技术18期：数据安全之加密与实现

原创

前言：大数据时代，每个人的生活中都不存在所谓的绝对“秘密”，通过网络上的数据信息可以分析出一个人生活的各种痕迹。因此，保障大数据信息安全至关重要。

本文主要介绍了**散列算法、对称加密算法**和**非对称加密算法**的概念和代码实现。

数据安全的重要性

1. 数据保密性

数据只能由授权实体存取、识别，放置非授权泄露，即数据不能被未授权的第三方使用 。

2. 数据完整性

防止非授权实体对数据进行非法篡改,即数据在传输过长中不能被未授权方修改。

3. 数据可用性

数据对于授权实体是可用的，有效的。

什么是数据加密？

数据加密的核心是**密码学**，指通过加密算法和加密密钥将明文的文件或者数据转变为密文，而解密则是通过解密算法和解密密钥将密文恢复为明文。

通过对数据的加密、解密有效地提高企业应用的安全性，数据加密可以在企业应用中的多个环节实现。比如：可对机要数据进行加密后再存储，对用户的口令加密后存储的等。

加密算法介绍

加密算法一般划分为：**对称加密**和**非对称加密算法**；除此之外还有一种是**散列（Hash）算法**。严格上讲：HASH 算法是一种消息摘要算法，不是一种加密算法，但由于其单向运算，具有一定的不可逆性，成为加密算法中的一个构成部分，完整的加密机制不能仅依赖 HASH 算法。

三种类别的加密算法详见下表：

对比项	散列算法	对称加密	非对称加密
说明	消息摘要算法，不需要密钥，从明文到密文的不可逆的映射,只有加密过程，没有解密过程 特点：是一种单向算法	加密和解密使用同一个密钥 特点：加密密钥=解密密钥	加密和解密所使用的不是同一个密钥，通常有两个密钥，称为“公钥”和“私钥”，它们两个必需配对使用。 特点：非对称加密是公钥加密，私钥来解密
优点	正向快速：给定明文和hash算法，在有限时间和有限资源内能计算出hash 值。	计算量小，加密速度快，加密效率高；适合大量数据运算	安全性好，加密解密速度慢，适合少量数据加密
安全性	盐是固定的，写死在程序中，一旦泄露就不安全了。	算法公开，不取决于算法，取决于加密密钥的传递	安全性好，取决于私钥的保存
常见算法	MD5、SHA、MAC	DES、3DES、IDEA、AES、TDEA、SM4等	RSA、ECC（移动设备用）、SM2（基于ECC）、DSA（数字签名用）等

应用建议：在实际的操作过程中，我们通常采用的方式是采用非对称加密算法管理对称算法的密钥，然后用对称加密算法加密数据，这样我们就集成了两类加密算法的优点，既实现了加密速度快的优点，又实现了安全方便管理密钥的优点。

常见算法速度、安全性等详见下表：

散列算法比较

名称	安全性	速度
SHA-1	高	慢
MD5	中	快

对称加密算法(加解密密钥相同)

