DidiSoft OpenPGP Library for Java

version 2.5

About

About the Library

DidiSoft OpenPGP Library for Java is a 100% Java library with no external dependencies.

Features

The library provides functions for OpenPGP encryption, decryption, signing, verification of signed data, clear text signing, one pass signing and encryption, key pair generation, key signing, key revocation, etc.

The library uses internally the open source BouncyCastle Java library.

Setup

The library consists of three JAR files located in the **Bin** folder of the library distribution ZIP file:

1) bcpg-jdk14-145.jar
2) bcprov-ext-jdk14-145.jar
3) pgplib-2.4.jar

They must be copied, referenced and distributes with your software in order the library to work.

Unlimited JCE

Due to export control restrictions, by default JDK is supplied with limited cipher key lengths.

In order to enable full cipher key length additionally should be downloaded:
Java Cryptography Extension (JCE) Unlimited Strength Jurisdiction Policy Files

In both cases you have to download a ZIP file and copy the content of the ZIP in
your JRE's jre/lib/security/ folder.

This operation has to be done for every machine where this library will be distributed.

Bellow you will find links where you can download them for Sun and IBM JVM
(Java Virtual Machine) version 1.6, 1.5 and 1.4.

Java 1.6

Sun JVM 1.6
http://www.oracle.com/technetwork/java/javase/downloads/index.html
Scroll down to
"Other Downloads" - Java Cryptography Extension (JCE)
Unlimited Strength Jurisdiction Policy Files 6"

IBM JVM 1.6
http://www.ibm.com/developerworks/java/jdk/security/60/
Scroll down to Java Cryptography Extension (JCE)
(Login for the IBM web site is required)

Java 1.5

Sun JVM 1.5
http://java.sun.com/javase/downloads/index_jdk5.jsp
Scroll down to "Other Downloads" - Java Cryptography Extension (JCE)
Unlimited Strength Jurisdiction Policy Files 5.0

IBM JVM 1.5
http://www.ibm.com/developerworks/java/jdk/security/50/
Scroll down to IBM SDK Policy files
(Login for the IBM web site is required)

Java 1.4

Sun JVM 1.4
http://java.sun.com/j2se/1.4.2/download.html
Scroll down to Other Downloads > Java Cryptography Extension (JCE)
Unlimited Strength Jurisdiction Policy Files 1.4.2

IBM JVM 1.4
Visit:
https://www14.software.ibm.com/webapp/iwm/web/preLogin.do?source=jcesdk

If you have trouble to set up the Unlimited JCE files do not hesitate to contact us.

From EvaluationToProduction

After a purchase you will receive **download** instructions for the **production copy** of the library.

Please **download** the **production copy** ZIP file and **replace** in your project the **evaluation** version JAR files with the once from the **/Bin** folder of the **production copy** ZIP archive.

The same process should be applied also for **upgrade to a newer version** of the library.

Javadoc in Eclipse

This article is a short list of steps to perform in order to see more meaningful **tooltips** when programming with DidiSoft OpenPGP Library for Java. It assumes that you use Eclipse as your Java IDE.

1. Download and unpack library ZIP.

2. Start a new **Eclipse project** and reference the three JARS located in the **Bin** folder in the location from step 1.

3. In your project **Referenced Libraries** section in the Eclipse **Package Explorer** tab right click pgplib-x.x.jar (x.x is your version in use), select **Properties.**
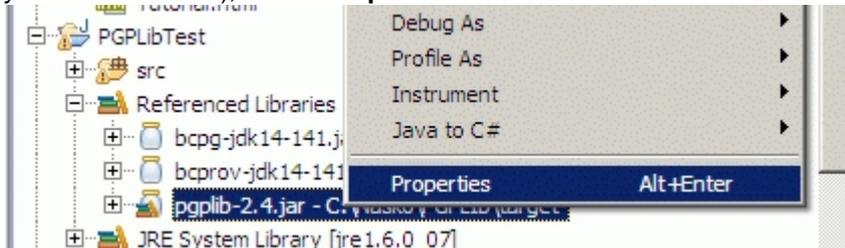
4. In the Javadoc Location dialog enter the location of the **JavaDoc** folder where the library was extracted in step 1.
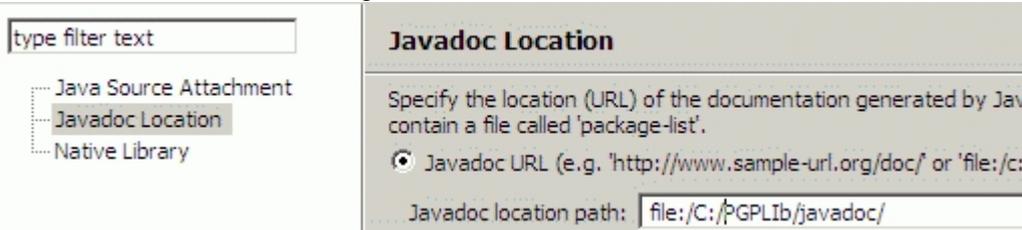
5. Now the JavaDoc should appear when you type methods or properties of the objects from the library, or simply press **F2** when you are over an already typed method.

WAR and EAR project

If your project is running in an Application server (WAR or EAR project) you may encounter **strange exceptions** after **application start/stop**.
You can skip this section if you do not plan to stop/start the application that uses the library.

The reason for these exceptions is that the library JAR files are bundled with the web application and after an application stop/start, they are loaded in another class loader but the BouncyCastle security provider was already registered with class loader that is unavailable (*has been destroyed after the web application stop*)

The solution to this situation is to ship the application without the library JAR files.
They must be placed in a folder **shared** for all applications running on the Application server.

Below are listed the shared folders for some application servers.

Tomcat
<tomcat folder>/shared/lib/

Web Sphere
<was folder>/lib

WebLogic

<weblogic folder>/common/lib

Note: If you application server is not mentioned here, please refer to your application server documentation.

