视频位置：P:\00\_传智播客\黑马程序员训练营\_公开课视频教程\中关村黑马程序员训练营\_公开课视频教程\_Java安全

# 对称加密与非对称加密

对称加密：加密速度快

非对称加密：相比对称加密算法上要更复杂，所以加密速度慢

非对称加密应用场景：多个人可以加密，但是只有一个人有解密。比如我们访问工商银行，数据传给工商银肯定需要加密的，假设工商银行和用户A、用户B分别约定了两个密码，用户A用密码一加密后把数据传工商银行，用户B用密码二加密后把数据传给工商银行，银行使用相应的密码进行解密，如果此时来了用户C，银行又得跟用户C商量好一个密码，如果有一万个用户呢？这时，使用非对称加密就能解决问题：工商银行把公钥放在网站上，所有人都使用这个公钥进行加密，工商银行使用私钥进行解密。

密钥类型：对称加密的密码称为密钥，非对称加密的有公钥和私钥

Key：SecretKey，PublicKey，PrivateKey

## 对称加密示例

对称加密也叫密钥加密，英文中对应的密钥类是：SecretKey

**public** **class** Demo {

**public** **static** **void** main(String[] args) **throws** Exception {

// createKey();

// encript();

*decript*();

}

/\*\* 创建密钥并保存起来，方便下次使用 \*/

**private** **static** **void** createKey() **throws** Exception {

Key key = KeyGenerator.*getInstance*("DES").generateKey();// 创建密钥用于DES算法的密钥

ObjectOutputStream oos = **new** ObjectOutputStream(**new** FileOutputStream("EvenDai.key"));

oos.writeObject(key);

oos.close();

}

/\*\* 读取密钥 \*/

**private** **static** Key readKey() **throws** Exception {

ObjectInputStream ois = **new** ObjectInputStream(**new** FileInputStream("EvenDai.key"));

Key key = (Key) ois.readObject();

ois.close();

**return** key;

}

/\*\* 加密 \*/

**private** **static** **void** encript() **throws** Exception {

Cipher cipher = Cipher.*getInstance*("DES"); // 创建DES算法的加密对象

cipher.init(Cipher.***ENCRYPT\_MODE***, *readKey*()); // 初始化为加密模式

String data = "好好学习，天天向上！"; // 要加密的数据

**byte**[] encriptBytes = cipher.doFinal(data.getBytes()); // 进行加密

System.***out***.println(**new** String(encriptBytes)); // 显示加密后的数据

// 保存加密后的数据

FileOutputStream fos = **new** FileOutputStream("encript.data");

fos.write(encriptBytes);

fos.close();

}

**private** **static** **void** decript() **throws** Exception {

Cipher cipher = Cipher.*getInstance*("DES"); // 创建DES算法的加密对象

cipher.init(Cipher.***DECRYPT\_MODE***, *readKey*()); // 初始化为加密模式

// 读取加密的数据

ByteArrayOutputStream baos = **new** ByteArrayOutputStream();

FileInputStream fis = **new** FileInputStream("encript.data");

**byte**[] buf = **new** **byte**[1024];

**int** len;

**while**((len = fis.read(buf)) != -1) {

baos.write(buf, 0, len);

}

fis.close();

**byte**[] decriptBytes = cipher.doFinal(baos.toByteArray());// 进行解密

System.***out***.println(**new** String(decriptBytes)); // 显示解密后的数据

}

}

上面代码生产的Key，就相当于是银行的U盾 ，即U盘里的证书就是Key。

还有另一种加密的方式是使用密码，如银行卡取款的密码，这时不能使用DES算法了，要用另一种叫“**PBEWithMD5AndDES**”的算法，其中的PBE代表“password-based encryption（基于密码的加密）”，代码和上面基本相同，需要改变的地方只有两处如下：

1、Key的创建：

注：这里的密码要求必须是8位长度，否则化抛异常

**private** **static** Key createKey(String password) **throws** Exception {

KeySpec keySpec = **new** PBEKeySpec(password.toCharArray());

SecretKey key = SecretKeyFactory.*getInstance*("PBEWithMD5AndDES").generateSecret(keySpec);

**return** key; // 这时的key没有必要保存了，我们只要记得自己设置的password字符串密码即可。

}

2、Cipher类的创建：

上面的PBEParameterSpec就是传说中的加盐，解密的时候也得加这个参数。

PBEParameterSpec parameterSpec = **new** PBEParameterSpec(**new** **byte**[]{1,2,3,4,5,6,7,8},1000);

Cipher cipher = Cipher.*getInstance*("PBEWithMD5AndDES"); // 创建DES算法的加密对象

cipher.init(Cipher.***ENCRYPT\_MODE***, *createKey*("12345678"), parameterSpec); // 初始化为加密模式

## 非对称加密

一般都是公钥加密，私钥解密。代码与对称加密的方式基本相同，不同的是算法改成RSA，密钥的生成使用KeyPairGenerator来生成公钥和私钥，加密用公钥，解密就用私钥。

**public** **class** Demo {

**public** **static** **void** main(String[] args) **throws** Exception {

// createKey();

*encript*();

*decript*();

}

/\*\* 创建密钥并保存起来，方便下次使用 \*/

**private** **static** **void** createKey() **throws** Exception {

KeyPairGenerator generator = KeyPairGenerator.*getInstance*("RSA");

