链滴

密码：参考 TLS 握手协议实现移动支付密码的一次一密加解密方案

作者：believelelf

原文链接：https://ld246.com/article/1573833378867

来源网站：链滴

本文阐述了一种参考TLS握手协议实现的移动端APP支付密码一次一密加解密方案。具体内容为五个分组成，即TLS协议简单介绍及握手协议的流程说明，支付密码一次一密加解密流程说明，涉及的加算法，实施过程，示例代码及测试结果等。

什么TLS协议?

TLS协议的主要目标是在两个通信应用程序之间提供隐私和数据完整性。该协议由两层组成:TLS记录议和TLS握手协议。

● 记录协议(Record protocol) : 建立在可靠的传输协议（如TCP）之上，为高层协议提供数据封装、缩、加密等基本功能的支持。

● 握手协议(Handshake protocol): 建立在SSL记录协议之上，用于在实际的数据传输开始前，通信方进行身份认证、协商加密算法、交换加密密钥等。

了解更多请参考RFC5246文档The Transport Layer Security (TLS) ProtocolVersion 1.2

1. Introduction

The primary goal of the TLS protocol is to provide privacy and data integrity between two communicating applications. The protocol is composed of two layers: the TLS Record Protocol and the TLS Handshake Protocol. At the lowest level, layered on top of some reliable transport protocol (e.g., TCP [TCP]), is the TLS Record Protocol. The TLS Record Protocol provides connection security that has two basic properties:

- The connection is private. Symmetric cryptography is used for data encryption (e.g., AES [AES], RC4 [SCH], etc.). The keys for this symmetric encryption are generated uniquely for each connection and are based on a secret negotiated by another protocol (such as the TLS Handshake Protocol). The Record

Protocol can also be used without encryption.

- The connection is reliable. Message transport includes a message integrity check using a keyed MAC. Secure hash functions (e.g., SHA-1, etc.) are used for MAC computations. The Record Protocol can operate without a MAC, but is generally only used in this mode while another protocol is using the Record Protocol as a transport for negotiating security parameters.

The TLS Record Protocol is used for encapsulation of various higher-level protocols. One such encapsulated protocol, the TLS Handshake Protocol, allows the server and client to authenticate each other and to negotiate an encryption algorithm and cryptographic keys before the application protocol transmits or receives its first byte of data. The TLS Handshake Protocol provides connection security that has three basic properties:

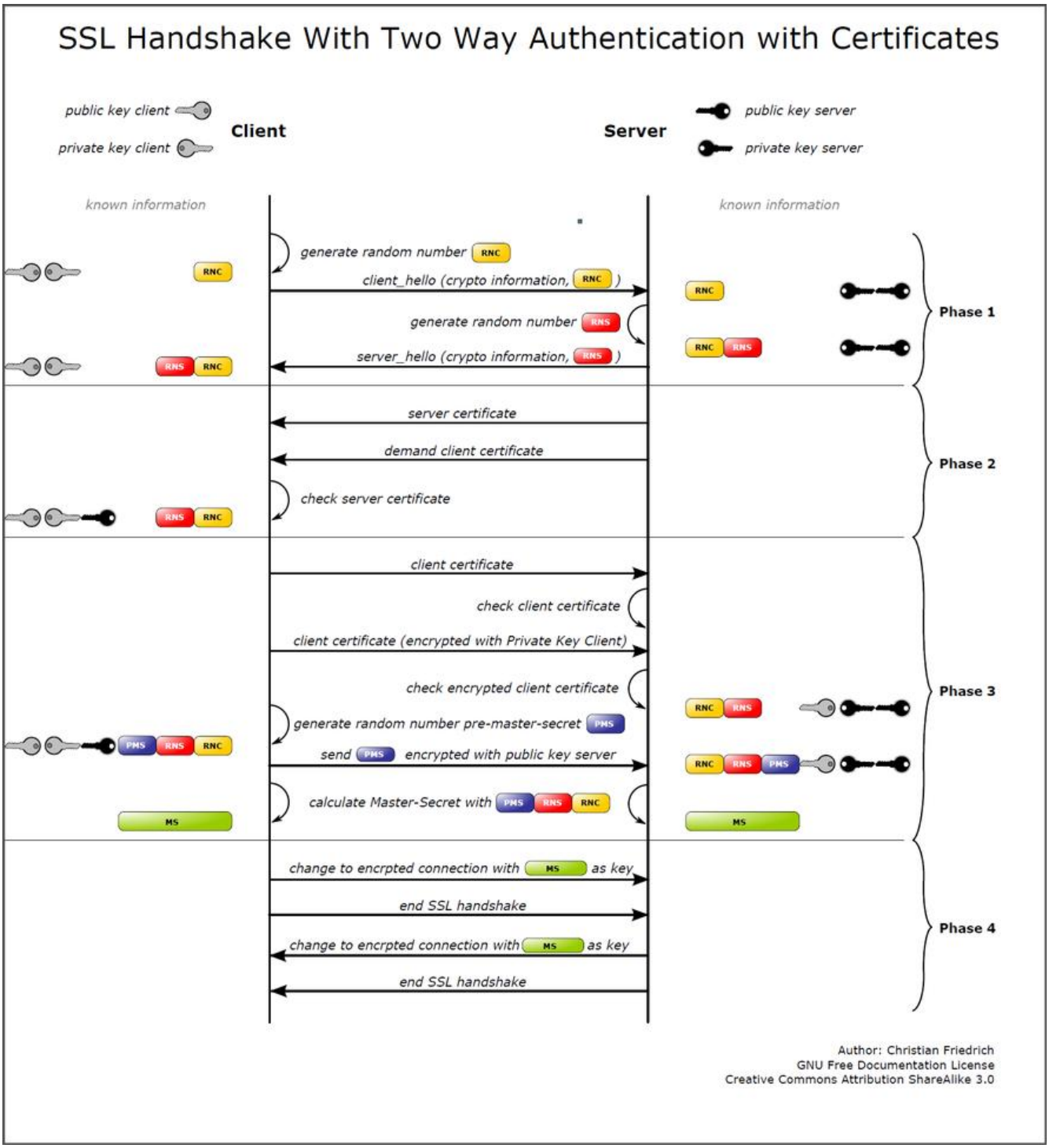
- The peer's identity can be authenticated using asymmetric, or public key, cryptography (e.g., RSA [RSA], DSA [DSS], etc.). This authentication can be made optional, but is generally required for at least one of the peers.