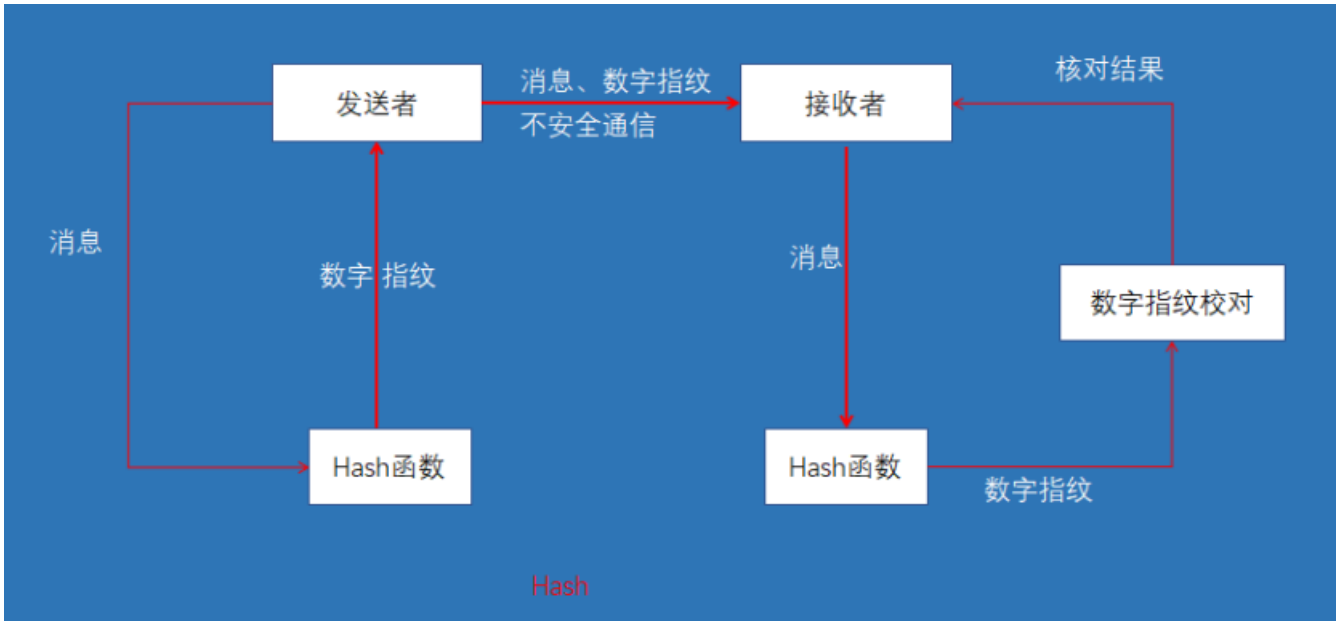
名称	密钥长度	运算速度	安全性	资源消耗
DES	56位	较快	低	中
3DES	112位或168位	慢	中	高
AES	128、192、256位	快	高	低

非对称算法(加密密钥和解密密钥不同)

