts with the security policy of the MIDP device. Appendix A shows an example configuration that will fix this problem.

2 EncryptedSMS MIDlet

The EncryptedSMS MIDlet presented in this document will encrypt a message using a password-based symmetric encryption algorithm and send it using a binary SMS. It is expected that the sender and the receiver have previously agreed on a common password, so there are no provisions for key exchange. The encryption algorithm used is Data Encryption Standard (DES), and the digest is calculated using MD5. Different algorithms could be used that offer higher strength. Appendix B compares the run-time performance of different ciphers and digest methods available in Bouncy Castle.

The SMS is simply a byte array containing a header, the ciphered text, and an optional message's digest. The header includes two bytes of metadata and two bytes containing the size of the cipher text. The first two bytes can be used to indicate properties of the message. In this example, the last bit of the header is used to specify whether a digest was appended to the message.

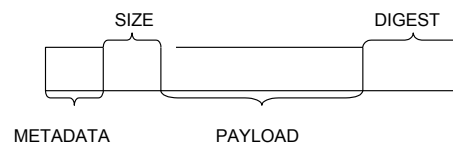


Figure 1: Message format

The MIDlet's user interface has two screens, one for sending a message and one for receiving. The sending message form contains fields for the destination number, text, password, and a choice item to indicate whether to append a digest. It also has a Send command to do the encryption and send the message.

On the receiving end, the application listens for incoming messages and upon their arrival it prompts for the password to be used in the decryption process. If a message has a digest, it will also verify the integrity of the message.

The MIDlet is composed of the main EncryptedSMS MIDlet class and two screens, as shown in Figure 2:

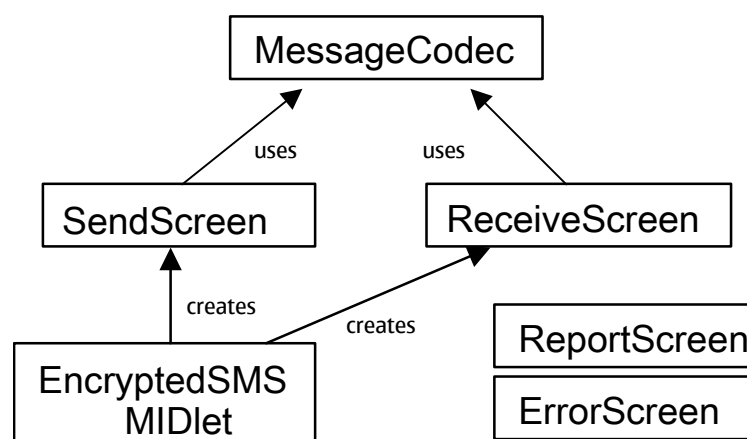


Figure 2: EncryptedSMS MIDlet class diagram

MessageCodec is a utility class containing methods that create the BlockCipher and Digest objects and methods to pack and unpack messages. ReportScreen is used to display the result of the encryption and decryption process and can also display an HEX view of the binary message.

The MIDlet does not register itself to be woken up when a message arrives. This implies that on the receiving end, the user must manually start the MIDlet to process an incoming message. Another option useful for testing is to run the MIDlet and send a message to its own number. In MIDP 2.0, the PushRegistry can be used to automatically start the MIDlet when a message is received.

2.1 User Interface

The MIDlet has screens for sending and receiving encrypted messages. On the sending side, the user interface provides a form containing several fields used to compose the message. The handling of the Send command will verify that all fields have been filled and that the password is at least eight characters long (eight characters are necessary for DES; other algorithms may vary). The MIDlet then encrypts and sends the message. After delivery it will display a report of the process including a measure of the duration of the process.



Figure 3: Sending message sequence

On the receiving end, the user interface displays an announcement and a field to enter the message's password. As previously noted, this example assumes that users will privately exchange passwords. The MIDlet also has a screen to report the received message's content.