- The negotiation of a shared secret is secure: the negotiated secret is unavailable to eavesdroppers, and for any authenticated connection the secret cannot be obtained, even by an attacker who can place himself in the middle of the connection.

- The negotiation is reliable: no attacker can modify the negotiation communication without being detected by the parties to the communication.

One advantage of TLS is that it is application protocol independent. Higher-level protocols can layer on top of the TLS protocol transparently. The TLS standard, however, does not specify how protocols add security with TLS; the decisions on how to initiate TLS handshaking and how to interpret the authentication certificates exchanged are left to the judgment of the designers and implementors of protocols that run on top of TLS.

TLS协议中的握手协议流程是怎样的?



以上图片来源为：SSL Handshake With Two Way Authentication with Certificates

TLS握手协议主要包括4个环节：协议算法、验证证书、产生密钥、加密交互。以下主要分析在4个环中服务器端与客户端构建加密交互的相关流程。

了解更多请参考文档Crypto in practice: What happens when you connect to a secure web server，其通过抓包实例对整个过程进行详细的描述。

1. 协商算法

- **generate random number**: 客户端产生随机数RNC(Random Number of Client),这个随机数将

后续构建密钥做准备。

● **client_hello**: 客户端将自身支持的SSL信息（版本和种类）、算法信息和随机数RNC发送到服务器。

● **generate random number**: 服务器端收到客户端请求后，产生相应的随机数RNS(Random Number of Server),这个随机数为后续构建密钥做准备。

● **server_hello**: 服务器端将自身支持的SSL信息（版本和种类）、算法信息、随机数RNS和其他信息应到客户端。其他信息包括服务器证书，甚至包含获取客户端证书的请求。

经过第一阶段后，服务器端和客户端已经确认两方交互时所使用的加密算法。

2. 验证证书

服务器端下发服务器证书给客户端后，由客户端验证证书，服务器端身份得以认证后，客户端和服务端可以进行以服务器端单向认证为基础的加密交互。

● **server certificate**： 服务器回复客户端响应时带有服务器证书

● **check server certificate**: 客户端将该证书发送到认证机构，由认证机构验证该证书，认证机构回客户端验证结果，如果验证失败将同时得到警告信息。

如果服务器端对于客户端身份有要求，下发服务器证书的同时要求客户端提供证书，将构建基于客户和服务器端两方的双赂认证基础，但通常客户端证书谁不一定是必需的。服务器端验证客户端证书步如下：

● **demand client certificate**: 服务器端请求客户端证书

● **client certificate**: 客户端发送客户证书

● **check client certificate**: 服务器端将该证书发送到认证机构，由认证机构验证该证书，认证机构应客户端验证结果，如果验证失败将同时得到警告信息。

3. 产生密钥

服务器端和客户端最初需要建立主密钥为构建会话密钥做准备。

● **generate random number pre-master-secret**: 客户端产生随机数，作为预备主密钥(Pre-Master Secret,PMS)。

● **send encrypted with PMS public key server**: 客户端使用服务器证书的公钥对随机数PMS加，并将PMS加密信息发送到服务器端。

● **decrypt PMS encrypted information**: 服务器端使用私钥对信息解密得到PMS信息。

● **calculate Master-Secret with PMS RNS RNC**: 客户端和服务端分别异步（不存在次序关系）用随机数RNC、RNS和PMS构建主密钥(Master Secret, MS)

完成主密钥构建操作后，服务器和客户端将建议会话密钥，即将完成握手协议。

● **change to encrpted connection with MS as key**: 客户端使用主密钥构建会话密钥，并通知服器端未来信息将使用会话密钥(对称加密算法中的秘密密钥client_write_key)加密

● **end SSL handshake**: 发送会话密钥加密的信息，终止握手

● **change to encrpted connection with MS as key**: 服务器端使用主密钥构建会话密钥，并通知户器端未来信息将使用会话密钥(对称加密算法中的秘密密钥server_write_key)加密

● **end SSL handshake**: 发送会话密钥加密的信息，终止握手

会话密钥生成过程可见rfc5246#section-6.3 Key Calculation。

The master secret is expanded into a sequence of secure bytes, which is then split to a client write MAC key, a server write MAC key, a client write encryption key, and a server write encryption key. Each of these is generated from the byte sequence in that order. Unused values are empty. Some AEAD ciphers may additionally require a client write IV and a server write IV (see Section 6.2.3.3).

