



MOTODEV

The Motorola developer network

USING CRYPTO APIS FOR SECURE COMMUNICATIONS

V 1.0

TECHNICAL ARTICLE



Copyright © 2007, Motorola, Inc. All rights reserved. This documentation may be printed and copied solely for use in developing products for Motorola products. In addition, two (2) copies of this documentation may be made for archival and backup purposes. Except for the foregoing, no part of this documentation may be reproduced or transmitted in any form or by any means or used to make any derivative work (such as translation, transformation, or adaptation) without express written consent from Motorola, Inc.

Motorola reserves the right to make changes without notice to any products or services described herein. "Typical" parameters, which may be provided in Motorola Data sheets and/or specifications, can and do vary in different applications and actual performance may vary. Customer's technical experts will validate all "Typicals" for each customer application.

Motorola makes no warranty in regard to the products or services contained herein. Implied warranties, including without limitation, the implied warranties of merchantability and fitness for a particular purpose, are given only if specifically required by applicable law. Otherwise, they are specifically excluded.

No warranty is made as to coverage, availability, or grade of service provided by the products or services, whether through a service provider or otherwise. No warranty is made that the software will meet your requirements or will work in combination with any hardware or application software products provided by third parties, that the operation of the software products will be uninterrupted or error free, or that all defects in the software products will be corrected.

In no event shall Motorola be liable, whether in contract or tort (including negligence), for any damages resulting from use of a product or service described herein, or for any indirect, incidental, special or consequential damages of any kind, or loss of revenue or profits, loss of business, loss of information or data, or other financial loss arising out of or in connection with the ability or inability to use the Products, to the full extent these damages may be disclaimed by law.

Some states and other jurisdictions do not allow the exclusion or limitation of incidental or consequential damages, or limitation on the length of an implied warranty, therefore the above limitations or exclusions may not apply to you.

This warranty gives you specific legal rights, and you may also have other rights, which vary from jurisdiction to jurisdiction.

Motorola products or services are not designed, intended, or authorized for use as components in systems intended for surgical implant into the body, or other applications intended to support or sustain life, or for any other application in which the failure of the Motorola product or service could create a situation where personal injury or death may occur.

Should the buyer purchase or use Motorola products or services for any such unintended or unauthorized application, the buyer shall release, indemnify and hold Motorola and its officers, employees, subsidiaries, affiliates, and distributors harmless against all claims, costs, damages, and expenses, and reasonable attorney fees arising out of, directly or indirectly, any claim of personal injury or death associated with such unintended or unauthorized use, even if such claim alleges that Motorola was negligent regarding the designing or manufacturing of the product or service.

Motorola recommends that if you are not the author or creator of the graphics, video, or sound, you obtain sufficient license rights, including the rights under all patents, trademarks, trade names, copyrights, and other third party proprietary rights.

If this documentation is provided on compact disc, the other software and documentation on the compact disc are subject to the license agreement accompanying the compact disc.

Using Crypto APIs for Secure Communications

Version 1.0

August 2008

For the latest version of this document, visit <http://developer.motorola.com>

Motorola, Inc.

<http://www.motorola.com>

Contents

Securing content with Java ME crypto APIs	4
Basic concepts introduction	4
Symmetric cryptography	4
Asymmetric cryptography	4
Message digest	4
Digital signature	5
Bouncy Castle and ProGuard	5
Secure communication process	6
Generate RSA asymmetric public/private key pair	9
Export and import the public/private key pair	9
Generate AES symmetric key	10
Pass the AES symmetric key with RSA encrypt/decrypt	10
Encrypt the plain text with AES	10
Digest the plain text and sign it with the RSA private key	11
Send the cipher and signature to server	11
Decrypt the cipher with AES key and verify the signature	12
Conclusion	13
References	13

Figures

Figure 1: Add ProGuard to MOTODEV Studio for Java ME	5
Figure 2: Secure communication sequence diagram	6
Figure 3: MIDlet input form.....	7
Figure 4: MIDlet preview form.....	8
Figure 5: Server verify result.....	8

Securing content with Java ME crypto APIs

MIDP 2.0 offers support for Secure Socket Layer/HyperText Transport Protocol Secure (SSL/HTTPS), which secures a connection from a client to a server. There are limitations when using SSL that you need to keep in mind:

- SSL secures the connection; it does not secure the content
- SSL is a point-to-point protocol, not end-to-end protocol

Securing the communication content with Java ME crypto APIs is another choice and is more flexible. It can be used to encrypt the data part of the communication, ensuring the data remains private during its transit through the server.