Figure 4: Receiving message sequence

In both report screens it is possible to display the content of the message's bytes in hexadecimal format, as shown in Figure 5. (This feature was only added as a means of illustrating the encryption's result and as a debugging aid. It would be removed in a "real-life" MIDlet.)

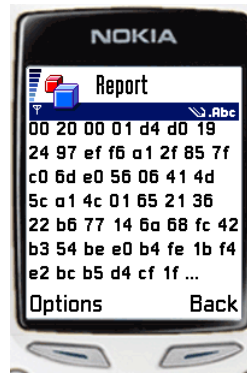


Figure 5: HEX representation of encoded message

In case of error, for example, when entering the wrong password, a report will show the exception's message.



Figure 6: Incorrect password

2.2 EncryptedSMSMIDlet.java

This is the main class, and it controls the state changes between screens. It also owns the message connection and registers itself as a message listener waiting for incoming messages. When sending a message, the *sendMessage* method will launch a new thread that will take care of sending the message. The SMS port used for the connection is read from the JAD file under the *port* property.

The encryption of the plain text and the construction of the message are delegated to MessageCodec.

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

// Main class which inits the connection and create the screens
// It owns the MessageConnection and Display
public class EncryptedSMSMIDlet
    extends MIDlet
    implements MessageListener, Runnable
{
    private final Image logo;
    private final SendScreen displayable;

    private int port;
```

```

private MessageConnection conn;
private Message nextMessage;

public EncryptedSMSMIDlet()
{
    // init basic parameters
    logo = makeImage("/logo.png");
    try
    {
        port = Integer.parseInt(getAppProperty("port"));
    }
    catch (Exception e)
    {
        // in case the property is missing or with a wrong format
        port = 6535;
    }
    ErrorScreen.init(logo, Display.getDisplay(this));
    displayable = new SendScreen(this);
}

public void startApp()
{
    // Build the connection string
    String connection = "sms://:" + port;
    try
    {
        // Initiate the connection and add listener
        conn = (MessageConnection) Connector.open(connection);
        conn.setMessageListener(this);
    }
    catch (IOException e)
    {
        ErrorScreen.showError("Exception creating message connection
to " + connection, displayable);
    }

    Displayable current = Display.getDisplay(this).getCurrent();

    if (current == null)
    {
        // shows splash screen
        String text = getAppProperty("MIDlet-Name") + "\n" +
            getAppProperty("MIDlet-Vendor");
        Alert splashScreen = new Alert(null,
            text,
            logo,
            AlertType.INFO);
        splashScreen.setTimeout(3000);

        Display.getDisplay(this).setCurrent(splashScreen,
displayable);
    }
    else
    {
        Display.getDisplay(this).setCurrent(current);
    }
}

```

```

    }

    public void pauseApp()
    {
    }

    public void destroyApp(boolean unconditional)
    {
        try
        {
            // Close the connection on exit
            if (conn != null)
            {
                conn.close();
            }
        }
        catch (IOException e)
        {
            // Ignore since we are closing anyway
        }
    }

    // Asynchronous callback for inbound message.
    public void notifyIncomingMessage(MessageConnection conn)
    {
        if (conn == this.conn)
        {
            try
            {
                // Create a ReceiveScreen upon message removal
                BinaryMessage incomingMessage = (BinaryMessage)
conn.receive();
                ReceiveScreen handler = new ReceiveScreen(this,
incomingMessage);
                Display.getDisplay(this).setCurrent(handler);
            }
            catch (IOException e)
            {
                ErrorScreen.showError("Exception receiving message " +
e.getMessage(), displayable);
            }
        }
    }

    // Send message in a different thread
    public void run()
    {
        if (nextMessage != null)
        {
            try
            {
                conn.send(nextMessage);
            }
            catch (IOException e)

```

```

        {
            ErrorScreen.showError("Exception sending message(s) " +
e.getMessage(), displayable);
        }
    }
    nextMessage = null;
}