Examples

Programs

EncryptFile

Encryption is the most used function of the OpenPGP cryptography. In order to encrypt a file we need the public key of the recipient.

With OpenPGP Library for Java we have two options. The first one is to keep the recipient's public key in a file on the disk. The second option is to store it in a KeyStore object.

1) Encrypt file with recipient's public key located in a file

This example shows how to encrypt a data file, having the recipient's public key in a file. In our case the recipient's public key file has extension .key, but it can be anything else. The most common public key file name extensions are: *.asc, *.pkr, *.pubkr.

```java
import com.didisoft.pgp.PGPLib;

public class EncryptFile {
  public static void main(String[] args) throws Exception{
    // create an instance of the library
    PGPLib pgp = new PGPLib();

    // is output ASCII or binary
    boolean asciiArmor = false;
    // should integrity check information be added
    boolean withIntegrityCheck = false;

    pgp.encryptFile("INPUT.txt",
            "public.key",
            "OUTPUT.pgp",
             asciiArmor,
             withIntegrityCheck);
  }
}
```

All encrypt methods have two additional parameters:

asciiArmor specifies the format of the result file, when true the file is in ASCII armored format suitable for Email attachments, when false the output file is in binary format.

When withIntegrityCheck is true additional integrity check information is appended to the encrypted file.

2) Encrypt file with recipient's public key located in a KeyStore

We should choose to store our OpenPGP keys in a KeyStore object when we need additional layer of security. This example shows how to encrypt a file with public key located in a Key store. (Note that a key with UserId demo@didisoft.com should already be imported in the KeyStore file.)

```java
import com.didisoft.pgp.KeyStore;
import com.didisoft.pgp.PGPLib;
public class KeystoreEncryptFile {
public static void main(String[] args) throws Exception{
  // create an instance of the KeyStore
  KeyStore keyStore = new KeyStore("pgp.keystore", "changeit");

  // create an instance of the library
```

```
    PGPLib pgp = new PGPLib();

    String recipientUserId = "demo@didisoft.com";

    // is output ASCII or binary
    boolean asciiArmor = true;
    // should integrity check information be added
    boolean withIntegrityCheck = true;

    pgp.encryptFile("INPUT.txt",
                    keyStore,
                    recipientUserId,
                    "encrypted.pgp",
                    asciiArmor,
                    withIntegrityCheck);
  }
}
```

3) Encrypt stream with recipient's public key located in a file

This example shows how to encrypt a stream. This way we can encrypt not only files but any other source that can be read as stream.

```
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.InputStream;
import java.io.OutputStream;

import com.didisoft.pgp.PGPLib;

public class EncryptStream {
 public static void main(String[] args) throws Exception{
  // create an instance of the library
  PGPLib pgp = new PGPLib();

  // is output ASCII or binary
  boolean asciiArmor = true;

  // should integrity check information be added
  boolean withIntegrityCheck = true;

  // obtain the streams
  InputStream inStream = new FileInputStream("INPUT.txt");
  InputStream keyStream = new FileInputStream("public.key");
  OutputStream outStream = new FileOutputStream("encrypted.pgp");

  // Here "INPUT.txt" is just a string to be written in the
  // message OpenPGP packet which contains:
  // file name string, timestamp, and the actual data bytes
  pgp.encryptStream(inStream, "INPUT.txt",
            keyStream,
            outStream,
            asciiArmor,
            withIntegrityCheck);
 }
}
```

4) Encrypt stream with recipient's public key located in a KeyStore

In this example the message source and the encrypted output are streams too. The public key of the recipient is located in a KeyStore file.

```
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.InputStream;
```

```java
import java.io.OutputStream;

import com.didisoft.pgp.KeyStore;
import com.didisoft.pgp.PGPLib;

public class KeyStoreEncryptStream {
 public static void main(String[] args) throws Exception{
  // create an instance of the KeyStore
  KeyStore keyStore = new KeyStore("pgp.keystore", "changeit");

  String recipientUserId = "demo@didisoft.com";

  // create an instance of the library
  PGPLib pgp = new PGPLib();

  // is output ASCII or binary
  boolean asciiArmor = true;
  // should integrity check information be added
  boolean withIntegrityCheck = true;

  // obtain the streams
  InputStream inStream = new FileInputStream("INPUT.txt");
  OutputStream outStream = new FileOutputStream("encrypted.pgp");

  // Here "INPUT.txt" is just a string to be written in the
  // message OpenPGP packet which contains:
  // file name string, timestamp, and the actual data bytes
  pgp.encryptStream(inStream, "INPUT.txt",
                    keyStore,
                    recipientUserId,
                    outStream,
                    asciiArmor,
                    withIntegrityCheck);
 }
}
```

Decrypt

We can decrypt an OpenPGP encrypted file if it is encrypted with a public key and we have the corresponding private key.
Below you will find examples that demonstrate in practice how to decrypt with DidiSoft OpenPGP Library for Java.

1) Decrypt file with private key located in file

This example demonstrates OpenPGP file decryption with a private key stored in a file. The private key password is also needed, in order the key to be used.

The last parameter is the location where we want the decrypted file to be stored. The DecryptFile method returns a String that represent the original file name of the content that was encrypted, which we can use lately to rename OUTPUT.txt with.

```java
import com.didisoft.pgp.PGPLib;

public class DecryptFile {
 public static void main(String[] args) throws Exception{
  // initialize the library instance
  PGPLib pgp = new PGPLib();

  String privateKeyFile = "private.key";
  String privateKeyPass = "changeit";

  String originalFileName = pgp.decryptFile("encrypted.pgp",
                                            privateKeyFile,
                                            privateKeyPass,
                                            "OUTPUT.txt");
 }
```