When keys and MAC keys are generated, the master secret is used as an entropy source.

To generate the key material, compute

```
key_block = PRF(SecurityParameters.master_secret,
                "key expansion",
                SecurityParameters.server_random +
                SecurityParameters.client_random);
```

until enough output has been generated. Then, the key_block is partitioned as follows:

```
client_write_MAC_key[SecurityParameters.mac_key_length]
server_write_MAC_key[SecurityParameters.mac_key_length]
client_write_key[SecurityParameters.enc_key_length]
server_write_key[SecurityParameters.enc_key_length]
client_write_IV[SecurityParameters.fixed_iv_length]
server_write_IV[SecurityParameters.fixed_iv_length]
```

Currently, the client_write_IV and server_write_IV are only generated for implicit nonce techniques as described in Section 3.2.1 of [AEAD].

Implementation note: The currently defined cipher suite which requires the most material is AES_256_CBC_SHA256. It requires 2 x 32 byte keys and 2 x 32 byte MAC keys, for a total 128 bytes of key material.

其代码实现可以参见文章

● TLS, Pre-Master Secrets and Master Secrets

● Crypto in practice: What happens when you connect to a secure web server

The pseudorandom function in TLS v 1.2.

```
#We define the PRF as specified in the standard, except we add a fourth parameter
#for the number of blocks to output.
def prf(secret,label,seed,numblocks):
 seed=label+seed
```

```
 output = ''
 a=hmac.new(secret,msg=seed,digestmod=hashlib.sha256).digest()
 for j in range(numblocks):
 output += hmac.new(secret,msg=a+seed,digestmod=hashlib.sha256).digest()
 a=hmac.new(secret,msg=a,digestmod=hashlib.sha256).digest()
 return output
```

Computation of the master secret from the premaster secret.

```
#Compute the master secret from the premaster secret.
#premaster_secret, client_random and server_random have string type, that
#is they are sequences of bytes represented as strings,
#so the addition in this code is concatenation of strings.
#We want 48-bit output, so we need to call prf to produce two
#32-bit blocks
def master_secret(pms,client_random,server_random):
 out=prf(pms,"master secret",client_random+server_random,2)
 return out[:48]
```

Computation of the keyblock from the master secret

```
#generate the key block. We will need 20+20+32+32=104 bytes, so we need
#to generate 4 blocks and partition. We are just doing the case AES256CBCSHA
def keyblock(ms,client_random,server_random):
 u=prf(ms,"key expansion",server_random+client_random,4)
 return (u[:20],u[20:40],u[40:72],u[72:104])
```

在Golang官方库源代码prf.go亦有对应实现。

以下测试数据来源于Golang对其TLS1.0协议实现的测试，见prf_test.go

pre-master-secret
03023f7527316bc12cbcd69e4b9e8275d62c028f27e65c745cfcddc7ce01bd3570a111378b6384
127f1c36e5f9e4890
96/2 = 48

client random
4ae66364b5ea56b20ce4e25555aed2d7e67f42788dd03f3fee4adae0459ab106
64/2 = 32

server random
4ae66363ab815cbf6a248b87d6b556184e945e9b97fbdf247858b0bdafacfa1c
64/2 = 32

master-secret
3d851bab6e5556e959a16bc36d66cfae32f672bfa9ecdef6096cbb1b23472df1da63dbbd982760
413221d149ed08ceb
96/2 = 48