// loads a given image by name
static Image makeImage(String filename)
{
    Image image = null;

    try
    {
        image = Image.createImage(filename);
    }
    catch (Exception e)
    {
        // use a null image instead
    }

    return image;
}

void showMessage(String message)
{
    ErrorScreen.showError(message, displayable);
}

void sendMessage(String number, String plainText, String password,
boolean addDigest)
{
    if (conn != null)
    {
        // create a new message
        BinaryMessage binarySMS = (BinaryMessage) conn.newMessage(
            MessageConnection.BINARY_MESSAGE);
        String address = new
StringBuffer("sms://").append(number).append(":").
append(port).toString();
        binarySMS.setAddress(address);

        // do encryption
        long difference = System.currentTimeMillis();
        try
        {
            byte[] messageContent =
MessageCodec.encodeMessage(plainText, password,
                addDigest);
            // measure how long does it take to encrypt
            difference = System.currentTimeMillis() - difference;
            if (messageContent != null)
            {
                binarySMS.setPayloadData(messageContent);
            }
        }
    }
}

```

```

        StringBuffer report = new StringBuffer(
            "Encryption succesful using cipher ")

        .append(MessageCodec.createEngine().getAlgorithmName());
        if (addDigest)
        {
            report.append(" and digest
");
        }
        .append(MessageCodec.createDigest().
            getAlgorithmName());
        }
        report.append(" took ").append(difference).append(
            " ms. Overall message size is ").
            append(messageContent.length).append(" and
uses ").append(
            conn.numberOfSegments(binarySMS)).append("
segments");

        showReport(report.toString(), messageContent);
        nextMessage = binarySMS;
        // send the message in another thread
        new Thread(this).start();
    }
}
catch (Exception e)
{
    showMessage(e.getMessage());
}
else
{
    showMessage("No connection available");
}
}

// shows the main screen
void showMain()
{
    Display.getDisplay(this).setCurrent(displayable);
}

void showReport(String result, byte[] content) {
    Display.getDisplay(this).setCurrent(new ReportScreen(this,
result, content, content.length));
}
}
}

```

2.3 SendScreen.java

This screen contains fields for sending the encrypted message. When the Send command is invoked it delegates the sending of the message to the main class.

```

import java.io.*;
import javax.microedition.lcdui.*;
import javax.microedition.midlet.*;
import javax.wireless.messaging.*;

```

```

class SendScreen
    extends Form
    implements CommandListener
{
    private final Command sendCommand, exitCommand;
    private final TextField number, text, password;
    private final ChoiceGroup checksum;
    private final EncryptedSMSMIDlet parent;

    public SendScreen(EncryptedSMSMIDlet parent)
    {
        super("Send Secure SMS");
        this.parent = parent;

        // init UI
        sendCommand = new Command("Send", Command.OK, 1);
        exitCommand = new Command("Exit", Command.EXIT, 1);

        number = new TextField("number", "", 20, TextField.PHONENUMBER);
        text = new TextField("message", "", 300, TextField.ANY);
        password = new TextField("password", "", 24,
TextField.PASSWORD);
        checksum = new ChoiceGroup("Add digest", ChoiceGroup.EXCLUSIVE,
new String[]{"Yes", "No"}, null);

        // create the form
        append(number);
        append(text);
        append(password);
        append(checksum);

        addCommand(sendCommand);
        addCommand(exitCommand);
        setCommandListener(this);
    }

    public void commandAction(Command cmd, Displayable displayable)
    {
        if (cmd == sendCommand)
        {
            // verify that all is in place
            if ((number.getString().length() == 0) ||
                (text.getString().length() == 0) ||
                (password.getString().length() < 8))
            {
                parent.showMessage("Parameters too short");
            }
            else
            {
                // send the message
                parent.sendMessage(number.getString(), text.getString(),
password.getString(), checksum.getSelectedIndex() == 0);
            }
        }
        if (cmd == exitCommand)
        {

```

```

        parent.notifyDestroyed();
    }
}
}

```

2.4 ReceiveScreen.java

This screen is activated when a new message arrives. The SMS has been already read by the main class and passed to ReceiveScreen. The screen will request the message's password, delegate the decryption process to MessageCodec, and display the decoded content of the message.

