GLOBAL 2007

Обучение Java

Тема **Безопасность в JAVA**

Часть **Шифрование с открытым ключом**

Автор **ASKIL (omendba@gmail.com)**

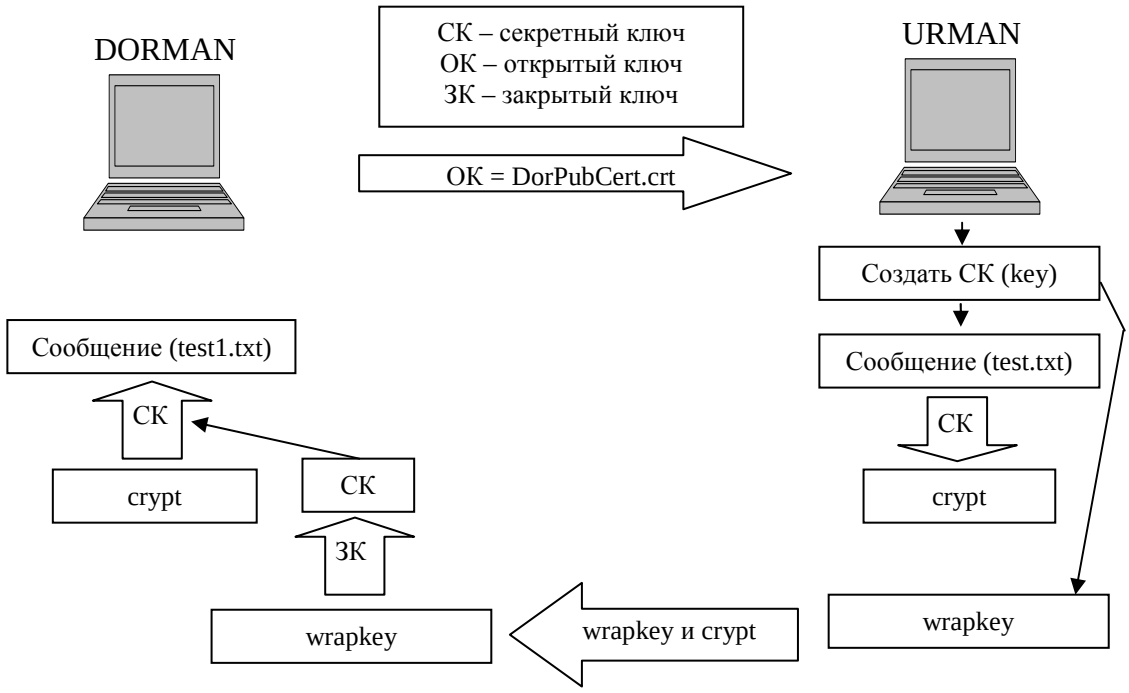
4.03.2007

В симметричном алгоритме для шифрования и расшифровки используется один и тот же ключ. Уязвимым местом симметричного алгоритма шифрования является передача ключа.

Для решения этой проблемы можно использовать шифрование с открытым ключом. В таком случае у DORMAN есть пара ключей: открытый и соответствующий ему закрытый ключ. Он может передавать открытый ключ, но закрытый ключ должен хранить в тайне. Теперь URMAN для шифрования сообщений DORMAN просто использует открытый ключ.

На самом деле не все так просто, потому что все алгоритмы с открытым ключом выполняются намного медленнее, чем алгоритмы с симметричными ключами, такими, как DES или AES. Поэтому было бы неэффективно и непрактично использовать открытые ключи для шифрования данных большого объема. Эту проблему легко решить, комбинируя шифрование с открытым ключом и симметричное шифрование. Рассмотрим следующий пример.

Наиболее популярным алгоритмом с открытым ключом является RSA, изобретенный Райвестом (Rivest), Шамиром (Shamir) и Эдлеманом (Adelman). До октября 2000 года этот алгоритм был защищен патентом RSA Security Inc. Сегодня данный алгоритм уже не является коммерческим и поддерживается в JDK 5.0 и более поздних реализациях.