For example, in a mobile IM (Instant Messaging, like OICQ) application, all clients are connected to the server and the server forwards messages from one client to another. If SSL is used for the secure communication, the clients must setup secure connections to the server and all the messages are encrypted, however the data portion of the message can be intercepted at the server. If crypto APIs are used, the sender client can encrypt only the data part of the messages and let the receiver client decrypt them, ensuring that the messages remain private even during handling at the server.

Basic concepts introduction

First, let's introduce several basic concepts.

Symmetric cryptography

Symmetric cryptography uses one key for both encryption and decryption. Advanced Encryption Standard (AES) is one of the most popular algorithms used in symmetric key cryptography. The AES key is a series of bits with fixed length. In our project, it is 128 bits in length.

Asymmetric cryptography

Also known as public-key cryptography, asymmetric cryptography uses a different key for encryption than for decryption. The user who knows the encryption key of an asymmetric cryptography can encrypt messages, but cannot derive the decryption key and cannot decrypt messages that have been encrypted.

RSA is an algorithm for public-key cryptography. The initials represent the first letters in the surnames (Rivest, Shamir, and Adleman) of the team that created this method. RSA utilizes a public/private key pair; a message encrypted by public key could be decrypted by private key and vice versa.

Message digest

Message digest is a hash algorithm which gets fixed length bytes from the input text. Any modification in the input text will change the output bytes. MD5 and SHA1 are the most frequently used message digest algorithms.

Digital signature

A digital signature is a representation of data generated using the sender's private key that can be used to authenticate the identity of the sender of the data. It can be used to ensure that the original data or document that has been transferred is unchanged. A digital signature can be used with any kind of message, so that the receiver can be sure of the sender's identity and that the message arrives intact and untampered.

Bouncy Castle and ProGuard

Bouncy Castle is an open source, light weight, Java crypto project. Compared to commercial Java ME crypto products, it has rich content and algorithms but less documentation and efficiency. The Bouncy Castle Java ME package 1.4.0 was used in our demo project. Bouncy Castle can be downloaded from the URL below:

<http://downloads.bouncycastle.org/java/lcrypto-j2me-140.tar.gz>

ProGuard is a free Java class obfuscator. It detects and removes unused classes, fields, methods, and attributes. It optimizes byte code and removes unused instructions. It also renames the remaining classes, fields, and methods. Before ProGuard processing, the size of our MIDlet JAR (Java Archive) package is more than 900Kb; after ProGuard processing, the size is less than 30Kb. All unused classes are removed from the package. The figure below shows how to include ProGuard in the MOTODEV package.