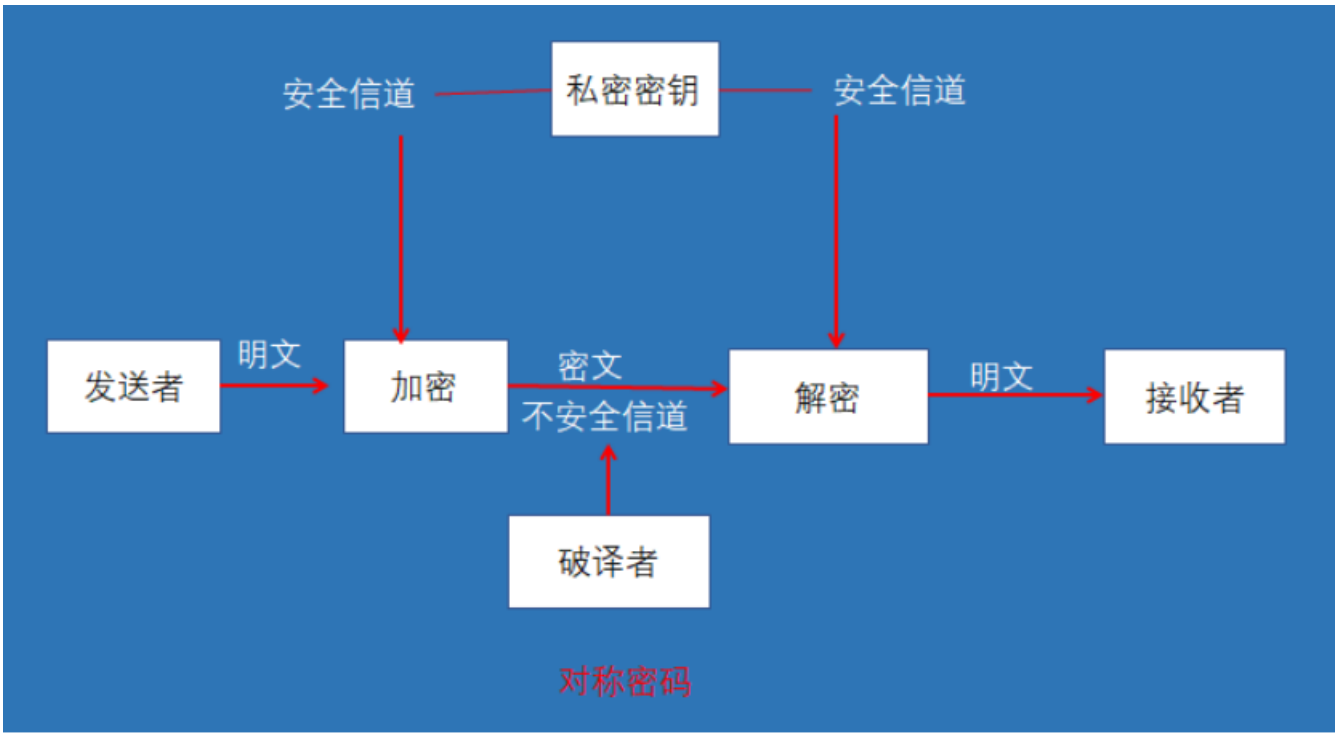
名称	成熟度	安全性(取决于密钥长度)	运算速度	资源消耗
RSA	高	高	慢	高
DSA	高	高	慢	只能用于数字签名
ECC	低	高	快	低(计算量小,存储空间占用小,带宽要求低)