URMAN захотел передать сообщение DORMAN. Для этого URMAN создает секретный ключ для шифрования сообщения. Процедура создания секретного ключа:

```
    /**
     * 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();
      ObjectOutputStream out = new ObjectOutputStream(new FileOutputStream(filekey));
      out.writeObject(sk); //Сериализация секретного ключа в файл
      out.close();
     }
     catch (NoSuchAlgorithmException e){GetError(e.getMessage());}
     catch (IOException e){GetError(e.getMessage());}
    }
```

URMAN для шифрования использует Blowfish алгоритм и сохраняет секретный ключ в файле key.

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

После выполнения кода на диске C: появится файл key размером 146 байт:

```
ACED 0005 7372 001F 6A61 7661 782E 6372 7970 746F  ....sr..javax.crypto
2E73 7065 632E 5365 6372 6574 4B65 7953 7065 635B  .spec.SecretKeySpec[
470B 66E2 3061 4D02 0002 4C00 0961 6C67 6F72 6974  G.f.0aM...L..algorit
686D 7400 124C 6A61 7661 2F6C 616E 672F 5374 7269  hmt..Ljava/lang/Stri
6E67 3B5B 0003 6B65 7974 0002 5B42 7870 7400 0842  ng;[..keyt..[Bxpt..B
6C6F 7766 6973 6875 7200 025B 42AC F317 F806 0854  lowfishur..[B......T
E002 0000 7870 0000 0010 9D59 DFC9 27A6 57DE 4BA2  ....xp.....Y..'.W.K.
0212 9AE4 ED29                                     .....)
```

Это не ключ, а сериализованный объект, то есть объект-ключ, который можно восстановить в память из файла.

URMAN будет использовать данный объект-ключ для шифрования сообщений.

Для того чтобы использовать данный ключ URMAN должен десериализовать его из файла. Процедура десериализации ключа в память будет иметь вид:

```
    /**
     * Прочитать симметричный ключ из файла.
     * @param alg <B>Алгоритм ключа</B>
     * @param filekey <B>Файл с ключом</B>
     * @return Секретный ключ
    */
    public SecretKey ReadSymmetricKey(String alg, String filekey)
    {
```

```
      try
      {
        ObjectInputStream in = new ObjectInputStream(new FileInputStream(filekey));
        SecretKey sk = (SecretKey) in.readObject();
        in.close();
        return sk;
      }
      catch (ClassNotFoundException e){GetError(e.getMessage()); return null;}
      catch (IOException e){GetError(e.getMessage()); return null;}
     }
```

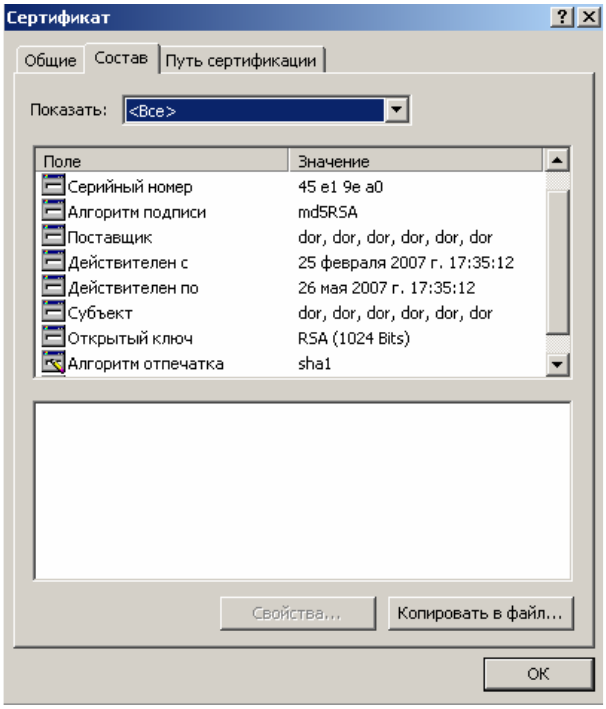
Получение секретного ключа будет иметь вид:

```
SecretKey sk = ReadSymmetricKey("Blowfish","c:/key");
```

Сообщение, которое хочет передать URMAN находится файле c:/test.txt и его содержимое:

```
Blowfish algorithm in Java
```

DORMAN отправляет открытый ключ URMAN. Он содержится в сертификате DorPubCert.crt.