clientMAC
3c7647c93c1379a31a609542aa44e7f117a70085
40/2 = 20

serverMAC
0d73102994be74a575a3ead8532590ca32a526d4

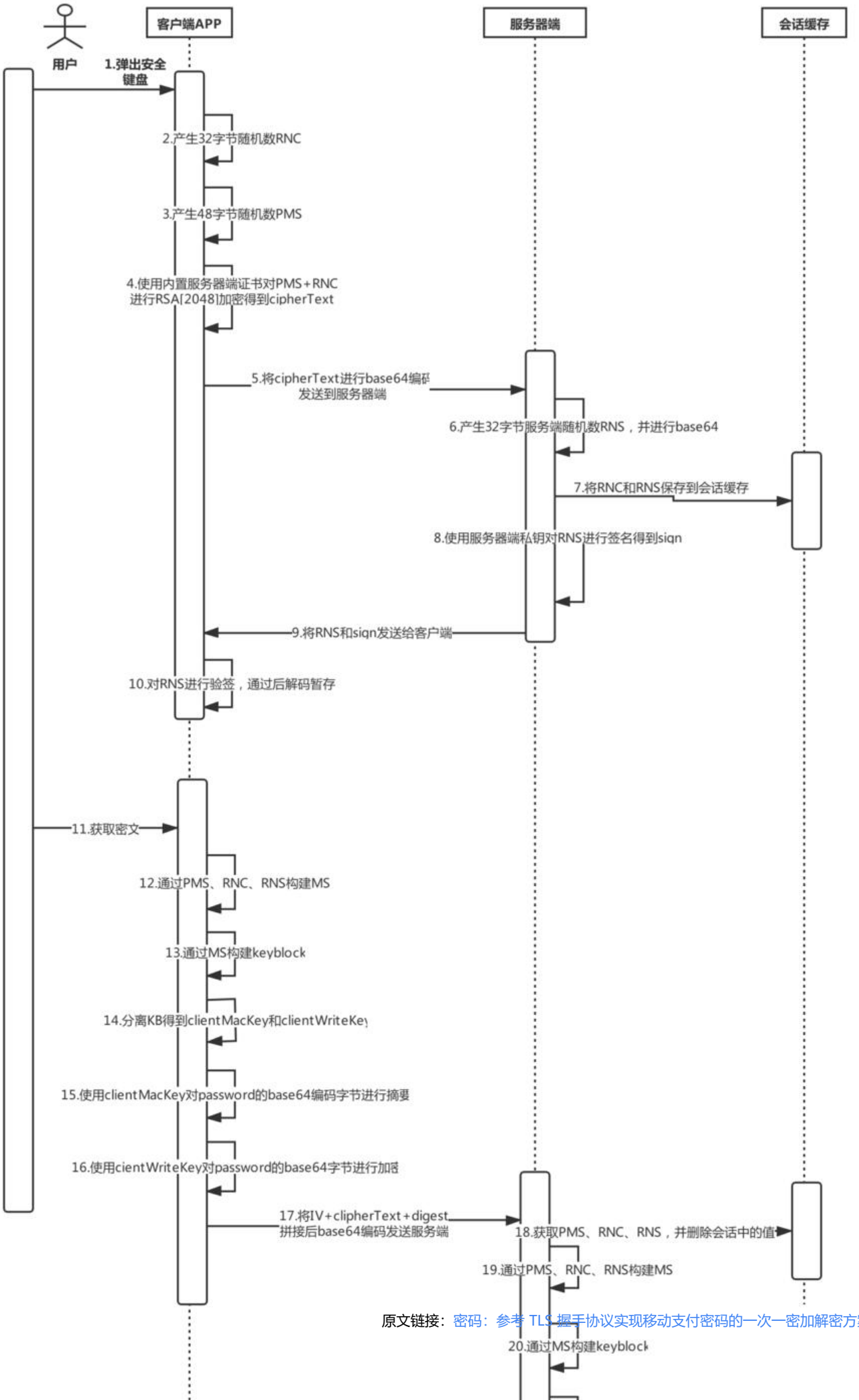
40/2 = 20

clientKey
ac7581b0b6c10d85bbd905ffbf36c65e
32/2 = 16

serverKey
ff07edde49682b45466bd2e39464b306
32/2 = 16

怎么设计一次一密加解密流程?

根据TLS握手协议修改的一次一密加解密流程如下图，将握手协议的流程裁剪为三步：交换随机数、生密钥和加密交互。



原文链接：密码：参考 TLS 握手协议实现移动支付密码的一次一密加解密方案

交换随机数

1. 客户端产生随机数RNC和PMS
2. 客户端使用服务器端证书公钥以RSA[2048]算法对RNC+PMS进行加密
3. 客户端将clipherText进行base64编码后发送到服务端
4. 服务端产生随机数RNS并进行base64
5. 服务端将随机数RNS和clipherText保存到会话缓存
6. 服务端使用私钥对RNS进行签名得到sign
7. 服务端将RNS和sign发送给客户端
8. 客户端对收到RNS进行验签，成功后暂存RNC,RNS、PMS.

产生密钥

1. 客户端使用RNC,RNS、PMS生成Master Secret
2. 客户端使用Master Secret产生keyblock
3. 客户端分离keyblock得到clientMacKey和clientWriteKey

加密交互

1. 客户端使用clientMacKey对password进行摘要
2. 客户端使用clientWriteKey对password进行AES-256-GCM加密
3. 客户将IV+clipherText+digest拼接进行base64后上送服务器端
4. 服务器端使用RNC,RNS、PMS生成Master Secret
5. 服务器端使用Master Secret产生keyblock
6. 服务器端分离keyblock得到clientMacKey和clientWriteKey
7. 服务器端使用clientWriteKey对password进行AES-256-GCM解密
8. 服务器端使用clientMacKey对password进行摘要，再进行摘要验证

10.服务器端使用pbkdf2产生密钥散列值与数据库散列进行比较

涉及哪些加密算法?

- 对称算法：AES-GCM-256
- 非对称加密算法: RSA[3072]
- 数字签名算法：SHA256withRSA
- 摘要算法：HmacSHA256、PBKDF2-SHA-1
- 单表置换算法：自定义Base64

了解更多相关算法选择因素，请参见Cryptography in Mobile Apps

The following algorithms are recommended:
Confidentiality algorithms: AES-GCM-256 or ChaCha20-Poly1305

Integrity algorithms: SHA-256, SHA-384, SHA-512, Blake2, the SHA-3 family
Digital signature algorithms: RSA (3072 bits and higher), ECDSA with NIST P-384
Key establishment algorithms: RSA (3072 bits and higher), DH (3072 bits or higher), ECDH wit NIST P-384

具体怎么做?