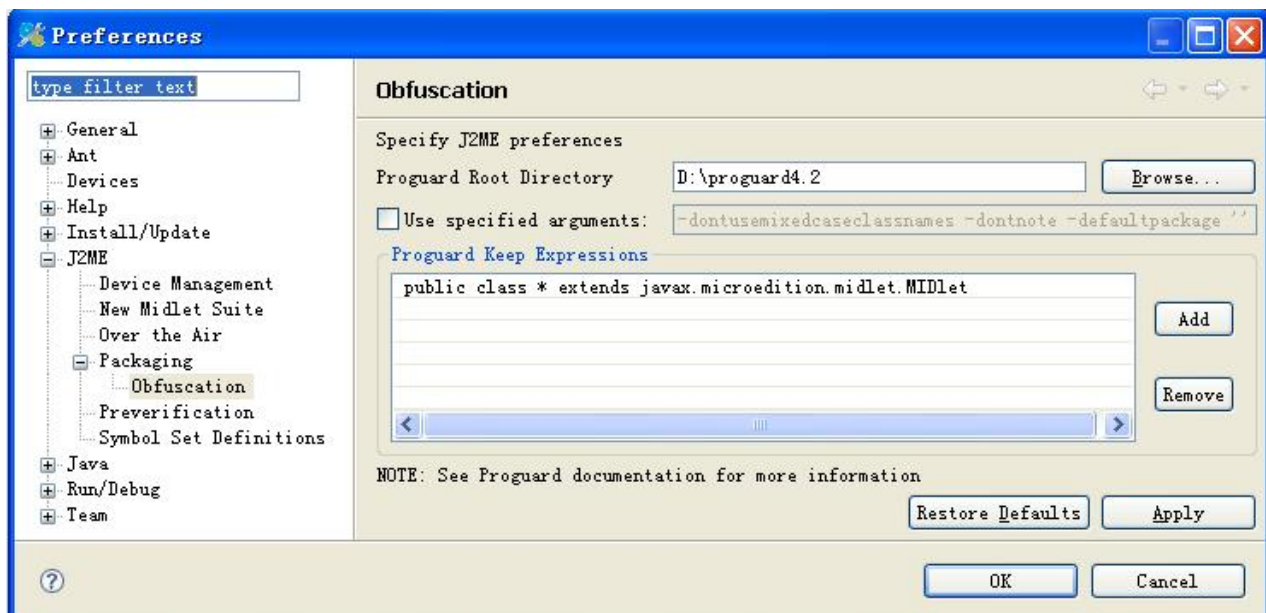


Figure 1: Add ProGuard to MOTODEV Studio for Java ME

The classes in a MIDlet must be obfuscated when using the Bouncy Castle light weight APIs. Otherwise the system will popup a “Class not found” exception. The exception occurs because of package name conflicts in the Bouncy Castle package. For more information, please read FAQ 3 in the URL below:

<http://www.bouncycastle.org/wiki/display/JA1/Frequently+Asked+Questions>

Secure communication process

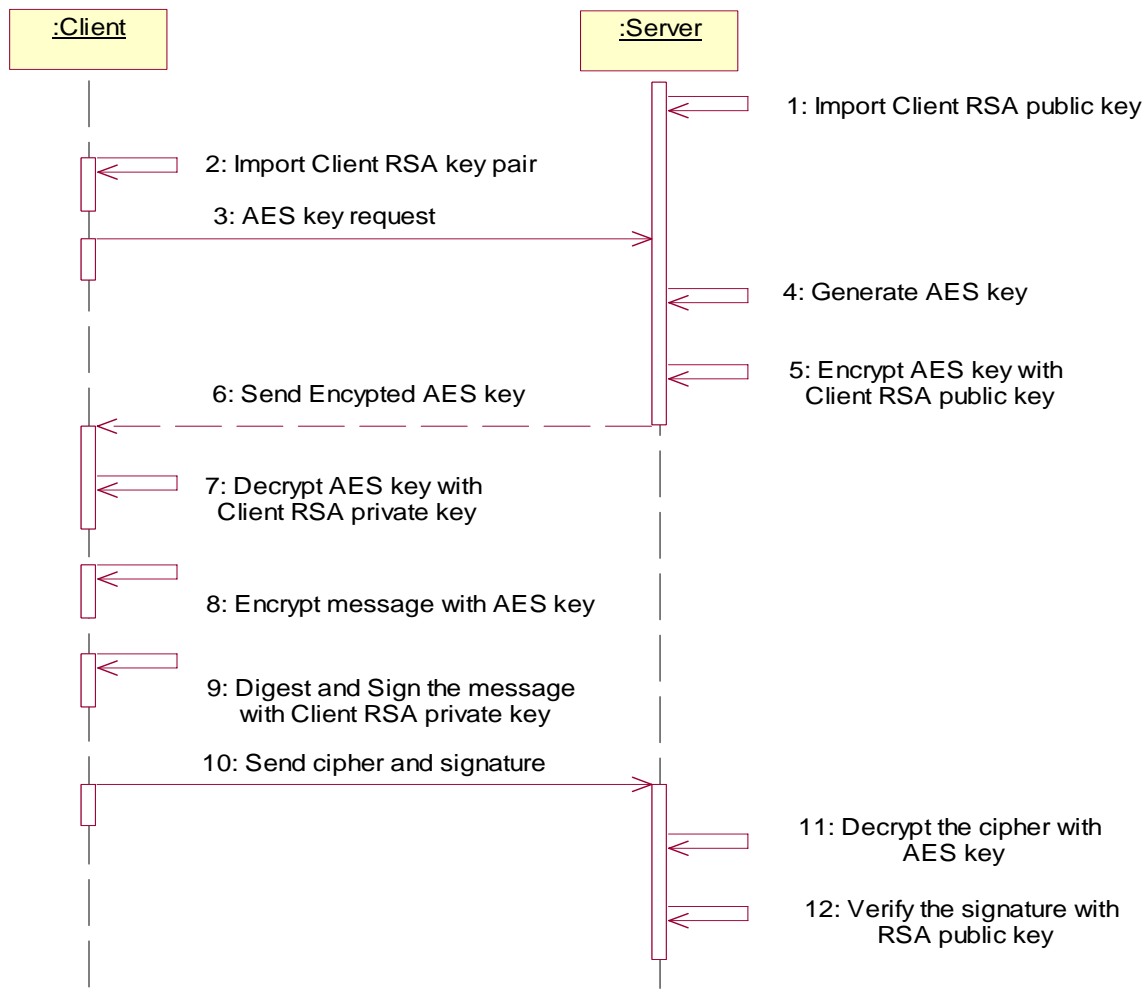


Figure 2: Secure communication sequence diagram

In theory, the asymmetric algorithm could be one thousand times slower than the symmetric algorithm. So RSA is used to exchange the symmetric key and sign the message, and the major content is encrypted using a symmetric algorithm.

MOTODEV SDK for Java ME uses Jboss4.2.1 to run the demo application on the desktop.

- First deploy the SecuServer.war to the Jboss application server
- Then copy the Bouncy Castle package “cldc_classes.zip” to the application server lib

NOTE: In jboss, the server lib is <jboss-4.2.1 path> \server\default\lib\.

Rewrite the server URL in the .jad file before running the demo MIDlet if the IP address or the port is changed. In our MIDlet, it is, “ServerUrl: http://localhost:8080/SecuServer/servlet/SecuServlet.”

http://developer.motorola.com/docstools/technicalarticles/crypto_apis/Crypto_Demo_MIDlet.zip/ Start the MIDlet in the MOTODEV Studio for Java ME Launchpad (supplied with the MOTODEV Java ME SDK, available for download from the MOTODEV website) after the application on the server side is successfully deployed.



Figure 3: MIDlet input form

Here are the steps to follow when running the demo MIDlet:

- First select “key request” to run the application
- Next select “encrypt msg”
- Select “sign msg”
- Select “preview” after the cipher and signature are generated



Figure 4: MIDlet preview form

The preview form has the cipher and the signature, select the “send msg” command to send them to the server. The server receives the cipher and signature from the client, decrypts the cipher, and verifies the signature.

```

16:10:23,984 INFO [STDOUT] signature: GoRGc79ZYsAxnUotXdvSuPdniQ39s0cwzMGmL+t2RpXqIF
16:10:23,984 INFO [STDOUT] cipher: JCJ/qD0ldczqD7YPv127cg==
16:10:23,984 INFO [STDOUT] text: Hello world!
16:10:23,984 INFO [STDOUT] verify succeed!
  