KeyPair keyPair = generator.generateKeyPair();

PublicKey publicKey = keyPair.getPublic();

PrivateKey privateKey = keyPair.getPrivate();

*saveKey*(publicKey, "publicKey.key");

*saveKey*(privateKey, "privateKey.key");

}

**private** **static** **void** saveKey(Key key, String keyFileName) **throws** Exception {

ObjectOutputStream oos = **new** ObjectOutputStream(**new** FileOutputStream(keyFileName));

oos.writeObject(key);

oos.close();

}

/\*\* 读取密钥 \*/

**private** **static** Key readKey(String keyFileName) **throws** Exception {

ObjectInputStream ois = **new** ObjectInputStream(**new** FileInputStream(keyFileName));

Key key = (Key) ois.readObject();

ois.close();

**return** key;

}

/\*\* 加密 \*/

**private** **static** **void** encript() **throws** Exception {

Cipher cipher = Cipher.*getInstance*("RSA"); // 创建DES算法的加密对象

cipher.init(Cipher.***ENCRYPT\_MODE***, *readKey*("publicKey.key")); // 初始化为加密模式

String data = "好好学习，天天向上！"; // 要加密的数据

**byte**[] encriptBytes = cipher.doFinal(data.getBytes()); // 进行加密

System.***out***.println(**new** String(encriptBytes)); // 显示加密后的数据

// 保存加密后的数据

FileOutputStream fos = **new** FileOutputStream("encript.data");

fos.write(encriptBytes);

fos.close();

}

**private** **static** **void** decript() **throws** Exception {

Cipher cipher = Cipher.*getInstance*("RSA"); // 创建DES算法的加密对象

cipher.init(Cipher.***DECRYPT\_MODE***, *readKey*("privateKey.key"));// 初始化为加密模式

// 读取加密的数据

ByteArrayOutputStream baos = **new** ByteArrayOutputStream();

FileInputStream fis = **new** FileInputStream("encript.data");

**byte**[] buf = **new** **byte**[1024];

**int** len;

**while**((len = fis.read(buf)) != -1) {

baos.write(buf, 0, len);

}

fis.close();

**byte**[] decriptBytes = cipher.doFinal(baos.toByteArray());// 进行解密

System.***out***.println(**new** String(decriptBytes)); // 显示解密后的数据

}

}

注：cipher.update方法在RSA加密中不起作用，RSA加密只能使用cihper.doFinal(byte[] data)方法

CipherInputStream和CipherOutputStream是用于在读或写的时候就解密或者解密，就看会传入的chipher对象的初始化模式，如果设置为解密，则CipherInputStream用于把加密的文件读进来，读进来就解密了，而CipherOutputStream用于把加密的数据写到文件，写的时候就解密了，也就是说这两个流这时都是用在解密：具体例子如下：

**private** **static** **void** decript() **throws** Exception {

Cipher cipher = Cipher.*getInstance*("RSA"); // 创建DES算法的加密对象

cipher.init(Cipher.***DECRYPT\_MODE***, *readKey*("privateKey.key"));// 初始化为加密模式

FileInputStream fis = **new** FileInputStream("encript.data");

CipherInputStream cis = **new** CipherInputStream(fis, cipher); // 从这个流中读取的内容就是解密的

// 读取加密的数据

ByteArrayOutputStream baos = **new** ByteArrayOutputStream();

**byte**[] buf = **new** **byte**[1024];

**int** len;

**while**((len = cis.read(buf)) != -1) {

baos.write(buf, 0, len);

}

cis.close();

fis.close();

System.***out***.println(**new** String(baos.toByteArray()));

FileInputStream fis2 = **new** FileInputStream("encript.data");

FileOutputStream fos2 = **new** FileOutputStream("unencript.data");

CipherOutputStream cos = **new** CipherOutputStream(fos2, cipher); // 用这个流写出去的内容就是解密的

**while**((len = fis2.read(buf)) != -1) {

cos.write(buf, 0, len);

}

cos.close();

fos2.close();

fis2.close();

}

# 数字摘要

* **消息摘要是一种算法：无论原始数据多长，消息摘要的结果都是固定长度的；原始数据任意bit位的变化，都会导致消息摘要的结果有很大的不同，且根据结果推算出原始数据的概率极低。消息摘要可以看作原始数据的指纹，指纹不同则原始数据不同。**

比如MD5，给MD5传入一堆二进制数据，这些数据无论多少（如1K、1M、1G），通过Md5算法对这些二进制进行计算，计算的结果长度是不固定不变，都是128位，即16个字节。只要传进来的数据有一点点变化，则算出来的结果就大相径庭。既然结果总是128位，能表示的结果组合是有限的，而数据是无限的，所以不同数据产生的MD5结果还是有可能相同，但是这是很难碰得到相同的情况的，可以略等于不会有相同的，比如人的指纹是不可能有一样的，如果哪一天出现了一个罪犯，他的指纹和你一样，然后你被抓了，警察认为你就是罪犯把你枪毕了也不怨枉，你太幸运，活该！

SHA产生的数字摘要是20个字节，所以相比MD5要更安全。

MD5不是加密，因为通过MD5结果是不能恢复出原始数据的。

MD5产生数字摘要示例：