根据上述三个阶段的流程描述，在本章节将阐述如何实现此方案。

无政策限制权限文件的获取和放置

1. 下载无政策限制权限文件。链接地址为：

Java Cryptography Extension (JCE) Unlimited Strength Jurisdiction Policy Files 8 Download

2. 安装步骤见其README.TXT。

```
---------------------------------------------------------------------
Installation
---------------------------------------------------------------------

Notes:

  o Unix (Solaris/Linux/Mac OS X) and Windows use different pathname
    separators, so please use the appropriate one ("\", "/") for your
    environment.

  o <java-home> (below) refers to the directory where the JRE was
    installed. It is determined based on whether you are running JCE
    on a JRE or a JRE contained within the Java Development Kit, or
    JDK(TM). The JDK contains the JRE, but at a different level in the
    file hierarchy. For example, if the JDK is installed in
    /home/user1/jdk1.8.0 on Unix or in C:\jdk1.8.0 on Windows, then
    <java-home> is:

        /home/user1/jdk1.8.0/jre          [Unix]
        C:\jdk1.8.0\jre                   [Windows]

    If on the other hand the JRE is installed in /home/user1/jre1.8.0
    on Unix or in C:\jre1.8.0 on Windows, and the JDK is not
    installed, then <java-home> is:

        /home/user1/jre1.8.0              [Unix]
        C:\jre1.8.0                       [Windows]

  o On Windows, for each JDK installation, there may be additional
    JREs installed under the "Program Files" directory. Please make
    sure that you install the unlimited strength policy JAR files
    for all JREs that you plan to use.


Here are the installation instructions:

1)  Download the unlimited strength JCE policy files.
```

2) Uncompress and extract the downloaded file.

This will create a subdirectory called jce.
This directory contains the following files:

```
README.txt              This file
local_policy.jar        Unlimited strength local policy file
US_export_policy.jar    Unlimited strength US export policy file
```

3) Install the unlimited strength policy JAR files.

In case you later decide to revert to the original "strong" but limited policy versions, first make a copy of the original JCE policy files (US_export_policy.jar and local_policy.jar). Then replace the strong policy files with the unlimited strength versions extracted in the previous step.

The standard place for JCE jurisdiction policy JAR files is:

```
<java-home>/lib/security        [Unix]
<java-home>\lib\security        [Windows]
```

生成RSA密钥库文件

使用java自带的keytool工具生成RSA[3072]密钥库及导出公钥证书。

keytool

Manages a keystore (database) of cryptographic keys, X.509 certificate chains, and trusted cer ificates.

- **Description**

The keytool command is a key and certificate management utility. It enables users to adminis er their own public/private key pairs and associated certificates for use in self-authentication where the user authenticates himself or herself to other users and services) or data integrity a d authentication services, using digital signatures. The keytool command also enables users to cache the public keys (in the form of certificates) of their communicating peers.

A certificate is a digitally signed statement from one entity (person, company, and so on.), tha says that the public key (and some other information) of some other entity has a particular va ue. (See Certificate.) When data is digitally signed, the signature can be verified to check the ata integrity and authenticity. Integrity means that the data has not been modified or tamper d with, and authenticity means the data comes from whoever claims to have created and sign d it.

The keytool command also enables users to administer secret keys and passphrases used in s mmetric encryption and decryption (DES).

The keytool command stores the keys and certificates in a keystore. See KeyStore aliases.

了解更多请参见keytool command

生成keyStore

```
-genkeypair
   {-alias alias} {-keyalg keyalg} {-keysize keysize} {-sigalg sigalg}
   [-dname dname] [-keypass keypass] {-startdate value} {-ext ext}*
   {-validity valDays} {-storetype storetype} {-keystore keystore}
   [-storepass storepass]
   {-providerClass provider_class_name {-providerArg provider_arg}}
   {-v} {-protected} {-Jjavaoption}
```

Generates a key pair (a public key and associated private key). Wraps the public key into an X.
09 v3 self-signed certificate, which is stored as a single-element certificate chain. This certifica
e chain and the private key are stored in a new keystore entry identified by alias.

The keyalg value specifies the algorithm to be used to generate the key pair, and the keysize
alue specifies the size of each key to be generated. The sigalg value specifies the algorithm th
t should be used to sign the self-signed certificate. This algorithm must be compatible with t
e keyalg value.

The dname value specifies the X.500 Distinguished Name to be associated with the value of al
as, and is used as the issuer and subject fields in the self-signed certificate. If no distinguished
name is provided at the command line, then the user is prompted for one.

The value of keypass is a password used to protect the private key of the generated key pair. I
no password is provided, then the user is prompted for it. If you press the Return key at the p
ompt, then the key password is set to the same password as the keystore password. The keyp
ss value must be at least 6 characters.

The value of startdate specifies the issue time of the certificate, also known as the "Not Before
value of the X.509 certificate's Validity field.