加密算法消息通信过程

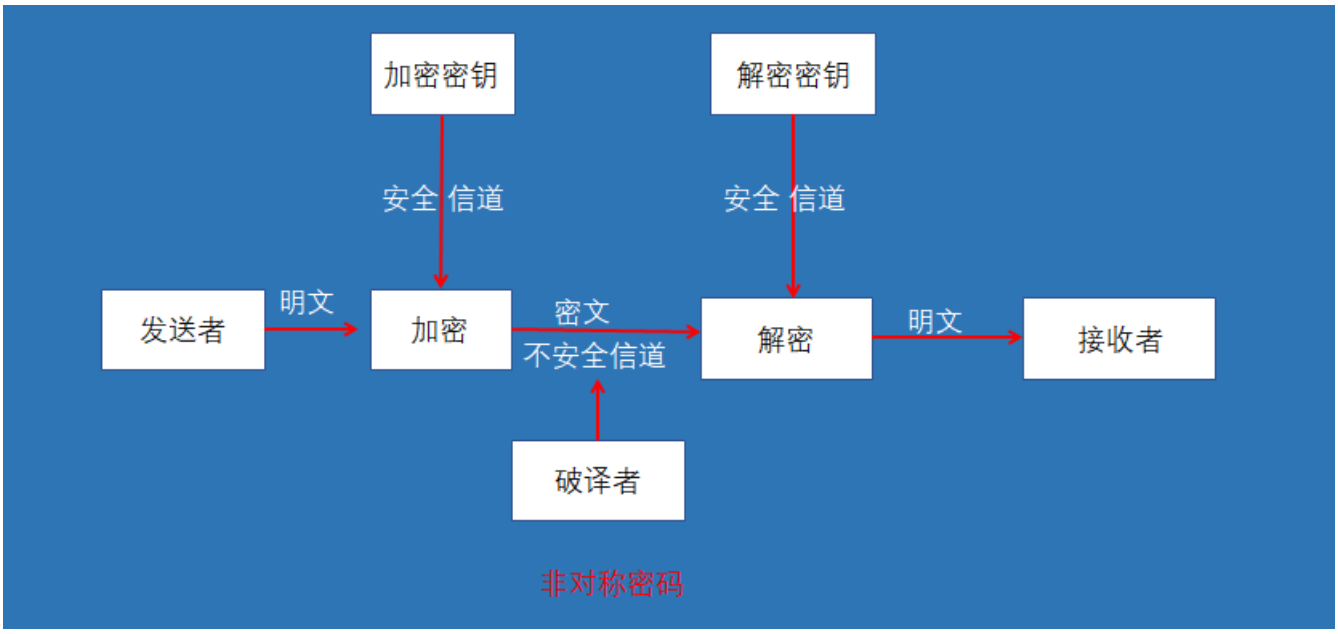
1、Hash算法



2、对称加密算法



3、非对称加密算法



加密实现

一、Hash算法

1、java自带的MessageDigest实现加密，实现步骤如下：

1) 获取指定摘要算法，参数可以输入MD5、SHA等

MessageDigest md=MessageDigest.getInstance("MD5");

2) 计算md5函数

md.update(byte str);或md.update(byte[] str);

3) 获取消息摘要结果

md.digest()

代码示例如下：

```
String pwd="AjiaoNiu13";
MessageDigest md5=MessageDigest.getInstance("MD5");
try {
    md5.update(pwd.getBytes( charsetName: "utf-8"));
    byte[] bytes=md5.digest();
    BASE64Encoder base64en = new BASE64Encoder();
    String newPwd=base64en.encode(bytes);
    System.out.println(newPwd);
} catch (UnsupportedEncodingException e) {
    e.printStackTrace();
}
```

2、java自带的Mac实现加密，实现步骤如下：

1) 产生密钥

algorithm算法，比如：HmacMD5、HmacSHA1

SecretKey secretKey =KeyGenerator.getInstance(algorithm).generateKey();

2) 实列MAC

Mac mac = Mac.getInstance(secretKey.getAlgorithm());

3) 执行摘要

mac.doFinal(pwd.getBytes());

