GLOBAL 2007

Обучение Java

Тема **Безопасность в JAVA**

Часть **Симметричное шифрование**

Автор **ASKIL (omendba@gmail.com)**

3.03.2007

Важным средством обеспечения безопасности является *шифрование*. Информация, отмеченная цифровой подписью, доступна для просмотра, а подпись лишь подтверждает, что эта информация не была изменена. Для просмотра зашифрованных данных этого недостаточно и их нужно расшифровать с помощью соответствующего ключа.

Часто бывает так, что апплет или приложение содержат конфиденциальную информацию, например номера кредитных карточек или другие личные данные, которые необходимо скрыть от посторонних. Именно в таких случаях используются технологии шифрования.

До недавнего времени во многих компаниях, включая Sun, существовали патентные и экспортные ограничения на использование мощных алгоритмов шифрования, счастью, сегодня все эти ограничения утратили силу, а срок патентного ограничения на использование алгоритма RSA и вовсе истек в октябре 2000 года. В стандартной библиотеке пакета JDK 1.4 предусмотрены достаточно мощные средства шифрования. Для более старых версий JDK предусмотрено расширение JCE, обеспечивающее поддержку шифрования.

Криптографическое расширение Java содержит класс Cipher, который является суперклассом всех классов, имеющих отношение к шифрованию. Для создания объекта, реализующего алгоритм шифрования, используется метод getInstance().

Cipher cipher = Chiper.getInstance (algorithName);

Cipher cipher = Chiper.getInstance (algorithName, providerName);

В JDK для всех шифров используется провайдер Sun JCE. Если имя провайдера не указано явно, то по умолчанию принимается имя SunJCE. Если вы хотите воспользоваться алгоритмами, которые не поддерживаются инструментами Sun, следует указать другого провайдера.

Имя алгоритма задается в виде строки, например "DES" или "DES/CBC/PKCS5Padding".

Стандарт DES (Data Encryption Standard) определяет очень старый алгоритм шифрования с длиной ключа в 56 бит. В настоящее время данный алгоритм не используется потому, что найдена простая возможность его взлома. Намного лучше использовать расширенный алгоритм шифрования AES.

Метода doFinal() необходим для дополнения (padding) блока данных, Например, в алгоритме шифрования DES блок имеет размер 8 байт. Предположим, что размер входных данных меньше чем 8 байт. Если заполнить остальные байты нулями и зашифровать блок, то при расшифровке блок будет отличаться от исходного. Поэтому разработаны специальные схемы дополнения. Одна из таких схем описана в стандарте Public Key Cryptography Standard (PKCS) #5, созданном специалистами RSA Security Inc. В этой схеме блок дополняется не нулями, а числами, равными недостающему количеству байтов.

Если же длина зашифрованных входных данных делится без остатка на 8, то в конце добавляется блок

08 08 08 08 08 08 08 08

При расшифровке последний байт декодированных данных определяет количество байтов, которые следует удалить.

Рассмотрим процедуру создания симметричного ключа. Даная процедура по выбранному алгоритму создает ключ и записывает его в файл.

```
/**
 * Создание симметричного ключа.
 * @param alg <B>Алгоритм ключа</B>
 * @param filekey <B>Файл с ключом</B>
 */
public void CreateSymmetricKey(String alg, String filekey)
{
  try
  {
   KeyGenerator keygen = KeyGenerator.getInstance(alg);
   SecureRandom random = new SecureRandom();
   keygen.init(random);
   SecretKey sk = keygen.generateKey();
   byte [] bytekey = sk.getEncoded();
   FileOutputStream out = new FileOutputStream(filekey);
   out.write(bytekey);
   out.close();
  }
  catch (NoSuchAlgorithmException e){GetError(e.getMessage());}
  catch (IOException e){GetError(e.getMessage());}
}
```

При выполнении кода:

```
CreateSymmetricKey("Blowfish","c:/key");
```

Получаем файл, содержимое которого будет выглядеть следующим образом:

```
7834 230A 4DA1 4B48
A87B 93C4 802D 9E17
```

То есть размер ключа алгоритма Blowfish равен 16 байт.