}

2) Decrypt file with private key located in a KeyStore

This example shows how to decrypt a file with private key stored in a Key store. Keeping our private keys in a KeyStore gives us extra layer of security.

```java
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.InputStream;
import java.io.OutputStream;

import com.didisoft.pgp.KeyStore;
import com.didisoft.pgp.PGPLib;

public class KeystoreDecryptFile {
  public static void main(String[] args) throws Exception {
    // initialize the KeyStore instance
    KeyStore keyStore = new KeyStore("pgp.keystore", "changeit");

    // initialize the library instance
    PGPLib pgp = new PGPLib();

    // The decrypt method returns the original name of the file
    // that was encrypted. We can use it afterwards,
    // to rename OUTPUT.txt to it for example.
    String originalFileName = pgp.decryptFile("encrypted.pgp",
                                              keyStore,
                                              "changeit",
                                              "OUTPUT.txt");
  }
}
```

3) Decrypt stream with private key located in stream

Sometimes we may receive an encrypted data in a way suitable to be read as an input stream, in that case it is easier to decrypt it directly instead of writing it to a file before that. The example below shows how to achieve this:

```java
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.InputStream;
import java.io.OutputStream;

import com.didisoft.pgp.PGPLib;

public class DecryptStream {

 public static void main(String[] args) throws Exception{
   // create instance of the library
   PGPLib pgp = new PGPLib();

  // obtain an encrypted data stream
  InputStream encryptedStream = new FileInputStream("encrypted.pgp");

  InputStream privateKeyStream = new FileInputStream("private.key");
  String privateKeyPassword = "changeit";

  // specify the destination stream of the decrypted data
  OutputStream decryptedStream = new FileOutputStream("OUTPUT.txt");

  pgp.decryptStream(encryptedStream,
               privateKeyStream,
         privateKeyPassword,
         decryptedStream);
 }
```

}

4) Decrypt stream with private key located in a KeyStore

This example shows how to decrypt an OpenPGP encrypted stream when our private decryption key is stored in a KeyStore object.

```java
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.InputStream;
import java.io.OutputStream;

import com.didisoft.pgp.KeyStore;
import com.didisoft.pgp.PGPLib;

public class KeyStoreDecryptStream {
 public static void main(String[] args) throws Exception {
   // initialize the KeyStore instance
   KeyStore keyStore = new KeyStore("pgp.keystore", "changeit");

   // initialize the library instance
   PGPLib pgp = new PGPLib();

   // obtain the encrypted stream
   InputStream encryptedStream = new FileInputStream("encrypted.pgp");
   // specify the decrypted output stream
   OutputStream decryptedStream = new FileOutputStream("OUTPUT.txt");

   String decryptionKeyPassword = "changeit";

   pgp.decryptStream(encryptedStream,
                       keyStore,
                       decryptionKeyPassword,
                       decryptedStream);
 }
}
```

Sign

A digital signature is a number cryptographically computed from the byte contents of the signed document. In OpenPGP the signature message digest is produced with a signing key, usually this is our private key.

OpenPGP signed files contain the original message compressed by default (see Compression) and the digital signature attached appended. The default signature message digest functions can be changed through the Hash option.

Below you will find examples that demonstrate how to OpenPGP sign file and stream data:

1) Sign file with private key located in file

```java
import com.didisoft.pgp.PGPLib;

public class SignFile {
 public static void main(String[] args) throws Exception{
        // initialize the library
        PGPLib pgp = new PGPLib();

        // specify should the output be ASCII or binary
        boolean asciiArmor = false;
        // sign
        pgp.signFile("INPUT.txt",
                   "private.key",
                   "changeit",
                   "signed.pgp",
                   asciiArmor);
 }
```

}

2) Sign file with private key located in a KeyStore

```java
import com.didisoft.pgp.KeyStore;
import com.didisoft.pgp.PGPLib;

public class KeyStoreSignFile {
 public static void main(String[] args) throws Exception{
         // create an instance of the KeyStore
         KeyStore keyStore = new KeyStore("pgp.keystore", "changeit");

         // initialize the library
         PGPLib pgp = new PGPLib();

         // The signing key is usually our private key
         String signUserId = "demo@didisoft.com";
         String signKeyPassword = "changeit";

         // specify should the output be ASCII or binary
         boolean asciiArmor = false;
         // sign
         pgp.signFile("INPUT.txt",
                     keyStore,
                     signUserId,
                     signKeyPassword,
                     "signed.pgp",
                     asciiArmor);
 }
}
```

3) Sign input stream with private key located in file

In the example below the file streams are used only to demonstrate how to encrypt with stream. The streams can be of any other type that implements InputStream.

```java
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.InputStream;
import java.io.OutputStream;
import com.didisoft.pgp.PGPLib;

public class SignStream {
 public static void main(String[] args) throws Exception {
  // create instance of the library
  PGPLib pgp = new PGPLib();

  // if true the output file will be in ascii armored format,
  // otherwise will be in binary format
  boolean asciiArmor = false;

  InputStream dataStream = new FileInputStream("INPUT.txt");
  InputStream privateKeyStream = new FileInputStream("private.key");
  String privateKeyPassword = "changeit";
  OutputStream signedStream = new FileOutputStream("signed.pgp");

  // This parameter is needed because OpenPGP requires
  // the encrypted content to have a file name label
  String internalFileName = "INPUT.txt";

  pgp.signStream(dataStream,
                 internalFileName,
                 privateKeyStream,
                 privateKeyPassword,
                 signedStream,
```

```
                    asciiArmor);
  }
}
```

4) Sign input stream with private key located in a KeyStore

You may notice that the second parameter of the signStream method for KeyStore has a file name parameter. This is often misunderstood, the idea is that this is just a label associated with the encrypted content in the output stream. The OpenPGP format in addition to the encrypted content stores a file name label and time stamp.

```
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.InputStream;
import java.io.OutputStream;
import com.didisoft.pgp.KeyStore;
import com.didisoft.pgp.PGPLib;

public class KeyStoreSignStream {
 public static void main(String[] args) throws Exception{
  // create an instance of the KeyStore
  KeyStore keyStore = new KeyStore("pgp.keystore", "changeit");

  // initialize the library
  PGPLib pgp = new PGPLib();

  // The signing key is usually our private key
  String signUserId = "demo@didisoft.com";
  String signKeyPassword = "changeit";

  InputStream dataStream = new FileInputStream("INPUT.txt");
  // this parameter is just a label that is associated with
  // the encrypted content in the OpenPGP archive
  String signedContentLabel = "INPUT.txt";

  // create the output stream
  OutputStream signedStream = new FileOutputStream("signed.pgp");

  // specify should the output be ASCII or binary
  boolean asciiArmor = false;

  pgp.signStream(dataStream,
                 signedContentLabel,
                 keyStore,
                 signUserId,
                 signKeyPassword,
                 signedStream,
                 asciiArmor);
  }
}
```

Verify

When we receive signed only OpenPGP file from our partners we can decrypt it with arbitrary key, ignoring this way the digital signature or we can verify and extract the data.