The option value can be set in one of these two forms:

([+-]nnn[ymdHMS])+

[yyyy/mm/dd] [HH:MM:SS]

With the first form, the issue time is shifted by the specified value from the current time. The
alue is a concatenation of a sequence of subvalues. Inside each subvalue, the plus sign (+) me
ns shift forward, and the minus sign (-) means shift backward. The time to be shifted is nnn un
ts of years, months, days, hours, minutes, or seconds (denoted by a single character of y, m, d,
H, M, or S respectively). The exact value of the issue time is calculated using the java.util.Greg
rianCalendar.add(int field, int amount) method on each subvalue, from left to right. For examp
e, by specifying, the issue time will be:

```
Calendar c = new GregorianCalendar();
c.add(Calendar.YEAR, -1);
c.add(Calendar.MONTH, 1);
c.add(Calendar.DATE, -1);
return c.getTime()
```

With the second form, the user sets the exact issue time in two parts, year/month/day and ho
r:minute:second (using the local time zone). The user can provide only one part, which means
he other part is the same as the current date (or time). The user must provide the exact numb

r of digits as shown in the format definition (padding with 0 when shorter). When both the da e and time are provided, there is one (and only one) space character between the two parts. he hour should always be provided in 24 hour format.

When the option is not provided, the start date is the current time. The option can be provid d at most once.

The value of valDays specifies the number of days (starting at the date specified by -startdate, or the current date when -startdate is not specified) for which the certificate should be consid red valid.

This command was named -genkey in earlier releases. The old name is still supported in this r lease. The new name, -genkeypair, is preferred going forward.

- **生成命令**

```
keytool -genkeypair -alias www.weiquding.com -keyalg RSA -keysize 3072 -keypass weiqudin
 -sigalg SHA256withRSA -dname "cn=www.weiquding.com,ou=IT,o=weiquding,l=shanghai,st
shanghai,c=CN" -validity 3650 -keystore safeKeyboard.keystore -storetype pkcs12 -storepass
weiquding
```

- **打印密钥库条目内容**

```
-list
    {-alias alias} {-storetype storetype} {-keystore keystore} [-storepass storepass]
    {-providerName provider_name}
    {-providerClass provider_class_name {-providerArg provider_arg}}
    {-v | -rfc} {-protected} {-Jjavaoption}
```

Prints to stdout the contents of the keystore entry identified by alias. If no alias is specified, t en the contents of the entire keystore are printed.

This command by default prints the SHA1 fingerprint of a certificate. If the -v option is specifi d, then the certificate is printed in human-readable format, with additional information such a the owner, issuer, serial number, and any extensions. If the -rfc option is specified, then the c rtificate contents are printed using the printable encoding format, as defined by the Internet FC 1421 Certificate Encoding Standard.

You cannot specify both -v and -rfc.

- **以-v选项进行打印**

```
keytool -list -v -storepass weiquding -keystore safeKeyboard.keystore
```

- **以-v选项打印结果**

```
E:\Java\idea_workspaces\SafeKeyboard>keytool -list -v -storepass weiquding -keystore safeK
yboard.keystore
密钥库类型: PKCS12
密钥库提供方: SUN

您的密钥库包含 1 个条目

别名: www.weiquding.com
```

```
创建日期: 2019年11月14日
条目类型: PrivateKeyEntry
证书链长度: 1
证书[1]:
所有者: CN=www.weiquding.com, OU=IT, O=weiquding, L=shanghai, ST=shanghai, C=CN
发布者: CN=www.weiquding.com, OU=IT, O=weiquding, L=shanghai, ST=shanghai, C=CN
序列号: 226cfc5b
生效时间: Thu Nov 14 22:00:02 CST 2019, 失效时间: Sun Nov 11 22:00:02 CST 2029
证书指纹:
         SHA1: B0:19:24:70:C7:B3:E8:58:E8:0B:54:A9:F3:59:BC:0B:57:CF:F7:3E
         SHA256: 93:5A:8E:53:47:AA:55:3A:D8:85:31:CB:1E:AE:ED:FA:21:BC:59:8A:33:2B:73:FC:CC:C
:D0:2E:96:72:9B:3D
签名算法名称: SHA256withRSA
主体公共密钥算法: 3072 位 RSA 密钥
版本: 3

扩展:

#1: ObjectId: 2.5.29.14 Criticality=false
SubjectKeyIdentifier [
KeyIdentifier [
0000: B7 72 E9 0E C3 35 98 39   7F B0 DC C2 6D DD E1 78  .r...5.9....m..x
0010: 58 68 BF 05                                        Xh..
]
]



*******************************************
*******************************************
```

● 以-rfc选项进行打印

```
keytool -list -rfc -storepass weiquding -keystore safeKeyboard.keystore
```

● 以-rfc选项打印结果

```
E:\Java\idea_workspaces\SafeKeyboard>keytool -list -rfc -storepass weiquding -keystore safe
eyboard.keystore
密钥库类型: PKCS12
密钥库提供方: SUN

您的密钥库包含 1 个条目

别名: www.weiquding.com
创建日期: 2019年11月14日
条目类型: PrivateKeyEntry
证书链长度: 1
证书[1]:
-----BEGIN CERTIFICATE-----
MIIEfzCCAuegAwIBAgIEImz8WzANBgkqhkiG9w0BAQsFADBwMQswCQYDVQQGEwJD
TjERMA8GA1UECBMIc2hhbmdoYWkxETAPBgNVBAcTCHNoYW5naGFpMRIwEAYDVQQK
Ewl3ZWlxdWRpbmcxCzAJBgNVBAsTAklUMRowGAYDVQQDExF3d3cud2VpcXVkaW5n
```