代码示例如下：

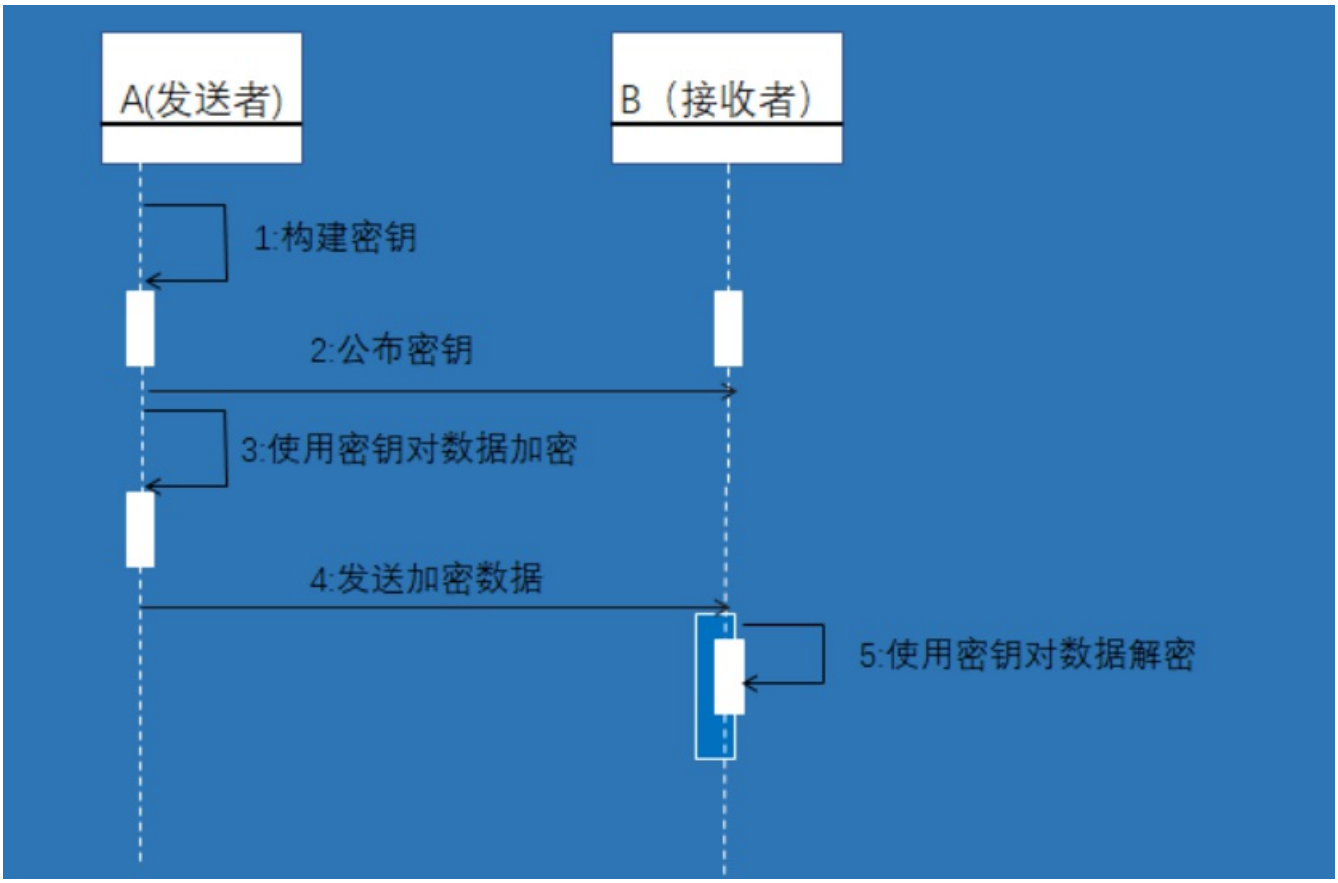
```
private static  String  encryptHMAC(String pwd) throws Exception {
    //产生密钥
    SecretKey secretKey = KeyGenerator.getInstance("HmacMD5").generateKey();
    byte[] key = secretKey.getEncoded();
    //还原来的key  SecretKey secretKey1=new SecretKeySpec(key,algorithm);
    //获取实列Mac
    Mac mac = Mac.getInstance(secretKey.getAlgorithm());
    //初始化MAC
    mac.init(secretKey);
    //执行消息摘要
    byte[] bytes=mac.doFinal(pwd.getBytes());
    BASE64Encoder base64en = new BASE64Encoder();
    return  base64en.encode(bytes);

}
```

二、对称加密算法

1、消息传递模型

消息传递模型步骤如下：



2、java自带的Cipher实现

实现步骤如下：

（1） 密钥构建，代码示例如下：

```java
//算法是DES
public static final String KEY_ALGORITHM = "DES";
/**
 * 加密/解密算法/工作模式
 */
public static final String CIPHER_ALGORITHM ="DES/ECB/PKCS5Padding";

private  static  byte[] initKey() throws NoSuchAlgorithmException {
    KeyGenerator keyGenerator=KeyGenerator.getInstance(KEY_ALGORITHM);
    keyGenerator.init( i: 56);
    SecretKey secretKey=keyGenerator.generateKey();
    return  secretKey.getEncoded();
}
```

（2）生成密钥，代码示例如下：

```java
/**
 * 转换密钥
 * @param key
 * @return
 * @throws InvalidKeyException
 * @throws NoSuchAlgorithmException
 * @throws InvalidKeySpecException
 */
private static Key tokey(byte[] key) throws InvalidKeyException, NoSuchAlgorithmException,
    //实例化DES密钥材料
    DESKeySpec desKeySpec= new DESKeySpec(key);
    //实例化密钥工厂
    SecretKeyFactory keyFactory= SecretKeyFactory.getInstance(KEY_ALGORITHM);
    //生成密钥返回
    return keyFactory.generateSecret(desKeySpec);
}
```