The examples below show how to verify the digital signature and extract the data in one pass with OpenPGP Library for Java. For the verification we use the public key of the sender.

1) Verify signed file with sender public key located in file on the disk.

This example assumes that the file signed.pgp was only signed with the private key of the sender. Note that this is different from sign and encrypt in one pass.

import com.didisoft.pgp.PGPLib;

```
public class VerifyFile {
 public static void main(String[] args) throws Exception{
   // create an instance of the library
   PGPLib pgp = new PGPLib();

   // verify
   boolean validSignature = pgp.verifyFile("signed.pgp", "public.key", "OUTPUT.txt");
   if (validSignature) {
         System.out.println("Signature is valid .");
   } else {
         System.out.println("!Signature is invalid!");
   }
 }
}
```

2) Verify signed file with sender public key located in a KeyStore.

In this example the digital signature in the signed file we have received is tried to be verified with the public keys we have imported previously in our KeyStore file. If the public key of the sender is not present in this KeyStore the verification will fail, but anyway the embedded file will be extracted.

```
import com.didisoft.pgp.KeyStore;
import com.didisoft.pgp.PGPLib;

public class KeyStoreVerifyFile {
 public static void main(String[] args) throws Exception{
   // create an instance of the KeyStore
   KeyStore keyStore = new KeyStore("pgp.keystore", "changeit");

   // initialize the library
   PGPLib pgp = new PGPLib();

   // verify
   boolean validSignature = pgp.verifyFile("signed.pgp",
                                           keyStore,
                                           "OUTPUT.txt");

   if (validSignature) {
         System.out.println("Signature is valid.");
   } else {
         System.out.println("Signature is invalid!");
   }
 }
}
```

3) Verify signed stream data with sender public key located in file on the disk.

In the example below the signed data is supplied as a file stream but it can be any kind of input stream.

```
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.InputStream;
import java.io.OutputStream;

import com.didisoft.pgp.PGPLib;

public class VerifyStream {
 public static void main(String[] args) throws Exception {
   PGPLib pgp = new PGPLib();

   InputStream signedStream = new FileInputStream("signed.pgp");
   InputStream senderPublicKeyStream = new FileInputStream("public.key");
   OutputStream outputStream = new FileOutputStream("OUTPUT.txt");

   boolean validSignature = pgp.verifyStream(signedStream, senderPublicKeyStream, outputStream);
```

```
        if (validSignature) {
                System.out.println("Signature is valid.");
        } else {
                System.out.println("Signature is invalid!");
        }
    }
}
```

4) Verify signed stream data with sender public key located in a KeyStore.

The example below checks a signed only stream data against the public keys located in a KeyStore file. Even if none of the public keys can decode the OpenPGP digital signature packet the content of the signed input stream is extracted into a destination decrypted output stream.

```
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.InputStream;
import java.io.OutputStream;

import com.didisoft.pgp.KeyStore;
import com.didisoft.pgp.PGPLib;

public class KeyStoreVerifyStream {
 public static void main(String[] args) throws Exception{
   // create an instance of the KeyStore
   KeyStore keyStore = new KeyStore("pgp.keystore", "changeit");

   // initialize the library
   PGPLib pgp = new PGPLib();

   // obtain the signed stream
   InputStream signedStream = new FileInputStream("signed.pgp");
   // specify the decrypted output stream
   OutputStream decryptedStream = new FileOutputStream("OUTPUT.txt");

   boolean validSignature = pgp.verifyStream(signedStream,
                                             keyStore,
                                             decryptedStream);

   if (validSignature) {
        System.out.println("Signature is valid.");
   } else {
        System.out.println("Signature is invalid!");
   }
  }
}
```

Sign and Encrypt

OpenPGP signing and encrypting in one pass is the most secure way by which we can send encrypted data. We sign the file with our private key and encrypt it with the public key of the person to whom we are sending the message. This way the encrypted file can be read only by the recipient and she can tell with absolute assurance that it was you who sent the message.

The examples below demonstrate in practice how to achieve this with OpenPGP Library for Java.

1) Sign and encrypt file with keys located in files

This example intentionally sets the parameter *withIntegrityCheck* to false, as integrity check information is not supported by PGP (r) 6.5 and prior versions, and if you do not know for sure what OpenPGP software your partners are using, it is better to avoid this feature.

```
import com.didisoft.pgp.PGPLib;

public class SignAndEncryptFile {
```

```java
 public static void main(String[] args) throws Exception{
   // create an instance of the library
   PGPLib pgp = new PGPLib();

   // should output be ASCII or binary
   boolean asciiArmor = false;
   // should integrity check information be added
   boolean withIntegrityCheck = false;

   // sign and encrypt
   pgp.signAndEncryptFile("INPUT.txt",
                          "our_private_key.asc",
                          "private_key_pass_phrase",
                          "recipient_public_key.asc",
                          "encrypted.pgp",
                          asciiArmor,
                          withIntegrityCheck);
 }
}
```