Процедура чтения ключа с диска:

```
/**
 * Прочитать симметричный ключ из файла.
 * @param alg <B>Алгоритм ключа</B>
 * @param filekey <B>Файл с ключом</B>
 * @return Секретный ключ
 */
public SecretKeySpec ReadSymmetricKey(String alg, String filekey)
{
  try
  {
    FileInputStream out = new FileInputStream(filekey);
    byte[] bytekey = new byte [out.available()];
    out.read(bytekey);
    return new SecretKeySpec(bytekey, alg);
  }
  catch (IOException e){GetError(e.getMessage()); return null;}
}
```

Вызов данной процедуры будет выглядеть следующим образом:

```
SecretKeySpec sks = ReadSymmetricKey("Blowfish","c:/key");
```

Процедура шифрования файла по симметричному ключу будет иметь вид:

```
/**
 * Шифрование файла по секретному ключу.
 * @param infile <B>Файл для шифрации</B>
 * @param outfile <B>Зашифрованный файл</B>
 * @param sks <B>Секретный ключ</B>
 */
public void FileEnCrypt(String infile, String outfile, SecretKeySpec sks)
{
  try
  {
   Cipher cipher = Cipher.getInstance(sks.getAlgorithm());
   cipher.init(Cipher.ENCRYPT_MODE,sks);
   FileInputStream in = new FileInputStream(infile);
   FileOutputStream out = new FileOutputStream(outfile);
   byte [] bytein = new byte [cipher.getBlockSize()]; //Блок для шифрования
   byte [] cryptbyte; //Блок шифрованных данных
   boolean flag = false; //Флаг окончания процесса "Чтение-шифрование"
   int bytefromfile=0;
   while (!flag)
   {
     bytefromfile = in.read(bytein); //Прочитали блок файла
     if (bytefromfile!=cipher.getBlockSize() ||
         bytefromfile == -1) flag=true; //Данные закончились - выходим
     else
     {
       cryptbyte = cipher.update(bytein,0,cipher.getBlockSize());
       out.write(cryptbyte);
     }
   }
   if (bytefromfile>0)
       cryptbyte = cipher.doFinal(bytein,0,bytefromfile);
   else
       cryptbyte = cipher.doFinal();
   out.write(cryptbyte);
   out.close();
   in.close();
  }
  catch (NoSuchAlgorithmException e){GetError(e.getMessage());}
  catch (InvalidKeyException e){GetError(e.getMessage());}
  catch (NoSuchPaddingException e){GetError(e.getMessage());}
  catch (IOException e){GetError(e.getMessage());}
  catch (IllegalBlockSizeException e){GetError(e.getMessage());}
  catch (BadPaddingException e){GetError(e.getMessage());}
}
```

Процедура дешифрования файла по симметричному ключу будет иметь вид:

```
/**
 * Дешифрование файла по секретному ключу.
 * @param infile <B>Зашифрованный файл</B>
 * @param outfile <B>Дешифрованный файл</B>
 * @param sks <B>Секретный ключ</B>
 */
public void FileDeCrypt(String infile, String outfile, SecretKeySpec sks)
{
 try
 {
```

```
      Cipher cipher = Cipher.getInstance(sks.getAlgorithm());
      cipher.init(Cipher.DECRYPT_MODE,sks);
      FileInputStream in = new FileInputStream(infile);
      byte [] bytein = new byte [cipher.getBlockSize()];
      FileOutputStream out = new FileOutputStream(outfile);
      byte [] decryptbyte;
      while (in.read(bytein)!=-1)
      {
        decryptbyte = cipher.update(bytein,0,cipher.getBlockSize());
        out.write(decryptbyte);
      }
      decryptbyte = cipher.doFinal();
      out.write(decryptbyte);
      out.close();
      in.close();
    }
    catch (NoSuchAlgorithmException e){GetError(e.getMessage());}
    catch (InvalidKeyException e){GetError(e.getMessage());}
    catch (NoSuchPaddingException e){GetError(e.getMessage());}
    catch (IOException e){GetError(e.getMessage());}
    catch (IllegalBlockSizeException e){GetError(e.getMessage());}
    catch (BadPaddingException e){GetError(e.getMessage());}
   }
```