LmNvbTAeFw0xOTExMTQxNDAwMDJaFw0yOTExMTExNDAwMDJaMHAxCzAJBgNVBAYT
AkNOMREwDwYDVQQIEwhzaGFuZ2hhaTERMA8GA1UEBxMIc2hhbmdoYWkxEjAQBgNV
BAoTCXdlaXF1ZGluZzELMAkGA1UECxMCSVQxGjAYBgNVBAMTEXd3dy53ZWlxdWRp
bmcuY29tMIIBojANBgkqhkiG9w0BAQEFAAOCAY8AMIIBigKCAYEAsmpeRVum/g06
HxEZ3TAKxloC0X/ooYQ+X1+OadolQsIrbkiVAGg4xwpNyaGpZvmEEtN5RXpeVo55
B//GQREUq1sYJ1jZPHKhLxfhAelHGAN3+bL2avT5HIzUH/XnFXWarnP4bnXFzKwe
UMWTzVrD/BUPJuaxacG5leUWBBcZjrFf4tKWVN45hm2/GJzZ9QeHCebdwzSAveSu
rtdMv/3lIsPUahRmF1ri7RLxvcGF4unV21Lt3ZSBuV/U22WOXy/E/dalog5XedFz
RowpYh5itke2gTs01aVX1OGSQCaxLLZHUDiZ9kqCJ+aGTHveYtmCqA9CuvbuelFx
u+ccF+qgAS1KXjPNQGYsiwON/VCz7oM4gIJmK1AYa+sZB0/S6Hj9/UjG79nwJkTO
M1o3mE5T6XLr/5CiaDvOgcEE5cvVe3PbVYgWKEVfCHL24xxWLWDqeQ54dAihdl/R
6BMa3o8jzKHcAXsU8ciw4CByll5jjbyx32yjFgwdligXikTPnC3XAgMBAAGjITAf
MB0GA1UdDgQWBBS3cukOwzWYOX+w3MJt3eF4WGi/BTANBgkqhkiG9w0BAQsFAAOC
AYEAeAlgBPKMq5ndex1oR19mfAji+D5IUSKQISIyoDYFP0NY94+9pW/HNBiFuKnU
EGP2VEOgmdidmE13pzU+Z2/Ya6kql0rX7ha8UJpyb8qy18mYdzbwzBlEd0JLDDX5
pLrJD9m91nOXI9ExTYs5ZY8VT4kmZKAwoStpMq+NuouAZy7ZIPlm0kMcH5dyd36A
7DSlWa3wOXpU7YhfJMF5YzZBK3eP0FZt65tSbhDBvodvwYmndTgCA4LxgGuF28iD
qR/kEfImvJRLZcebv4abuHtoa7Ss6Jpd/mkPN615cM7/xkkwxQ37wie5Ro9xJNEC
eLuZouObCplwlHC7PBCky9YodpNR6P8ZXybYKPxCefJXwUzTnnZ4HCqgNE6TkFzt
HaDqsJpUP5Cz8Sk5OBjiwcPAqK5jTk171YnPYUvahQZ+daux9Rj5WkP7IMVlNecw
fZh+w2f/yMLBmYFoDG/1APUvSL59nyR7uPi3CHNhASjjg8sJnLRoT6F40QigthtQ
hcxp
-----END CERTIFICATE-----

****
****

导出包含RSA公钥的数字证书

```
-exportcert
   {-alias alias} {-file cert_file} {-storetype storetype} {-keystore keystore}
   [-storepass storepass] {-providerName provider_name}
   {-providerClass provider_class_name {-providerArg provider_arg}}
   {-rfc} {-v} {-protected} {-Jjavaoption}
```

Reads from the keystore the certificate associated with alias and stores it in the cert_file file. hen no file is specified, the certificate is output to stdout.

The certificate is by default output in binary encoding. If the -rfc option is specified, then the utput in the printable encoding format defined by the Internet RFC 1421 Certificate Encoding Standard.

If alias refers to a trusted certificate, then that certificate is output. Otherwise, alias refers to a ey entry with an associated certificate chain. In that case, the first certificate in the chain is ret rned. This certificate authenticates the public key of the entity addressed by alias.

This command was named -export in earlier releases. The old name is still supported in this re ease. The new name, -exportcert, is preferred going forward.

- **导出命令**

```
keytool -exportcert -alias www.weiquding.com -storepass weiquding -file safeKeyboard.cer -k
ystore safeKeyboard.keystore
```

- **打印公钥内容**

-printcert
{-file cert_file | -sslserver host[:port]} {-jarfile JAR_file {-rfc} {-v}
{-Jjavaoption}

Reads the certificate from the file cert_file, the SSL server located at host:port, or the signed J R file JAR_file (with the -jarfile option and prints its contents in a human-readable format. Wh n no port is specified, the standard HTTPS port 443 is assumed. Note that -sslserver and -file ptions cannot be provided at the same time. Otherwise, an error is reported. If neither option s specified, then the certificate is read from stdin.

When-rfc is specified, the keytool command prints the certificate in PEM mode as defined by he Internet RFC 1421 Certificate Encoding standard. See Internet RFC 1421 Certificate Encodi g Standard.

