**JAVA 加解密总结 (一)**

## 1、Java 安全体系架构

### 1.1、Java 安全领域组成部分

**JCA**(Java Cryptography Architecture,Java加密体系结构)

提供基本的加密框架，如证书、数字签名、消息摘要和密钥对产生器，不限制出口。

**JCE**(Java Cryptography Extension,Java加密扩展包)

JCE在JCA的基础上面做了扩展，提供了各种加密算法、消息摘要算法、和密钥管理功能，常用的算法有（DES,AES,RSA,DSA），对出口做了限制。

**JSSE**(Java Secure Sockets Extension,Java安全套接字扩展包)

**JAAS**(Java Authentication and Authentication Service,Java鉴别与安全服务)

JCA和JCE是Java平台提供的用于安全和加密接口的两组API。他们并不执行任何算法，只是连接应用和实际算法程序之间的一组接口。

密钥长度通常和加密强度成正比，加密在军事中有着重要的意义，所以我们使用java jdk时会受到出口限制的影响。

### 1.2、安全提供者

软件开发商根据JCE接口（有称安全提供者接口）将各种算法实现后，打包成一个Provider（安全提供者），动态加载到Java的运行环境之中。Bouncy Castle JCE是一组开源免费的安全提供者。

Provider和Security类共同构成了安全提供者的概念

### 1.3、Provider和Security类

**Provider**类实现了Java安全性的一部分或者全部，我们称之为提供者。

在JAVA安全体系中，安全提供者可能不是来自sun本身，不同的算法可能需要不同的提供者提供，甚至提供者本身属于第三方

BouncyCastleProvider使用第三方开源组织[www.bouncycastle.org](http://www.bouncycastle.org)来提供的

**Security**类管理Java程序中使用到的提供者

添加一个提供者：

|  |
| --- |
| **static** {  Security.*addProvider*(**new** BouncyCastleProvider());  } |

查看系统提供的所有提供者：

|  |
| --- |
| **for** (Provider provider : Security.*getProviders*()) {  System.***out***.println(provider);  **for** (Map.Entry<Object, Object> entry : provider.entrySet()) {  System.***out***.println("\t"+entry.getKey());  }  } |

### 1.4 密钥相关

#### 1.4.1 Key接口

所有密钥的顶层接口，一切与加密有关的操作都离不开Key接口。包含：算法名称getAlgorithm()，编码形式：getEncoded()，格式:getFormat()。

SecretyKey对称密钥顶层接口

PublicKey和PrivateKey非对称密钥顶层接口

#### 1.4.2 KeyPair接口

非对称密钥的扩展，密钥对得载体，通常由KeyPairGenerator的generateKeyPair()来提供。

#### 1.4.3 KeyFactory和SecretKeyFactory类

**KeyFactory**生成密钥（公钥和私钥）的引擎类，通过密钥规范来还原密钥。与之对应的是**SecretKeyFactory**类，用来生成私钥

#### 1.4.4 KeyGenerator和KeyPairGenerator类

**KeyGenerator**用来生成私密密钥

Java 7 版本 中 提供 了 Blowfish、 AES、 DES 和 DESede 等 多种 对称 加密 算法 实现， 以及 HmacMD5、 HmacSHA1 和 HmacSHA256 等 多种 安全 消息 摘要 算法 实现。

**KeyPairGenerator**用来生成密钥对

### 1.5 加解密

#### 1.5.1 Cipher

1. Cipher提供加密和解密功能，是JCE框架的核心。

实例化：算法名/工作模式/填充模式

1. Cipher提供了对密钥的包装和解包功能

### 1.6 密钥规范和算法参数规范

实现在java. security. spec 包 和 javax. crypto. spec接口中

#### 1.6.1 KeySpec

所有密钥规范都必须实现此接口

KeySpec 的 抽象 实现 类（ EncodedKeySpec） 构建 了 用于 构建 公 钥 规范 和 私 钥 规范 的 两个 实现（ **X509EncodedKeySpec** 用于 构建 公 钥 规范， **PKCS8EncodedKeySpec** 用于 构建 私 钥 规范）。

|  |
| --- |
| /\*\*  \* 使用私钥进行解密  \* **@param** data  \* **@param** key  \* **@return**  \* **@throws** Exception  \*/  **public** **static** **byte** [] decryptByPrivateKey(**byte** [] data, **byte** [] key) **throws** Exception {  PKCS8EncodedKeySpec pkcs8KeySpec = **new** PKCS8EncodedKeySpec(key);  KeyFactory keyFactory = KeyFactory.*getInstance*(***KEY\_ALGORITHM***);  PrivateKey privateKey = keyFactory.generatePrivate(pkcs8KeySpec);  Cipher cipher = Cipher.*getInstance*(keyFactory.getAlgorithm());  cipher.init(Cipher.***DECRYPT\_MODE***, privateKey);  **return** cipher.doFinal(data);  } |