Теперь просмотрим, как работают данные процедуры.

```
SecretKeySpec sks = ReadSymmetricKey("Blowfish","c:/key");
FileEnCrypt("c:/test.txt","c:/crypt",sks);
FileDeCrypt("c:/crypt","c:/test1.txt",sks);
```

Пусть имеется файл c:/test.txt. Для алгоритма Blowfish размер блока равен *cipher.getBlockSize()=8*. То есть шифрация/дешифрация идет по 8 байт.

Пусть тестовый файл заполнен значениями:

```
1234567
```

Тогда шифрованный файл будет иметь вид:

```
DB4B 8AEC 2F88 AA32
```

То есть произошло дополнение до 8 байт. При значении:

```
12345678
```

шифрованный файл будет иметь вид:

```
D94F 880C 10E1 6C4D
11E5 BC09 9A0F 5B6F
```

Размер конечного шифрованного файла будет рассчитываться по формуле:

$$L = (\text{ЦелаяЧасть}(\text{ДлинаФайла}/\text{РазмерБлока})+1) \cdot \text{РазмерБлока}$$

Пусть размер начального файла равен 25 байт, то размер зашифрованного файла:

$$L = (\text{ЦЧ}(25/8)+1)\cdot 8=(3+1)\cdot 8=32$$

что совпадает с измеряемым результатом.

Для шифрования можно использовать шифрующие потоки. В библиотеке JCE содержится набор классов, реализующих потоки. Эти классы отличаются тем, что в процессе записи и чтения автоматически происходит шифрование и расшифровка информации.

При использовании классов, реализующих шифрующие потоки, обращение к методам update() и doFinal() остается прозрачным для программиста.

Вот процедуры дешифрации и шифрации файлов с помощью данного механизма.

```
    /**
     * Дешифрование файла по секретному ключу.
     * @param infile <B>Зашифрованный файл</B>
     * @param outfile <B>Дешифрованный файл</B>
     * @param sks <B>Секретный ключ</B>
     */
    public void FileStreamDeCrypt(String infile, String outfile, SecretKeySpec sks)
    {
     try
     {
       Cipher cipher = Cipher.getInstance(sks.getAlgorithm());
       cipher.init(Cipher.DECRYPT_MODE,sks);
       FileInputStream in = new FileInputStream(infile);
       FileOutputStream out = new FileOutputStream(outfile);
       CipherInputStream cin = new CipherInputStream(in, cipher);
       byte[] buffer = new byte[cipher.getBlockSize()];
       int flag = cin.read(buffer); //Узнали, сколько байт прочитано из потока
       while (flag!= -1)
       {
          out.write(buffer,0,flag); //Сколько прочитали, столько и записали
          flag = cin.read(buffer);
       }
       cin.close();
       out.close();
       in.close();
     }
     catch (NoSuchAlgorithmException e){GetError(e.getMessage());}
     catch (InvalidKeyException e){GetError(e.getMessage());}
     catch (NoSuchPaddingException e){GetError(e.getMessage());}
     catch (IOException e){GetError(e.getMessage());}
    }
```

```
    /**
     * Шифрование файла по секретному ключу.
     * @param infile <B>Файл для шифрации</B>
     * @param outfile <B>Зашифрованный файл</B>
     * @param sks <B>Секретный ключ</B>
     */
    private void FileStreamEnCrypt(String infile, String outfile, SecretKeySpec sks)
    {
     try
     {
        Cipher cipher = Cipher.getInstance(sks.getAlgorithm());
        cipher.init(Cipher.ENCRYPT_MODE,sks);
        FileInputStream in = new FileInputStream(infile);
        FileOutputStream out = new FileOutputStream(outfile);
        CipherOutputStream cout = new CipherOutputStream(out, cipher);
        byte [] buffer = new byte[cipher.getBlockSize()];
        int bytefromfile = in.read(buffer); //Узнали, сколько байт прочитано из потока
        while (bytefromfile!= -1)
        {
           cout.write(buffer,0,bytefromfile); //Сколько прочитали, столько и записали
           bytefromfile = in.read(buffer);
        }
```

```
        cout.close(); //Шифруем последние байты, число которых <
                      //размера cipher.getBlockSize()
        in.close();
        out.close();
      }
      catch (NoSuchAlgorithmException e){GetError(e.getMessage());}
      catch (NoSuchPaddingException e){GetError(e.getMessage());}
      catch (InvalidKeyException e){GetError(e.getMessage());}
      catch (IOException e){GetError(e.getMessage());}
     }
```