```

import java.io.*;
import javax.microedition.lcdui.*;
import javax.wireless.messaging.*;

class ReceiveScreen
    extends Form
    implements CommandListener
{
    private final Command decodeCommand, backCommand;
    private final EncryptedSMS MIDlet parent;
    private final BinaryMessage message;
    private final TextField password;

    ReceiveScreen(EncryptedSMS MIDlet parent, BinaryMessage message)
    {
        super("Got Secured SMS");

        this.message = message;
        this.parent = parent;

        // init UI
        decodeCommand = new Command("Decode", Command.OK, 0);
        backCommand = new Command("Back", Command.BACK, 0);
        password = new TextField("Enter message's password", "", 256,
TextField.PASSWORD);

        append("Recived message from " + message.getAddress());
        append(password);

        addCommand(decodeCommand);
        addCommand(backCommand);
        setCommandListener(this);
    }

    public void commandAction(Command cmd, Displayable displayable)
    {
        if (cmd == decodeCommand && password.getString().length()>0) {
            decode(message);
            removeCommand(decodeCommand);
        }
        if (cmd == backCommand) {
            parent.showMain();
        }
    }
}

```

```

private void decode(BinaryMessage message)
{
    try
    {
        byte payload[] = message.getPayloadData();
        // measure how long does it take to decrypt
        long difference = System.currentTimeMillis();
        String result = MessageCodec.decodeMessage(payload,
password.getString());
        difference = System.currentTimeMillis() - difference;
        parent.showReport("Decoded message in " + difference + " ms:
" + result, payload);
    }
    catch (Exception e)
    {
        parent.showMessage(e.getMessage());
    }
}
}
}

```

2.5 MessageCodec.java

This utility class takes care of bundling and unbundling the messages. It creates the Block Cipher and the Digest engines and packs the headers and contents of messages. This class sees only bytes arrays and does not care whether they arrived by SMS or some other delivery method.

The *createEngine* and *createDigest* method will construct a DES and MD5 objects in this example. Using a different engine is sufficient for modifying those create methods.

```

import org.bouncycastle.crypto.*;
import org.bouncycastle.crypto.digests.*;
import org.bouncycastle.crypto.engines.*;
import org.bouncycastle.crypto.modes.*;
import org.bouncycastle.crypto.paddings.*;
import org.bouncycastle.crypto.params.*;
import java.io.*;

// The class MessageCodec encodes and decodes messages
class MessageCodec
{
    final static int OPTION_DIGEST = 0x1;

    // Creates a BlockCipher Engine, for example DES or AES
    static BlockCipher createEngine()
    {
        return new CBCBlockCipher(new DESEngine());
    }

    // Creates a Digest Engine, for example MD5 or SHA1
    static Digest createDigest()
    {
        return new MD5Digest();
    }

    // Do the encoding of the message including the headers, encrypted
    content and digest

```

```

    static byte[] encodeMessage(String plainText, String password,
boolean addDigest)
    throws Exception
    {
        byte content[] = plainText.getBytes();
        byte key[] = password.getBytes();

        // Create the cipher. Modify createEngine to use another Engine
        BufferedBlockCipher cipherEngine = new
PaddedBufferedBlockCipher(createEngine());
        // Initialize the cipher for encryption
        cipherEngine.init(true, new KeyParameter(key));

        byte[] cipherText = new
byte[cipherEngine.getOutputSize(content.length)];
        byte[] digest = null;

        // Do encryption
        int cipherTextLength = cipherEngine.processBytes(content, 0,
content.length,
            cipherText, 0);
        cipherEngine.doFinal(cipherText, cipherTextLength);

        // add a digest if required
        if (addDigest)
        {
            Digest digestEngine = createDigest();
            int digestSize = digestEngine.getDigestSize();
            digest = new byte[digestSize];
            digestEngine.update(content, 0, content.length);
            digestEngine.doFinal(digest, 0);
        }