2) Sign and encrypt file with keys located in a KeyStore

This example is equivalent to the above one except that the encryption (public) key of the receiver has been imported to a KeyStore and our secret key has either been generated or imported in the same KeyStore object.

```java
import com.didisoft.pgp.KeyStore;
import com.didisoft.pgp.PGPLib;

public class KeystoreSignAndEncryptFile {
 public static void main(String[] args) throws Exception{
   // create an instance of the KeyStore
   KeyStore keyStore = new KeyStore("pgp.keystore", "changeit");

   // create an instance of the library
   PGPLib pgp = new PGPLib();

   // is output ASCII or binary
   boolean asciiArmor = false;
   // should integrity check information be appended
   boolean withIntegrityCheck = false;

   // The signing key is usually our private key
   String signUserId = "demo@didisoft.com";
   String signKeyPassword = "changeit";
   // the User Id of the recipient, this
   // example assumes her public key is
   // already imported in the KeyStore file
   String encUserId = "recipient@company.com";

   pgp.signAndEncryptFile("INPUT.txt",
                          keyStore,
                          signUserId,
                          signKeyPassword,
                          encUserId,
                          "encrypted.pgp",
                          asciiArmor,
                          withIntegrityCheck);
 }
}
```

3) Sign and encrypt input stream with keys located in files

We can sign and encrypt an input stream source too.

The second parameter of this method is often misunderstood. Although the encrypted data is stored in an output stream,

the OpenPGP data format stores in addition to the encrypted bytes a file name string associated with them, so we are forced to set an additional artificial file name label.

```java
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.InputStream;
import java.io.OutputStream;

import com.didisoft.pgp.PGPLib;

public class SignAndEncryptStream {
 public static void main(String[] args) throws Exception{
   // initialize the library
   PGPLib pgp = new PGPLib();

   // is output ASCII or binary
   boolean asciiOutput = true;
   // should integrity check information be appended
   boolean withIntegrityCheck = true;

   // recipient public key as stream
   InputStream recipientPublicKeyStream =
       new FileInputStream("recipient_public_key.asc");

   // private signing key as stream
   InputStream privateKeyStream =
       new FileInputStream("our_private_key.asc");
   String privateKeyPassword = "changeit";

   // input stream to be encrypted
   InputStream inputStream = new FileInputStream("INPUT.txt");
   // encrypted output destination
   OutputStream encryptedOutStream =
       new FileOutputStream("encrypted.pgp");

   // Here "INPUT.txt" is just a string to be written in
   // the message OpenPGP packet which stored
   // file name label, timestamp, and the actual data bytes
   pgp.signAndEncryptStream(inputStream,
                            "INPUT.txt",
                            privateKeyStream,
                            privateKeyPassword,
                            recipientPublicKeyStream,
                            encryptedOutStream,
                            asciiOutput,
                            withIntegrityCheck);
 }
}
```

4) Sign and encrypt input stream with keys located in a KeyStore

This example is equivalent to the above one, except the keys are located in a KeyStore.

```java
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.InputStream;
import java.io.OutputStream;

import com.didisoft.pgp.KeyStore;
import com.didisoft.pgp.PGPLib;

public class KeyStoreSignAndEncryptStream {
 public static void main(String[] args) throws Exception{
  // create an instance of the KeyStore
  KeyStore keyStore =
```

```java
    new KeyStore("pgp.keystore", "changeit");

  // create an instance of the library
  PGPLib pgp = new PGPLib();

  // is output ASCII or binary
  boolean asciiArmor = true;
  // should integrity check information be added
  boolean withIntegrityCheck = true;

  // The signing key is usually our private key
  String signUserId = "demo@didisoft.com";
  String signKeyPassword = "changeit";
  // the User Id of the recipient, this
  // example assumes her public key is
  // already imported in the KeyStore file
  String encUserId = "recipient@company.com";

  InputStream dataStream = new FileInputStream("INPUT.txt");
  // this parameter is just a label that is associated with
  // the encrypted content in the OpenPGP archive
  String encryptedContentLabel = "INPUT.txt";

  // create the output stream
  OutputStream encryptedStream =
    new FileOutputStream("encrypted.pgp");

  pgp.signAndEncryptStream(dataStream,
                           encryptedContentLabel,
                           keyStore,
                           signUserId,
                           signKeyPassword,
                           encUserId,
                           encryptedStream,
                           asciiArmor,
                           withIntegrityCheck);
 }
}
```

All of the above mentioned examples produce signatures in OpenPGP version 4 format, which is the current format. Some OpenPGP compatible software systems can expect digital signatures in the older version 3 format, although this is a very rare case. To produce version 3 signatures you can either use the examples above and replace the signAndEncrypt methods with the ones that end with Version3 or see the examples from the Examples folder in the library distribution package.

Decrypt and Verify

This example demonstrates how to decrypt a file that has been OpenPGP signed and encrypted in one pass. We can still decrypt it with the standard decrypt method, but with decrypt and verify in one pass we can also verify that the sender of the message is the one that we expect.

In order to run the decrypt and verify method we need our private key (to decrypt the message) and the public key of the sender (to verify the digital signature).

1) Decrypt and verify file with keys located in files on the disk

Note that even if the signature is invalid the contained file will be extracted.

```java
import com.didisoft.pgp.PGPLib;

public class DecryptAndVerify {
 public static void main(String[] args) throws Exception{
  // create an instance of the library
  PGPLib pgp = new PGPLib();
```

```java
    boolean validSignature =
          pgp.decryptAndVerifyFile("encrypted.pgp",
                                  "my_private_key.asc", "private key pass",
                                  "sender_public_key.asc",
                                  "OUTPUT.txt");
  if (validSignature) {
          System.out.println("Signature is valid.");
  } else {
          System.out.println("!Signature is invalid!");
  }
 }
}
```