Для шифрации/дешифрации строк можно поступить проще, т.к. их размер не сможет вызвать ошибку переполнения в машине Java. Эти процедуры выглядят следующим образом:

```
    /**
     * Шифрование строк по секретному ключу.
     * @param data <B>Строка для шифрования</B>
     * @param alg <B>Алгоритм</B>
     * @param sks <B>Секретный ключ</B>
     * @return Шифрованные байты
     */
    public byte [] StringEnCrypt(String data, String alg, SecretKeySpec sks)
    {
     try
     {
      Cipher cipher = Cipher.getInstance(alg);
      cipher.init(Cipher.ENCRYPT_MODE,sks);
      byte [] bytedata = new byte [data.length()];
      for (int i=0;i<data.length();i++)
         bytedata[i] = (byte)data.charAt(i);
      byte [] bcipher = cipher.doFinal(bytedata);
      return bcipher;
     }
     catch(NoSuchAlgorithmException e){GetError(e.getMessage()); return null;}
     catch(NoSuchPaddingException e){GetError(e.getMessage()); return null;}
     catch(InvalidKeyException e){GetError(e.getMessage()); return null;}
     catch(IllegalBlockSizeException e){GetError(e.getMessage()); return null;}
     catch(BadPaddingException e){GetError(e.getMessage()); return null;}
    }
```

```
    /**
     * Дешифрование строк по секретному ключу.
     * @param key <B>Секретный ключ</B>
     * @param decryptdata <B>Байты для дешифрования</B>
     * @param alg <B>Алгоритм</B>
     * @return Расшифрованная строка
     */
    public String StringDeCrypt(byte [] decryptdata, String alg, SecretKeySpec key)
    {
     try
     {
      Cipher cipher = Cipher.getInstance(alg);
      cipher.init(Cipher.DECRYPT_MODE,key);
      byte [] bcipher = cipher.doFinal(decryptdata);
      return new String(bcipher);
     }
     catch(NoSuchAlgorithmException e){GetError(e.getMessage()); return null;}
     catch(NoSuchPaddingException e){GetError(e.getMessage()); return null;}
     catch(InvalidKeyException e){GetError(e.getMessage()); return null;}
     catch(IllegalBlockSizeException e){GetError(e.getMessage()); return null;}
     catch(BadPaddingException e){GetError(e.getMessage()); return null;}
    }
```

Результат:

```
String s = "Hello java in 2007";
String alg = "Blowfish";
SecretKeySpec sks = ReadSymmetricKey(alg,"c:/key");
System.out.println(StringDeCrypt(StringEnCrypt(s,alg,sks),alg,sks));
```

```
run:
Hello java in 2007
```