|  |
| --- |
| /\*\*  \* 使用公钥进行解密  \* **@param** data  \* **@param** key  \* **@return**  \* **@throws** Exception  \*/  **public** **static** **byte** [] decryptByPublicKey(**byte** [] data, **byte** [] key) **throws** Exception {  X509EncodedKeySpec x509KeySpec = **new** X509EncodedKeySpec(key);  KeyFactory keyFactory = KeyFactory.*getInstance*(***KEY\_ALGORITHM***);  PublicKey publicKey = keyFactory.generatePublic(x509KeySpec);  Cipher cipher = Cipher.*getInstance*(keyFactory.getAlgorithm());  cipher.init(Cipher.***DECRYPT\_MODE***, publicKey);  **return** cipher.doFinal(data);  } |

**SecretKeySpec**用于构建私密密钥规范，兼容所有对称加密的算法，可以直接new

DESKeySpec只针对DES算法…

|  |
| --- |
| // 实例 化 SecretKey 对象。  SecretKey secretKey = new SecretKeySpec( key, "RC2"); |

#### 1.6.2 AlgorithmParameterSpec

所有参数规范都必须实现此接口

## 2、常用算法及其实现

### 2.1、Base64与Hex

UrlBase64 专门为url中数据传输进行准备的。

### 2.2、消息摘要算法

消息摘要算法又称散列算法， 其核心在于散列函数的单向性。 即通过散列 函数 可获得 对应 的 散 列 值， 但不 可通过 该 散 列 值 反推 其 原始 信息。 这是 消息 摘要 算法 的 安全 性的 根本 所在。

消息摘要算法主要分为三大类：MD（ Message Digest， 消息 摘要 算法）、 SHA- 1（ Secure Hash Algorithm， 安全散列算法） 和 Hmacmd5（ Message Authentication Code， 消息认证

#### 2.2.1 一般消息摘要算法

**1、使用jdk自带的进行实现**

|  |
| --- |
| /\*\*  \* 计算字符串的md5值  \* **@param** str  \* **@return**  \*/  **public** **static** String md5(String str) {    **try** {  **byte** [] strBytes = str.getBytes();  MessageDigest digest = MessageDigest.*getInstance*("MD5");  digest.update(strBytes);  **byte** [] md = digest.digest();  **return** *toHexString*(md);  } **catch** (Exception e) {  e.printStackTrace();  }  **return** **null**;  }    /\*\*  \* 计算文件的md5值  \* linux: md5sum filename  \* **@param** file  \* **@return**  \*/  **public** **static** String md5(File file) {  **if** (file == **null**) **return** **null**;  **try** {  MessageDigest digest = MessageDigest.*getInstance*("MD5");  @SuppressWarnings("resource")  FileInputStream fis = **new** FileInputStream(file);  FileChannel fc = fis.getChannel();    MappedByteBuffer buffer = fc.map(FileChannel.MapMode.***READ\_ONLY***, 0, file.length());  digest.update(buffer);  **byte** [] md = digest.digest();  **return** *toHexString*(md);  } **catch** (Exception e) {  e.printStackTrace();  }  **return** **null**;  } |

**2、使用apache commons codec**

DigestUtils.md5(input);

#### 2.2.2 安全消息摘要算法

Mac

|  |
| --- |
| /\*\*  \* mac消息摘要算法  \* **@throws** Exception  \*/  **public** **static** **void** mac() **throws** Exception {  **byte** [] input = "MAC".getBytes();  //  KeyGenerator keyGenerator = KeyGenerator.*getInstance*("HmacMD5");  SecretKey secretKey = keyGenerator.generateKey();    Mac mac = Mac.*getInstance*(secretKey.getAlgorithm());  mac.init(secretKey);    **byte** [] result = mac.doFinal(input);    System.***out***.println(**new** String(Hex.*encode*(result)));  } |

### 2.3、对称加密算法

#### 2.3.1 DES（初等加密标准）

#### 2.3.2 AES（高级加密标准）

详见AESCoder.java

### 2.4、非对称加密算法

#### 2.4.1 RSA

详见RSACoder.java

公钥加密，私钥解密

私钥加密，公钥解密

### 2.5、AES+RSA生产场合常用demo

使用AES对数据进行加密

使用RSA对AES的密钥进行加密

客户端保存RSA的公钥

服务器端保存RSA的私钥

客户端使用公钥对AES得密钥加密后连同使用AES加密的数据一起发送的服务器端

服务器端先使用私钥对加密后的私钥密钥进行解密得到AES密钥

服务器端再使用解密得到的AES密钥解密上传的数据，得到上传的数据

反过来同上！！！



实现demo。

详见CryptServlet.java

## 3、常见问题总结