（3） 加密实现，实现步骤如下：

1) 实例化Cipher

2) 初始化

3) 设置加密模式

4) 执行操作

代码示例如下：

```java
private static byte[] encrypt(byte[] data,byte[] key) throws
    //还原key
    Key k= DESCoder.tokey(key);
    //实例化
    Cipher cipher =Cipher.getInstance(CIPHER_ALGORITHM);
    //初始化，设置加密模式
    cipher.init(Cipher.ENCRYPT_MODE,k);
    return  cipher.doFinal(data);
}
```

（4）解密实现，实现步骤如下：

1) 实例化Cipher

2) 初始化

3) 设置解密模式

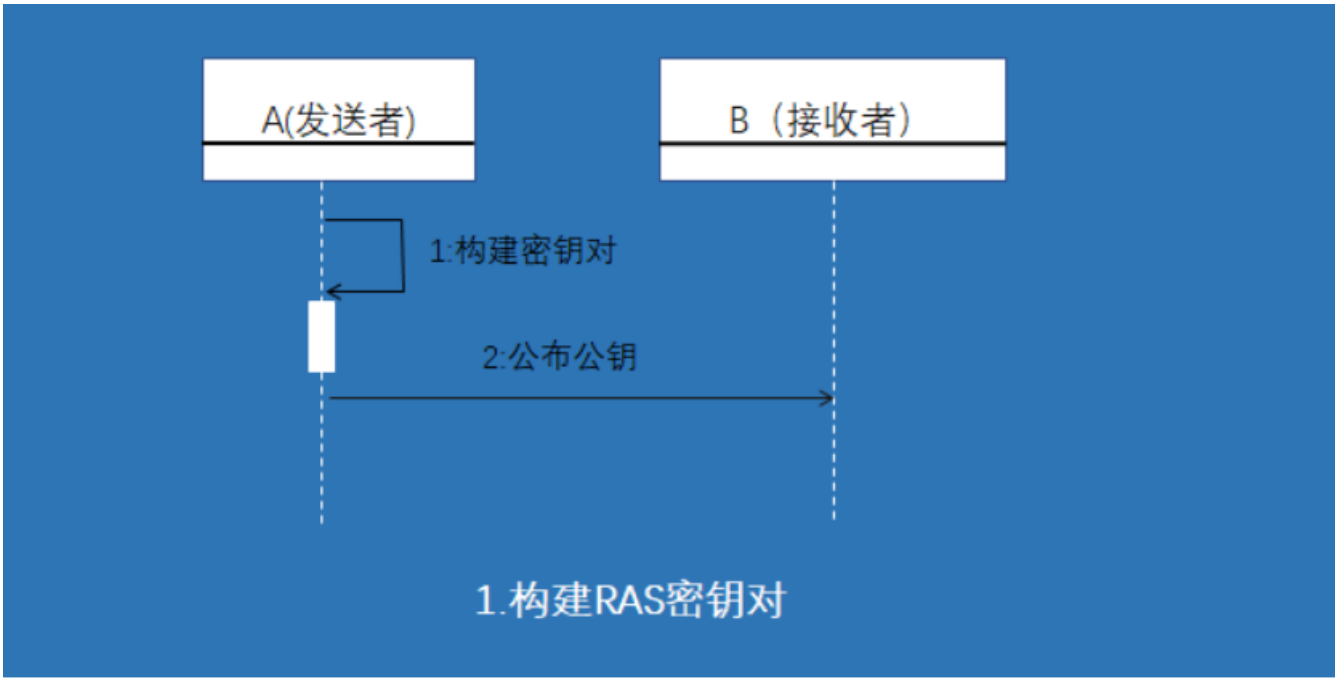
4) 执行操作

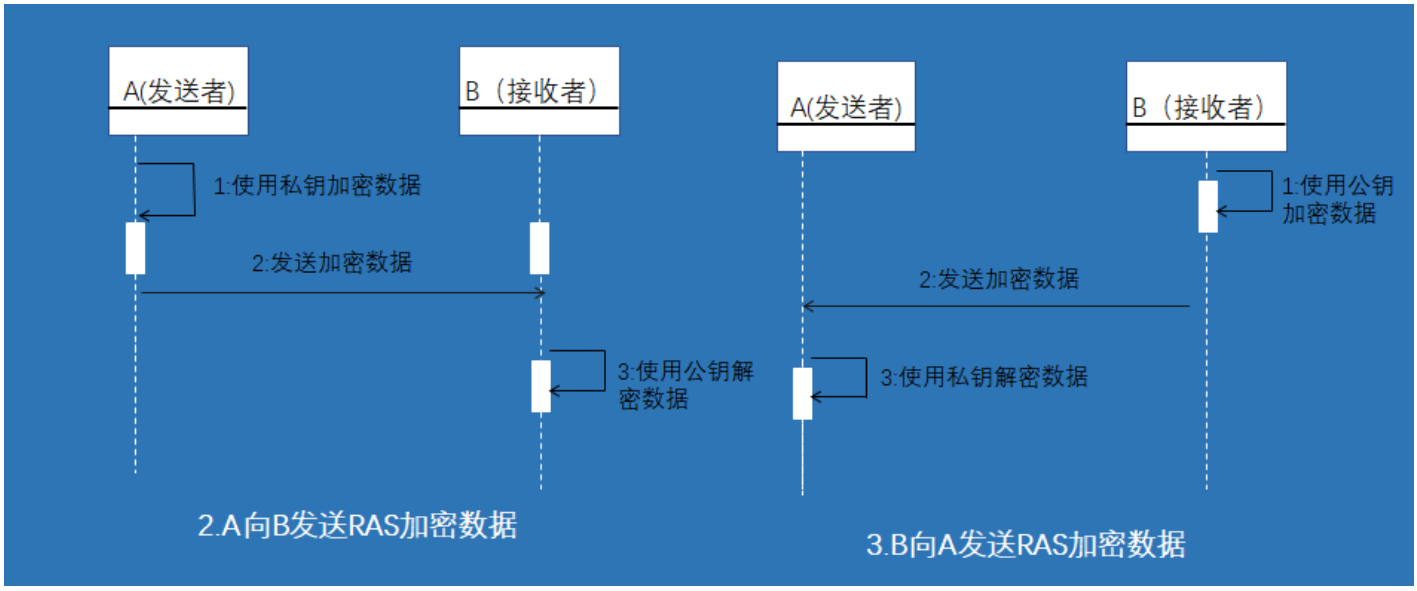
代码示例如下：

```
 */
public  static  byte[] decrypt(byte[] data,byte[] key) th
    //还原key
    Key k= DESCoder.tokey(key);
    //实例化
    Cipher cipher =Cipher.getInstance(CIPHER_ALGORITHM);
    //初始化，设置解密模式
    cipher.init(Cipher.DECRYPT_MODE,k);
    return  cipher.doFinal(data);
}
```

三、非对称加密算法

1、消息传递模型步骤如下：





2、java自带的Cipher实现

实现步骤如下：

（1） 公钥、私钥，密钥对构建，代码示例如下：

```java
//算法是DES
public static final String ALGORITHM = "RSA";
public static final String PUBLIC_KEY = "PUSHIAI";
public static final String PRIVATEC_KEY = "1qaz2wsx";
//密钥长度 512~63336
public static final int KEY_SIZE = 1024;
/**
 * 密钥对初始化
 * @return
 * @throws NoSuchAlgorithmException
 */
private  static  Map<String,Object> initKey() throws NoSuchAlgorithmException {
    //实例化密钥对生成器
    KeyPairGenerator keyPairGenerator = KeyPairGenerator.getInstance(ALGORITHM);
    //初始化密钥生成器
    keyPairGenerator.initialize(KEY_SIZE);
    //生成密钥对
    KeyPair keyPair = keyPairGenerator.generateKeyPair();
    //公钥
    RSAPublicKey rsaPublicKey = (RSAPublicKey) keyPair.getPublic();
    //私钥
    RSAPrivateKey rsaPrivateKey = (RSAPrivateKey) keyPair.getPrivate();
    Map<String,Object> keyMap =new HashMap<>();
    keyMap.put(PUBLIC_KEY,rsaPublicKey);
    keyMap.put(PRIVATEC_KEY,rsaPrivateKey);
    return  keyMap;
}
```