Процедура получения открытого ключа будет иметь вид:

```
/**
  * Получить самоподписанный сертификат из файла.
  * @param crtfile <B>Файл с сертификатом</B>
  * @return Сертификат
  */
public X509Certificate GetCert(String crtfile)
{
  try
  {
   FileInputStream in = new FileInputStream(crtfile);
   CertificateFactory cf = CertificateFactory.getInstance("X509");
   X509Certificate x509 = (X509Certificate) cf.generateCertificate(in);
```

```
   in.close();
   return x509;
  }
  catch(java.security.cert.CertificateException e)
  {
    GetError(e.getMessage()); return null;
  }
  catch(FileNotFoundException e){GetError(e.getMessage()); return null;}
  catch(IOException e){GetError(e.getMessage()); return null;}
}
```

Получение открытого ключа будет иметь вид:

```
    PublicKey pubk = GetCert("c:/DorPubCert.crt").getPublicKey();
```

Сообщение шифруется с помощью процедуры, которая описывалась в предыдущих документах и выглядит следующим образом:

```
    SecretKeySpec sks = new SecretKeySpec(sk.getEncoded(),"Blowfish");
    FileStreamEnCrypt("c:/test.txt","c:/crypt",sks);
```

После этого получим шифрованный файл crypt размером 32 байт:

```
6FC3 7F07 7AC7 9B79|o...z..y
01D9 9E39 D89A 0B33|...9...3
B958 680B 490F 7845|.Xh.I.xE
D7AC A4FC 35DD ACEB|....5...
```

Осталось зашифровать симметричный ключ URMAN с помощью открытого ключа от DORMAN. Этим занимается процедура ниже:

```
   /**
    * Зашифровать симметричный ключ.
    * @param pubkey <B>Открытый ключ</B>
    * @param sk <B>Секретный ключ</B>
    * @param wrapfile <B>Зашифрованный симметричный ключ в файле</B>
    * @param alg <B>Алгоритм шифрования</B>
    */
   public void WrapSymmetricKey(String alg, PublicKey pubkey, SecretKey sk,
                                String wrapfile)
   {
    try
    {
     Cipher cipher = Cipher.getInstance(alg);
     cipher.init(Cipher.WRAP_MODE,pubkey);
     byte [] wrapkey = cipher.wrap(sk);
     FileOutputStream fos = new FileOutputStream(wrapfile);
     fos.write(wrapkey);
     fos.close();
    }
    catch(Exception e){GetError(e.getMessage());}
   }
```

URMAN ее вызывает так:

```
WrapSymmetricKey("RSA",pubk,sk,"c:/wrapkey");
```

После этого создается файл wrapkey, размер которого равен 128 байт:

```
4BF3 985E 85B3 F92D D0DB 31C8 FCB4 957B|K..^...-..1....{
0D28 8EDC 9148 B364 ED90 A896 B492 34A7|.(...H.d......4.
E5B5 4879 1A8A 66C2 9808 1F99 6326 8BED|..Hy..f.....c&..
D53B 7564 0D87 8700 9068 896F 5506 85DE|.;ud.....h.oU...
D999 E477 4285 89DA D4A1 208F 1EA4 FFC2|...wB..... .....
8D30 D148 B26F 47A6 30EF 2733 2B45 1E4F|.0.H.oG.0.'3+E.O
7575 832B 04F6 CA9D 48F9 38E2 53CA 27CE|uu.+....H.8.S.'.
2814 5AA2 171A A61A 92ED 44F4 1BDA 721B|(.Z.......D...r.
```

Теперь URMAN отправляет эти два файла (crypt и wrapkey) DORMAN.