2) Decrypt file with keys located in a KeyStore

We must specify our password for the private key that will be used to decrypt the file (the library searches for a key with Key ID equal to the one used to encrypt the file). For the verification a public key with Key ID equal to the one in the digital signature is searched among the public keys stored in the KeyStore.

```java
import com.didisoft.pgp.KeyStore;
import com.didisoft.pgp.PGPLib;

public class KeystoreDecryptAndVerifyFile {
 public static void main(String[] args) throws Exception{
  // create an instance of the KeyStore
  KeyStore keyStore = new KeyStore("pgp.keystore", "changeit");

  // initialize the library
  PGPLib pgp = new PGPLib();

  // our private decryption key password
  String privateKeyPassword = "changeit";

  boolean validSignature =
     pgp.decryptAndVerifyFile("encrypted.pgp",
                                  keyStore,
                                  privateKeyPassword,
                                  "OUTPUT.txt");

  if (validSignature) {
          System.out.println("Signature is valid.");
  } else {
          System.out.println("Signature is invalid!");
  }
 }
}
```

3) Decrypt stream with keys supplied as streams

This method gives more freedom on how the encrypted data and OpenPGP keys are stored.

```java
import java.io.FileOutputStream;
import java.io.FileInputStream;
import java.io.InputStream;
import java.io.OutputStream;

import com.didisoft.pgp.PGPLib;

public class DecryptAndVerifyStream {
 public static void main(String[] args) throws Exception{
  // create an instance of the library
  PGPLib pgp = new PGPLib();

  // obtain a signed and encrypted data stream
  InputStream encryptedStream = new FileInputStream("encrypted.pgp");
```

```java
InputStream privateKeyStream = new FileInputStream("private.key");
String privateKeyPassword = "changeit";

InputStream senderPublicKeyStream = new FileInputStream("public.key");

// specify the destination stream of the decrypted data
OutputStream decryptedStream = new FileOutputStream("OUTPUT.txt");

boolean validSignature =
        pgp.decryptAndVerifyStream(encryptedStream,
                               privateKeyStream, privateKeyPassword,
                               senderPublicKeyStream,
                               decryptedStream);
if (validSignature) {
        System.out.println("Signature is valid.");
} else {
        System.out.println("Signature is invalid!");
}
}
}
```

4) Decrypt stream with keys located in a KeyStore

This example is equivalent to the decrypt above, only the encrypted data and the decrypted output are streams.

```java
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.InputStream;
import java.io.OutputStream;

import com.didisoft.pgp.KeyStore;
import com.didisoft.pgp.PGPLib;

public class KeyStoreDecryptAndVerifyStream {
 public static void main(String[] args) throws Exception{
  // create an instance of the KeyStore
  KeyStore keyStore = new KeyStore("pgp.keystore", "changeit");

  // initialize the library
  PGPLib pgp = new PGPLib();

  // our private decryption key password
  String privateKeyPassword = "changeit";

  // obtain an encrypted stream
  InputStream encryptedStream = new FileInputStream("encrypted.pgp");
  // specify the decrypted output stream
  OutputStream decryptedStream = new FileOutputStream("OUTPUT.txt");

  boolean validSignature =
     pgp.decryptAndVerifyStream(encryptedStream,
                               keyStore,
                               privateKeyPassword,
                               decryptedStream);

  if (validSignature) {
        System.out.println("Signature is valid.");
  } else {
        System.out.println("Signature is invalid!");
  }
 }
}
```

Clear sign

The clear signature OpenPGP message format is designed to sign text messages. The original message is kept as is and an additional signature is appended. This way the recipient can still read the original message without special software.

Clear signed messages are verified just like ordinary signed data.

The examples below demonstrate how to achieve this with DidiSoft OpenPGP Library for Java.

1) Clearsign string message with private key located in file

This example shows how to clear sign a text message. The signature algorithm is specified explicitly in contrast to the standard sign method.

```
import com.didisoft.pgp.HashAlgorithm;
import com.didisoft.pgp.PGPLib;

public class ClearSignString {
 public static void main(String[] args) throws Exception{
  // create an instance of the library
  PGPLib pgp = new PGPLib();

  String message = "The quick brown fox jumps.";

  // clear sign
  String clearSignedMessage =
      pgp.clearSignString(message,
                  "private.asc", "private key pass",
                  HashAlgorithm.SHA256);
 }
}
```

2) Clearsign file with private key located in file

This example demonstrates how to clear text sign a file. The result file will contain the original file contents intact and an additional signature.

```
import com.didisoft.pgp.HashAlgorithm;
import com.didisoft.pgp.PGPLib;

public class ClearSignFile {
 public static void main(String[] args) throws Exception{
  // create an instance of the library
  PGPLib pgp = new PGPLib();

  // clear sign
  pgp.clearSignFile("INPUT.txt",
              "private.asc", "private key pass",
              HashAlgorithm.SHA256,
              "OUTPUT.sig.txt");
 }
}
```

Generate RSA keys

This example demonstrates how to generate an RSA based OpenPGP key pair with OpenPGP Library for Java.