```

Figure 5: Server verify result

Generate RSA asymmetric public/private key pair

The RSA key pair is generated and exported off line in the J2SE environment.

```
public void generateRSAKeyPair () throws Exception {
    RSAPrivateCrtKeyParameters RSAPrivKey = null;
    RSAKeyParameters RSApubKey = null;
    SecureRandom sr = new SecureRandom();
    BigInteger pubExp = new BigInteger("10001", 16);
    RSAKeyGenerationParameters RSAKeyGenPara =
    new RSAKeyGenerationParameters(pubExp, sr, 1024, 80);
    RSAKeyPairGenerator RSAKeyPairGen = new RSAKeyPairGenerator();
    RSAKeyPairGen.init(RSAKeyGenPara);
    AsymmetricCipherKeyPair keyPair = RSAKeyPairGen.generateKeyPair();
    RSAPrivKey = (RSAPrivateCrtKeyParameters) keyPair.getPrivate();
    RSApubKey = (RSAKeyParameters) keyPair.getPublic();
}
```

Export and import the public/private key pair

Bouncy Castle does not offer key pair serialize functions, so we have to implement this functionality by saving/loading all key parameters to a file.

```
public void exportRSAKeypair(){
    BigInteger mod = RSAPrivKey.getModulus();
    BigInteger privExp = RSAPrivKey.getExponent();
    BigInteger pubExp = RSAPrivKey.getPublicExponent();
    BigInteger dp = RSAPrivKey.getDP();
    BigInteger dq = RSAPrivKey.getDQ();
    BigInteger p = RSAPrivKey.getP();
    BigInteger q = RSAPrivKey.getQ();
    BigInteger qInv = RSAPrivKey.getQInv();
    System.out.println("mod:"+ new
    String(Base64.encode(mod.toByteArray())));
    ...
}
```

To export the public/private key pair to a file, just redirect the system out to a file.

```
public void loadKeypair(String smod, String sprivExp,
    String spubExp, String sdp,
    String sdq, String sp, String sq, String sqInv){
    BigInteger mod = new BigInteger(Base64.decode(smod));
    BigInteger privExp = new BigInteger(Base64.decode(sprivExp));
    BigInteger pubExp = new BigInteger(Base64.decode(spubExp));
    BigInteger dp = new BigInteger(Base64.decode(sdp));
    BigInteger dq = new BigInteger(Base64.decode(sdq));
    BigInteger p = new BigInteger(Base64.decode(sp));
}
```

```
BigInteger q = new BigInteger(Base64.decode(sq));
BigInteger qInv = new BigInteger(Base64.decode(sqInv));
RSAPrivKey = new RSAPrivateCrtKeyParameters
    (mod, pubExp, privExp, p, q, dp, dq, qInv);
RSAPubKey = new RSAKeyParameters(false, mod, pubExp);
}
```

Generate AES symmetric key

The AES key is a 128-bit random big integer and is generated in code as follows:

```
public void generateAESKey(){
    SecureRandom sr = new SecureRandom();
    AESKey = new byte[16];
    sr.nextBytes(AESKey);
}
```

Pass the AES symmetric key with RSA encrypt/decrypt

The server encrypts the AES key, sends it to the client, and the client can decrypt it.

```
public byte[] RSAEncrypt(String plainText) throws Exception {
    byte[] rv = null;
    AsymmetricBlockCipher eng = new RSAEngine();
    eng.init(true, RSAPubKey);
    byte[] ptBytes = plainText.getBytes();
    rv = eng.processBlock(ptBytes, 0, ptBytes.length);
    return rv;
}

public String RSADecrypt(byte[] cipherText) throws Exception {
    byte[] rv = null;
    AsymmetricBlockCipher eng = new RSAEngine();
    eng.init(false, RSAPrivKey);
    rv = eng.processBlock(cipherText, 0, cipherText.length);
    return new String(rv);
}
```