（2）私钥加密，加密步骤如下：

1) 获取私钥

2) 生成私钥

3) 实例化Cipher

4) 初始化

5) 设置加密模式

6) 执行操作

代码示例如下：

```
*/
private  static  byte[] encryptByPrivateKey(byte[] date,byte[] key ) throws NoSuchAlg
    //取得私钥
    PKCS8EncodedKeySpec pkcs8EncodedKeySpec= new PKCS8EncodedKeySpec(key);
    KeyFactory keyFactory =KeyFactory.getInstance(ALGORITHM);
    //生成私钥
    PrivateKey privateKey = keyFactory.generatePrivate(pkcs8EncodedKeySpec);
    //对数据解密
    Cipher cipher =Cipher.getInstance(keyFactory.getAlgorithm());
    cipher.init(Cipher.ENCRYPT_MODE,privateKey);
    return  cipher.doFinal(date);
}
```

（3）

私钥解密，解密步骤如下：

1) 获取私钥

2) 生成私钥

3) 实例化Cipher

4) 初始化

5) 设置解密模式

6) 执行操作

代码示例如下：

```
private  static  byte[] decryptByPrivateKey(byte[] date,byte[] key ) throws NoS
    //取得私钥
    PKCS8EncodedKeySpec pkcs8EncodedKeySpec= new PKCS8EncodedKeySpec(key);
    KeyFactory keyFactory =KeyFactory.getInstance(ALGORITHM);
    //生成私钥
    PrivateKey privateKey = keyFactory.generatePrivate(pkcs8EncodedKeySpec);
    //对数据解密
    Cipher cipher =Cipher.getInstance(keyFactory.getAlgorithm());
    cipher.init(Cipher.DECRYPT_MODE,privateKey);
    return  cipher.doFinal(date);
}
```

（4）公钥加密，加密步骤如下：

1) 获取公钥

2) 生成公钥

3) 实例化Cipher

4) 初始化

5) 设置加密模式

6) 执行操作

代码示例如下：

```
private  static  byte[] encryptByPublicKey(byte[] date,byte[] key ) throws No
    //取得公钥
    X509EncodedKeySpec x509EncodedKeySpec= new X509EncodedKeySpec(key);
    KeyFactory keyFactory =KeyFactory.getInstance(ALGORITHM);
    //生成公钥
    PublicKey publicKey =keyFactory.generatePublic(x509EncodedKeySpec);
    //对数据解密
    Cipher cipher =Cipher.getInstance(keyFactory.getAlgorithm());
    cipher.init(Cipher.ENCRYPT_MODE,publicKey);
    return  cipher.doFinal(date);
}
```

（5）公钥解密，解密步骤如下：

1) 获取公钥

2) 生成公钥

3) 实例化Cipher

4) 初始化

5) 设置解密模式

6) 执行操作

代码示例如下：

```
private  static  byte[] decryptByPublicKey(byte[] date,byte[] key ) throws NoSuc
 //取得公钥
    X509EncodedKeySpec x509EncodedKeySpec= new X509EncodedKeySpec(key);
    KeyFactory keyFactory =KeyFactory.getInstance(ALGORITHM);
    //生成公钥
    PublicKey publicKey =keyFactory.generatePublic(x509EncodedKeySpec);
    //对数据解密
    Cipher cipher =Cipher.getInstance(keyFactory.getAlgorithm());
    cipher.init(Cipher.DECRYPT_MODE,publicKey);
    return  cipher.doFinal(date);
}
```

总结：加密重要吗？当然，它们是一个额外的保护层。

- THE END -

文章内容仅代表作者个人观点

作者：陈鸿姣

编辑：詹思璇

想了解更多关于人工智能的资讯

别忘了关注**普适极客**