        // Create temporary streams
        ByteArrayOutputStream out = new ByteArrayOutputStream();
        DataOutputStream dout = new DataOutputStream(out);

        // write length
        dout.writeShort(cipherText.length);
        // write options
        int options = (addDigest) ? OPTION_DIGEST : 0;

        dout.writeShort(options);
        // write cipher text
        out.write(cipherText);
        // write digest
        if (addDigest)
        {
            out.write(digest);
        }
        return out.toByteArray();
    }

    // Decodes a message reading the header, decrypting the content and
    veryfing the password
    static String decodeMessage(byte[] messageContent, String password)
    throws Exception

```

```

{
    byte key[] = password.getBytes();

    // create utility streams
    ByteArrayInputStream in = new
ByteArrayInputStream(messageContent);
    DataInputStream din = new DataInputStream(in);

    // read the message content
    int cipherTextLength = din.readShort();
    int options = din.readShort();

    byte[] cipherText = new byte[cipherTextLength];
    in.read(cipherText, 0, cipherTextLength);

    byte[] digest = null;
    Digest digestEngine = createDigest();
    // If the message contains a digest read it
    if ((options & OPTION_DIGEST) == 1)
    {
        int digestSize = digestEngine.getDigestSize();
        digest = new byte[digestSize];
        in.read(digest, 0, digestSize);
    }

    // Create decryption engine
    BufferedBlockCipher cipherEngine = new
PaddedBufferedBlockCipher(createEngine());
    // Initialize the cipher for decryption
    cipherEngine.init(false, new KeyParameter(password.getBytes()));

    byte[] plainText = new
byte[cipherEngine.getOutputSize(cipherTextLength)];

    // do encryption
    int size = cipherEngine.processBytes(cipherText, 0,
cipherTextLength, plainText, 0);
    cipherEngine.doFinal(plainText, size);

    // It is important to do trimming to remove possible padding
    String resultText = new String(plainText).trim();

    // verify the text digest
    if ((options & OPTION_DIGEST) == 1)
    {
        // calculate the digest of the resulting text
        plainText = resultText.getBytes();
        digestEngine.update(plainText, 0, plainText.length);
        byte[] calculatedDigest = new
byte[digestEngine.getDigestSize()];

        digestEngine.doFinal(calculatedDigest, 0);
        // compare digests size
        if (calculatedDigest.length != digest.length)
        {
            throw new Exception("Digest size mismatch");
        }
        // compare byte per byte

```

```

        for (int i = 0; i < calculatedDigest.length; i++)
        {
            if (calculatedDigest[i] != digest[i])
            {
                throw new Exception("Digest mismatch. Integrity of
the message compromised");
            }
        }
    }
    return resultText;
}
}
}

```

2.6 ErrorScreen.java

This is a simple utility screen that displays some message information.

```

import javax.microedition.lcdui.*;

class ErrorScreen
    extends Alert
{
    private static Image image;
    private static Display display;
    private static ErrorScreen instance = null;

    private ErrorScreen()
    {
        super("Error");
        setType(AlertType.ERROR);
        setTimeout(5000);
        setImage(image);
    }

    static void init(Image img, Display disp)
    {
        image = img;
        display = disp;
    }

    static void showError(String message, Displayable next)
    {
        if (instance == null)
        {
            instance = new ErrorScreen();
        }
        instance.setTitle("Error");
        instance.setString(message);
        display.setCurrent(instance, next);
    }
}

```

2.7 ReportScreen.java

This class shows a text report. Additionally it can display a HEX view of a byte array.