The generated keys have no expiration date. An overloaded version exists that accepts expiration time parameter.

```
import com.didisoft.pgp.*;

public class GenerateKeyPairRSA {
 public static void main(String[] args) throws Exception {
  // initialize the KeyStore where the key will be generated
```

```
KeyStore ks = new KeyStore("pgp.keystore", "changeit");

// key primary user Id
String userId = "demo2@didisoft.com";

// preferred hashing algorithms
String[] hashingAlgorithms = new String[]
                               {HashAlgorithm.SHA1,
                                HashAlgorithm.SHA256,
                                HashAlgorithm.SHA384,
                                HashAlgorithm.SHA512,
                                HashAlgorithm.MD5};

// preferred compression algorithms
String[] compressions = new String[]
                               {CompressionAlgorithm.ZIP,
                                CompressionAlgorithm.ZLIB,
                                CompressionAlgorithm.UNCOMPRESSED};

// preferred symmetric key algorithms
String[] cyphers = new String[]
                      {CypherAlgorithm.CAST5,
                       CypherAlgorithm.AES_128,
                       CypherAlgorithm.AES_192,
                       CypherAlgorithm.AES_256,
                       CypherAlgorithm.TWOFISH};

String privateKeyPassword = "changeit";

int keySizeInBytes = 2048;
ks.generateKeyPair(keySizeInBytes,
                   userId,
                   KeyAlgorithm.RSA,
                   privateKeyPassword,
                   compressions,
                   hashingAlgorithms,
                   cyphers);
}
}
```
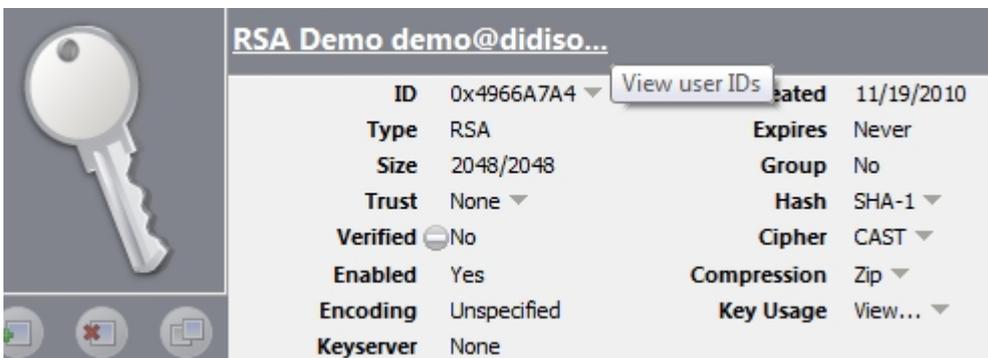
After the key pair is generated usually we will export the public key and send it to our partners.

Below is a screenshot of the generated key properties when we open it with PGP (r) 10:

Generate DH/DSS keys

This example will demonstrate how to generate an OpenPGP key pair compatible with the Diffie Hellman algorithm that is recognized by PGP (r) version 10 and all OpenPGP standard compatible software systems as DH/DSS type key.

```
import com.didisoft.pgp.*;

public class GenerateKeyPairDHDSS {
```

```java
public static void main(String[] args) throws Exception {
 // initialize the KeyStore where the key will be generated
 KeyStore ks = new KeyStore("pgp.keystore", "changeit");

 // key primary user Id
 String userId = "demo@didisoft.com";

 // preferred hashing algorithms
 String[] hashingAlgorithms = new String[]
                        {HashAlgorithm.SHA1,
                         HashAlgorithm.SHA256,
                         HashAlgorithm.SHA384,
                         HashAlgorithm.SHA512,
                      HashAlgorithm.MD5};

 // preferred compression algorithms
 String[] compressions = new String[]
                        {CompressionAlgorithm.ZIP,
                         CompressionAlgorithm.ZLIB,
                         CompressionAlgorithm.UNCOMPRESSED};

 // preferred symmetric key algorithms
 String[] cyphers = new String[]
                        {CypherAlgorithm.AES_128,
                         CypherAlgorithm.AES_192,
                         CypherAlgorithm.AES_256,
                         CypherAlgorithm.CAST5,
                      CypherAlgorithm.TWOFISH};

 String privateKeyPassword = "changeit";

 int keySizeInBytes = 2048;
 ks.generateKeyPair(keySizeInBytes,
                        userId,
                        KeyAlgorithm.ELGAMAL,
                        privateKeyPassword,
                        compressions,
                        hashingAlgorithms,
                        cyphers);
 }
}
```

After the key is generated it can be exported in a standalone file and imported into another OpenPGP software.

You may notice that the key algorithm parameter is ELGAMAL. The ElGamal is an implementation of the Diffie Hellman algorithm and the key is accepted with no complains from PGP (r) 10. The screenshot below shows the key properties for the exported public key in PGP (r) 10.

Although we have requested the key size to be 2048 bits the DSS (digital signature standards) signing sub key is 1024 bits length. The explanation is that we use DSA (digital signature algorithm) to produce the signing sub key and it is limited to 1024 bits.

Import keys

This example demonstrates how to import existing keys from standalone files into a KeyStore object.

```java
import com.didisoft.pgp.KeyStore;

public class ImportKeys {
 public static void main(String[] args) throws Exception {
  // initialize the KeyStore. The key store file may not exist
  // and subsequent operations will create it
  KeyStore keyStore = new KeyStore("pgp.keystore", "changeit");

  // import private key
  keyStore.importPrivateKey("private.asc", "changeit");

  // import public key
  keyStore.importPublicKey("public.asc");

  // imports key ring file. The file may contain public, private or
  // both type of keys if it is in ASCII armored format
  keyStore.importKeyRing("keypair.asc");
 }
}
```

Export keys

This example will demonstrate how to export a key from a KeyStore file. This is an operation that usually follows after we generate a key pair and have to send the public key to our partners so they could encrypt data that only we will be able to read.

```java
import com.didisoft.pgp.KeyStore;

public class ExportKeys {
 public static void main(String[] args) throws Exception{
  // initialize the key store
  KeyStore keyStore = new KeyStore("pgp.keystore", "changeit");

  // should output be ASCII or binary
  boolean asciiArmor = false;

  // export public key
  keyStore.exportPublicKey("exported_pub.pkr",
                "demo@didisoft.com",
                asciiArmor);
  // export private key
  keyStore.exportPrivateKey("exported_sec.skr",
                 "demo@didisoft.com",
                 asciiArmor);
  // export both keys in one file (in this case
  // only ASCII armored output is accepted)
  keyStore.exportKeyRing("keypair.asc", "demo@didisoft.com");
 }
}
```

Delete keys

This example demonstrates how to delete a key pair from an OpenPGP KeyStore.

```java
import com.didisoft.pgp.KeyStore;

public class DeleteKeyPair {
 public static void main(String[] args) throws Exception{
        // initialize the KeyStore instance
        KeyStore ks = new KeyStore("pgp.keystore", "changeit");
```

```
        // delete the key pair
        ks.deleteKeyPair("test@gmail.com");
 }
}
```

Change password

The example below demonstrates how to change the password of a private key that has been imported in a KeyStore file.