DORMAN, получив два файла должен иметь доступ к своему закрытому ключу. Следующая процедура решает эту задачу:

```
/**
 * Получить закрытый ключ.
 * @param ks <B>Хранилище ключей</B>
 * @param alias <B>Алиас</B>
 * @param pass <B>Пароль</B>
 * @return Закрытый ключ
 */
public PrivateKey GetPrivateKey(KeyStore ks, String alias, char [] pass)
{
 try
 {
  return (PrivateKey) ks.getKey(alias,pass);
 }
 catch(KeyStoreException e){GetError(e.getMessage()); return null;}
 catch(NoSuchAlgorithmException e){GetError(e.getMessage()); return null;}
 catch(UnrecoverableKeyException e){GetError(e.getMessage()); return null;}
}
```

Также ему необходим доступ к своему хранилищу ключей, в котором содержится данный закрытый ключ. Решение:

```
/**
 * Получить хранилище ключей.
 * @param filekeystore <B>Файл хранилища</B>
 * @param c <B>Пароль</B>
 * @return База ключей
 */
public KeyStore GetKeyStore(String filekeystore, char [] c)
{
 try
 {
  KeyStore ks = KeyStore.getInstance("JCEKS");
  InputStream in = new FileInputStream(filekeystore);
  ks.load(in,c);
  Arrays.fill(c,'0'); //Сразу же обнуляем пароль
  return ks;
 }
 catch(IOException e){GetError(e.getMessage()); return null;}
 catch(KeyStoreException e){GetError(e.getMessage()); return null;}
 catch(NoSuchAlgorithmException e){GetError(e.getMessage()); return null;}
 catch(CertificateException e){GetError(e.getMessage()); return null;}
}
```

DORMAN вызывает процедуры:

```
KeyStore ks = GetKeyStore("c:/DkeyStore","PASSWORD1".toCharArray());
PrivateKey prk = GetPrivateKey(ks,"DorAlias","PASSWORD1".toCharArray());
```

Затем он должен расшифровать симметричный ключ wrapkey. Следующая процедура решает эту проблему:

```
/**
 * Расшифровать симметричный ключ.
 * @param prkey <B>Закрытый ключ</B>
 * @param algsymmetric <B>Алгоритм шифрования симметричного ключа</B>
 * @param wrapfile <B>Зашифрованный симметричный ключ в файле</B>
 * @param alg <B>Алгоритм шифрования</B>
 * @return Симметричный ключ
 */
public SecretKey UnWrapSymmetricKey(String alg, PrivateKey prkey,
                                    String wrapfile, String algsymmetric)
{
 try
 {
  FileInputStream fin = new FileInputStream(wrapfile);
  byte [] unwrapkey = new byte [fin.available()];
  fin.read(unwrapkey);
  Cipher cipher = Cipher.getInstance(alg);
  cipher.init(Cipher.UNWRAP_MODE,prkey);
  return (SecretKey)cipher.unwrap(unwrapkey,algsymmetric,Cipher.SECRET_KEY);
 }
 catch(Exception e){GetError(e.getMessage()); return null;}
}
```

DORMAN вызывает следующие методы и получает расшифрованное сообщение:

```
SecretKey sk = UnWrapSymmetricKey("RSA",prk,"c:/wrapkey","Blowfish");
SecretKeySpec sks= new SecretKeySpec(sk.getEncoded(),"Blowfish");
FileStreamDeCrypt("c:/crypt","c:/test1.txt",sks);
```

Если не использовать сертификаты, то сначала необходимо сгенерировать открытый и закрытый ключ в памяти, затем десериализовать их. Процедура генерации будет иметь следующий вид:

```
      SecureRandom random = new SecureRandom();
      KeyPairGenerator kpg = KeyPairGenerator.getInstance("RSA");
      kpg.initialize(1024,random); //1024 - число бит
      KeyPair kp = kpg.generateKeyPair();
      Key pubkey = kp.getPublic();
      Key prkey = kp.getPrivate();
```