```
import org.bouncycastle.util.encoders.Hex;
import javax.microedition.lcdui.*;

class ReportScreen
    extends TextBox
    implements CommandListener
{
    private final Command backCommand, viewHexCommand;
    private final byte[] content;
    private final int size;
    private final EncryptedSMSMIDlet parent;

    ReportScreen(EncryptedSMSMIDlet parent, String text, byte[] content,
int size)
    {
        super("Report", text, 256, TextField.ANY);

        this.content = content;
        this.size = size;
        this.parent = parent;

        backCommand = new Command("Back", Command.BACK, 0);
        viewHexCommand = new Command("View Hex", Command.ITEM, 0);

        addCommand(backCommand);
        addCommand(viewHexCommand);
        setCommandListener(this);
    }

    public void commandAction(Command cmd, Displayable displayable)
    {
        if (cmd == viewHexCommand && content != null)
        {
            String result = new String(Hex.encode(content));
            // format the output nicely
            StringBuffer hexText = new StringBuffer();
            for (int i = 0; i < size; i++)
            {
                hexText.append(result.substring(i * 2, i * 2 +
2)).append(" ");
            }
            String finalText = hexText.toString();

            // remove the tail
            if (getMaxSize() > finalText.length())
            {
                result = finalText.substring(0, finalText.length() - 3)
+ "...";
            }
            setString(result);
        }
        if (cmd == backCommand)
        {

```

```

        parent.showMain();
    }
}
}

```

2.7.1 EncryptedSMS.jad

The application descriptor includes a port property used to set the SMS address.

```

MIDlet-1: EncryptedSMS, EncryptedSMS.png, EncryptedSMSMIDlet
MIDlet-Jar-Size: 428
MIDlet-Jar-URL: EncryptedSMS.jar
MIDlet-Name: Encrypted SMS
MIDlet-Vendor: Nokia
MIDlet-Version: 1.0
MicroEdition-Configuration: CLDC-1.0
MicroEdition-Profile: MIDP-1.0
port=6535

```

3 References

- MIDPPROG** *Brief Introduction to MIDP Programming*
Forum Nokia, 2002
<http://www.forum.nokia.com>
- WMA** *JSR 120: Wireless Messaging API*
Java Community Process, 2003
<http://www.jcp.org/en/jsr/detail?id=120>
- CHAT** *A MIDlet Example Using the Wireless Messaging API and the Nokia SMS API: Chat*
Forum Nokia, 2003
<http://www.forum.nokia.com>
- GSM0340** *ETSI TS 100 901 V7.2.0 (1999-07), Digital Cellular Telecommunications System (Phase 2+); Technical Realization of the Short Message Service (SMS) Point-to-Point (PP)*, European Telecommunications Standards Institute
- GSM0338** *ETSI TS 100 900 V7.2.0 (1999-07), Digital Cellular Telecommunications System (Phase 2+); Alphabets and Language-Specific Information*, European Telecommunications Standards Institute
- NOKSMS** *Sending Content over SMS to Nokia Phones, Version 1.0*
Forum Nokia, May 2001
<http://www.forum.nokia.com>
- MIDP 1.0** *Mobile Information Device Profile (MIDP)*
Java Community Process, 2000
<http://jcp.org/aboutJava/communityprocess/final/jsr037/index.html>
- MIDP 2.0** *Mobile Information Device Profile 2.0*
Java Community Process, 2002
<http://jcp.org/aboutJava/communityprocess/final/jsr118/index.html>
- BC** *Legion of the Bouncy Castle*
<http://www.bouncycastle.org>

Appendix A. Bouncy Castle and Obfuscation

The size of the Bouncy Castle JAR file for J2ME is about 530 kilobytes. The size is due to the large number of ciphers, digests, and utility classes implemented in Bouncy Castle. Since one would normally use only one or two of those algorithms, obfuscation is a very effective means to reduce the size of the MIDlet's JAR file.

One commonly used obfuscator is Proguard, which is very effective in reducing the size of the JAR file by removing unnecessary classes and methods. Proguard has been incorporated into Sun's Wireless Toolkit, but the default obfuscation parameters are not adequate for processing the Bouncy Castle API.

The obfuscation of Bouncy Castle presents some special challenges. In particular, some of the algorithms require Java classes that are not available in CLDC. Examples of this are the `java.security.SecureRandom` and `java.math.BigInteger` classes. To solve this problem, Bouncy Castle's J2ME version ships with implementation of those classes for J2ME. This creates an important difficulty because MIDP implementation should reject classes that are under the `java` and `javax` packages for security reasons.

Obfuscation with Proguard doesn't necessarily solve this problem, since package names are preserved. For example, the `java.security.SecureRandom` class will become `java.security.a` and the device will reject it anyway. Proguard offers a solution — the `-defaultpackage` option. This option will move all the classes to a specific package or remove the package name altogether. It should be noted that by using the `-defaultpackage` option the MIDlet main class package name will change. The JAD and MANIFEST files must reflect this to work properly.

If it becomes necessary to unpack the already obfuscated JAR file (for instance, for preverification), it is also wise to use the `-dontusemixedcaseclassnames` option. Otherwise, Proguard may rename the classes using letters in upper- and lowercases like `a` and `A`, when it runs out of names. This may happen when obfuscating Bouncy Castle because of the large number of classes involved. Mixed-case classes are legal but there can be problems when unpacking the file in operating systems that don't distinguish files with different cases.

Below is a possible Proguard configuration file that will correctly obfuscate Bouncy Castle:

```
-libraryjars midpapi.zip
-injars EncryptedSMS.jar
-outjar EncryptedSMS_o.jar
-keep -keep public class * extends javax.microedition.midlet.MIDlet
-defaultpackage
-dontusemixedcaseclassnames
```

Appendix B. Cipher Comparison

Often developers must decide which cipher to use. Different factors play a role in this decision, including factors such as the cipher's strength, run-time performance, and royalties for the use of some algorithms. Given the constraints on the processing power of a MIDP device, run-time performance should be carefully examined. Below is a list comparing the run-time performance of different block ciphers implemented in Bouncy Castle, running in a MIDP 1.0 device. The measurements indicate the average time to process a 224-byte plain text message. The average time indicates how long it takes to process the plain text 100 times. In the case of AES, Bouncy Castle includes three different implementations (fast, middle, light) that have different speed and memory consumption.

It should be noted that the measurements don't reflect the inherent performance differences between the algorithms, because not all of them have been optimized. Also, some algorithms will perform better when processing larger plain text than others. The column to the right displays the average time in milliseconds among five runs. Each run does the encryption 100 times.

The table only provides a relative speed comparison. The lower the number, the faster the algorithm performs the task.

Algorithm	Average Time
DES	787
3DES	1,997
AES-Fast	601
AES-Middle	1,463
AES-Light	1,432
Rijndael	3,858
Blowfish	501
IDEA	881
RC2	1,107
RC5	542

The table below presents the same measurements for creating a digest of the same plain text:

Algorithm	Average Time
MD5	422
SHA1	667

Build > Test > Sell

Developing and marketing mobile applications and services with Nokia

1

Get started

Use free resources available through www.forum.nokia.com: articles covering the latest technical and business issues, tools for developing and deploying applications and services, biweekly newsletters, and active discussion boards.

www.forum.nokia.com

2

Download tools and SDKs

Download free SDKs, emulators, simulators, and other tools that integrate with industry-leading integrated development environments.

www.forum.nokia.com/tools

3

Get specifications and documentation

Get comprehensive device and platform specifications, and find detailed technical documentation, tutorials, technical notes, case studies, FAQs, and more.

www.forum.nokia.com/devices
www.forum.nokia.com/documents

4

Get support and testing services

Access Nokia's wide range of technical support services, including case resolutions from our professional support staff and fee-based online support from our experts. Forum Nokia's Developer Hub services help with development and testing by providing both online and onsite access to servers, messaging centers, and the like.

www.forum.nokia.com/support

5

Take your applications to market

Take your applications and services to market through Nokia Tradepoint and Nokia Software Market. Other opportunities are available through Nokia Mobile Phones and other Series 60 licensees.

www.forum.nokia.com/business

6

Build your business with Nokia

Nokia works with selected leading developers in the games, branded media and content, and enterprise software industries. Submit your application to Nokia at www.forum.nokia.com/business.

www.forum.nokia.com/business