We should know either the User Id of the key or the key Id; this example uses the method that accepts User Id.

```
import com.didisoft.pgp.KeyStore;

public class ChangePrivateKeyPassword {
 public static void main(String[] args) throws Exception{
  // initialize the KeyStore instance
  KeyStore ks = new KeyStore("pgp.keystore", "changeit");

  // change secret key password
  String keyUserId = "test@gmail.com";
  String oldPassword = "changeit";
  String newPassword = "new_private_key_password";
  ks.changePrivateKeyPassword(keyUserId, oldPassword, newPassword);
 }
}
```

Eventually we can export it lately if we prefer to keep it in a standalone file.

Cipher

By default all encrypt methods use CAST 5 as an internal symmetric key encryption algorithm. This can be changed through a setter method PGPLib.setCypher(string cypher). The accepted values that can be passed for the cypher parameter are listed in the CypherAlgorithm interface:

CypherAlgorithm.TRIPLE_DES;
CypherAlgorithm.String CAST5;
CypherAlgorithm.String BLOWFISH;
CypherAlgorithm.String AES_128;
CypherAlgorithm.String AES_192;
CypherAlgorithm.String AES_256;
CypherAlgorithm.String TWOFISH;
CypherAlgorithm.String DES;
CypherAlgorithm.String SAFER;
CypherAlgorithm.String IDEA;

Here is how to set the default symmetric algorithm to 128 bit AES :

PGPLib pgp = new PGPLib();
pgp.setCypher(CypherAlgorithm.AES_128);

Each subsequent call to encrypt or sign and encrypt will use the new algorithm, but only for the current instance of the library.

Have in mind that if the public key we use for encryption does not support the preferred symmetric algorithm, then the first cypher from this key preferred list will be used (each OpenPGP key stores internally it's algorithm preferences).

Hashing

Hashing is used in digital signatures. By default DidiSoft OpenPGP Library for Java uses SHA1. To set another hashing algorithm to be used for signing, we have to change the default one with the method PGPLib.setHash(hash). The possible values are listed in the interface HashAlgorithm:

HashAlgorithm.SHA1
HashAlgorithm.SHA256

HashAlgorithm.SHA384
HashAlgorithm.SHA512
HashAlgorithm.SHA224
HashAlgorithm.MD5
HashAlgorithm.MD2
HashAlgorithm.RIPEMD160

The change will have effect on subsequent calls to all *sign* and *signAndEncrypt* methods for the current PGPLib instance. For each new instance it has to be set explicitly again. This example shows how to set the hashing algorithm to SHA 256:

```java
import com.didisoft.pgp.HashAlgorithm;
import com.didisoft.pgp.PGPLib;

public class SetHashDemo {
  public static void main(String[] args) throws Exception {
        PGPLib pgp = new PGPLib();
        pgp.setHash(HashAlgorithm.SHA256);
 }
}
```

Compresion

By default DidiSoft OpenPGP Library for Java uses ZIP compression when encrypting and signing data.

This can be changed through the method *PGPLib.setCompression*(compression). The supported options are listed in the *CompressionAlgorithm* interface:

CompressionAlgorithm.ZIP
CompressionAlgorithm.ZLIB
CompressionAlgorithm.BZIP2
CompressionAlgorithm.UNCOMPRESSED

The change has effect on subsequent calls to the encryption methods only on the current instance of the PGPLib object. For each new instance the preferred compression should be set explicitly.

The example below shows how to set the compression to ZLib:

```java
import com.didisoft.pgp.CompressionAlgorithm;
import com.didisoft.pgp.PGPLib;

public class SetCompressionDemo {
  public static void main(String[] args) throws Exception {
        PGPLib pgp = new PGPLib();
        pgp.setCompression(CompressionAlgorithm.ZLIB);
     // each subsequent encryption will use ZLib
  }
}
```

Example files

In the library distribution ZIP package you will find a folder named **Examples.**

Create a new Java project with your favorite IDE of choice and add the files from the **Examples\src** folder.

The other files in the Examples folder are used as data for some of the examples.

Online

All the examples below are available online at our web site:
http://www.didisoft.com/examples/java-openpgp-examples/

Quick introduction to OpenPGP
Getting Started with the library

Most common functions

Encrypt
Decrypt
Sign
Verify
Sign and Encrypt
Decrypt and Verify
Clear text sign

KeyStore and key generation.

Generate RSA keys
Generate DH/DSS keys
Import keys
Export keys

Key revocation
Introduction to OpenPGP key revocation
Revoke key directly
Revocation certificate
Designated revoker

Advanced Topics

Set preferred cypher (symmetric key algorithm)
Set preferred compression
Set preferred hashing

Appendix

Common Exceptions

Some common exceptions that may occur when working with the library are:

org.bouncycastle.openpgp.PGPException: Exception creating cipher

org.bouncycastle.openpgp.PGPException: exception constructing public key

org.bouncycastle.openpgp.PGPException: exception encrypting session key

java.lang.SecurityException: Unsupported keysize or algorithm parameters

Resolution:
Try to download the files listed in [JCE Unlimited Strength Policy files](#) and try again.

If the problem appears after that, please [contact us](#).

Exporting keys from a GnuPG keystore

List keys contained in the GnuPG keystore:
gpg --list-keys

Export Public key
gpg --export my_key -o my_public_key.gpg

Export Private key
gpg --export-secret-key my_key -o my_secret_key.gpg

Support

Technical support

To receive general information or **technical support**, please contact us at
support@didisoft.com.

Sales

For questions related to sales, volume licensing, or OEM licensing, please contact us at
sales@didisoft.com.

Product Updates

If you have purchased the library you can access our [Customers' Portal](#) where you can download new versions.

Newsletter

To receive product update news, you can subscribe to our [Newsletter](#)

For further information, visit us at www.didisoft.com
If you have any ideas, wishes, questions or criticism, don't hesitate to contact us. We will be glad to hear from you.