Encrypt the plain text with AES

```
public String AESEncryptStr(String inputString){
    byte[] original = inputString.getBytes();
    BufferedBlockCipher cipher = new PaddedBufferedBlockCipher
        (new CBCBlockCipher(new AEngine()));
    KeyParameter key = new KeyParameter(AESKey);
    cipher.init(true, key);
    int size = cipher.getOutputSize(original.length);
    byte[] result = new byte[size];
}
```



```
int len = cipher.processBytes(original,0,original.length,
    result, 0);
try{
    cipher.doFinal(result, len);
}
catch(Exception e){
}
String b64Result = new String(Base64.encode(result));
return b64Result;
}
```

Digest the plain text and sign it with the RSA private key

```
public String RSASign(String text) throws Exception {
    byte[] toSign = text.getBytes();
    if (RSAPrivateKey == null)
        throw new Exception("Generate RSA keys first!");
    SHA1Digest dig = new SHA1Digest();
    RSAEngine eng = new RSAEngine();
    PSSSigner signer = new PSSSigner(eng, dig, 64);
    signer.init(true, RSAPrivateKey);
    signer.update(toSign, 0, toSign.length);
    return new String(Base64.encode(signer.generateSignature()));
}
```

Send the cipher and signature to server

```
public void sendMessage(String cipher, String signature){
    String url = serverUrl;
    try{
        HttpURLConnection c = (HttpURLConnection) Connector.open(url);
        byte[] buffer = cipher.getBytes();
        byte[] buffer2 = signature.getBytes();
        c.setRequestMethod(HttpURLConnection.POST);
        OutputStream os = c.getOutputStream();
        int length = buffer2.length;
        os.write(length);
        os.write(buffer2);
        int ciLength = buffer.length;
        os.write(ciLength);
        os.write(buffer);
        os.flush();
        os.close();
    }
    catch(Exception e){
    }
}
```

Decrypt the cipher with AES key and verify the signature

```
public String AESDecryptStr(String sCipher){
    byte[] b64Bytes = sCipher.getBytes();
    byte[] toDecrypt = Base64.decode(b64Bytes);
    BufferedBlockCipher cipher = new PaddedBufferedBlockCipher(
        new CBCBlockCipher(new AESEngine()));
    KeyParameter key = new KeyParameter(AESKey);
    cipher.init(false, key);
    int size = cipher.getOutputSize(toDecrypt.length);
    byte[] result = new byte[size];
    int len = cipher.processBytes
        (toDecrypt, 0, toDecrypt.length, result, 0);
    try{
        cipher.doFinal(result, len);
    }
    catch(Exception e){
    }
    String text = new String(result);
    return text;
}

public boolean RSAVerify(byte[] mesg, byte[] sig) throws Exception {
    if (RSAPubKey == null)
        throw new Exception("Generate RSA keys first!");
    SHA1Digest dig = new SHA1Digest();
    RSAEngine eng = new RSAEngine();
    PSSSigner signer = new PSSSigner(eng, dig, 64);
    signer.init(false, RSAPubKey);
    signer.update(mesg, 0, mesg.length);
    return signer.verifySignature(sig);
}
```

```
boolean isV = false;
try{
    text = engine.AESDecryptStr(cipher);
    System.out.println("text: "+ text);
    isV = engine.RSAVerify(text.getBytes(),
        Base64.decode(signature));
}
catch(Exception e){
    signature = "Exception!";
}
if(isV){
    System.out.println("verify succeed!");
}
```

Conclusion

Java Me Crypto APIs like Bouncy Castle offer a powerful and flexible way to handle the secure communication challenge in mobile applications. In this article, we introduced implementing network security including symmetric encryption/decryption, asymmetric encryption/decryption, message digest and signatures.

Our experience is that Public Key/Private Key related data processing, which includes message signing and verification, is a performance bottle neck. The performance could be improved by using 512-bit public/private key pair instead of 1024-bit key pairs.

References

Bouncy Castle Project

<http://www.bouncycastle.org/>

ProGuard Project

<http://proguard.sourceforge.net/>

Enterprise J2ME Developing Mobile Java Applications, Michael Juntao Yuan