If the certificate is read from a file or stdin, then it might be either binary encoded or in print ble encoding format, as defined by the RFC 1421 Certificate Encoding standard.

If the SSL server is behind a firewall, then the -J-Dhttps.proxyHost=proxyhost and -J-Dhttps.p oxyPort=proxyport options can be specified on the command line for proxy tunneling. See Ja a Secure Socket Extension (JSSE) Reference Guide at

http://docs.oracle.com/javase/8/docs/technotes/guides/security/jsse/JSSERefGuide.html

Note: This option can be used independently of a keystore.

- **以-v选项进行打印**

keytool -printcert -v -file safeKeyboard.cer

- **以-v选项打印结果**

```
E:\Java\idea_workspaces\SafeKeyboard>keytool -printcert -v -file safeKeyboard.cer
所有者: CN=www.weiquding.com, OU=IT, O=weiquding, L=shanghai, ST=shanghai, C=CN
发布者: CN=www.weiquding.com, OU=IT, O=weiquding, L=shanghai, ST=shanghai, C=CN
序列号: 226cfc5b
生效时间: Thu Nov 14 22:00:02 CST 2019, 失效时间: Sun Nov 11 22:00:02 CST 2029
证书指纹:
        SHA1: B0:19:24:70:C7:B3:E8:58:E8:0B:54:A9:F3:59:BC:0B:57:CF:F7:3E
        SHA256: 93:5A:8E:53:47:AA:55:3A:D8:85:31:CB:1E:AE:ED:FA:21:BC:59:8A:33:2B:73:FC:CC:C
:D0:2E:96:72:9B:3D
签名算法名称: SHA256withRSA
主体公共密钥算法: 3072 位 RSA 密钥
版本: 3

扩展:

#1: ObjectId: 2.5.29.14 Criticality=false
SubjectKeyIdentifier [
KeyIdentifier [
0000: B7 72 E9 0E C3 35 98 39   7F B0 DC C2 6D DD E1 78  .r...5.9....m..x
0010: 58 68 BF 05                                        Xh..
]
]
```

- **以-rfc选项进行打印**

```
keytool -printcert -rfc -file safeKeyboard.cer
```

- **以-rfc选项打印输出**

```
E:\Java\idea_workspaces\SafeKeyboard>keytool -printcert -rfc -file safeKeyboard.cer
-----BEGIN CERTIFICATE-----
MIIEfzCCAuegAwIBAgIEImz8WzANBgkqhkiG9w0BAQsFADBwMQswCQYDVQQGEwJD
TjERMA8GA1UECBMIc2hhbmdoYWkxETAPBgNVBAcTCHNoYW5naGFpMRIwEAYDVQQK
Ewl3ZWlxdWRpbmcxCzAJBgNVBAsTAklUMRowGAYDVQQDExF3d3cud2VpcXVkaW5n
LmNvbTAeFw0xOTExMTQxNDAwMDJaFw0yOTExMTExNDAwMDJaMHAxCzAJBgNVBAYT
AkNOMREwDwYDVQQIEwhzaGFuZ2hhaTERMA8GA1UEBxMIc2hhbmdoYWkxEjAQBgNV
BAoTCXdlaXF1ZGluZzELMAkGA1UECxMCSVQxGjAYBgNVBAMTEXd3dy53ZWlxdWRp
bmcuY29tMIIBojANBgkqhkiG9w0BAQEFAAOCAY8AMIIBigKCAYEAsmpeRVum/g06
HxEZ3TAKxloC0X/ooYQ+X1+OadolQsIrbkiVAGg4xwpNyaGpZvmEEtN5RXpeVo55
B//GQREUq1sYJ1jZPHKhLxfhAelHGAN3+bL2avT5HIzUH/XnFXWarnP4bnXFzKwe
UMWTzVrD/BUPJuaxacG5leUWBBcZjrFf4tKWVN45hm2/GJzZ9QeHCebdwzSAveSu
rtdMv/3lIsPUahRmF1ri7RLxvcGF4unV21Lt3ZSBuV/U22WOXy/E/dalog5XedFz
RowpYh5itke2gTs01aVX1OGSQCaxLLZHUDiZ9kqCJ+aGTHveYtmCqA9CuvbuelFx
u+ccF+qgAS1KXjPNQGYsiwON/VCz7oM4gIJmK1AYa+sZB0/S6Hj9/UjG79nwJkTO
M1o3mE5T6XLr/5CiaDvOgcEE5cvVe3PbVYgWKEVfCHL24xxWLWDqeQ54dAihdl/R
6BMa3o8jzKHcAXsU8ciw4CBylI5jjbyx32yjFgwdligXikTPnC3XAgMBAAGjITAf
MB0GA1UdDgQWBBS3cukOwzWYOX+w3MJt3eF4WGi/BTANBgkqhkiG9w0BAQsFAAOC
AYEAeAlgBPKMq5ndex1oR19mfAji+D5IUSKQISIyoDYFP0NY94+9pW/HNBiFuKnU
EGP2VEOgmdidmE13pzU+Z2/Ya6kql0rX7ha8UJpyb8qy18mYdzbwzBlEd0JLDDX5
pLrJD9m91nOXI9ExTYs5ZY8VT4kmZKAwoStpMq+NuouAZy7ZIPlm0kMcH5dyd36A
7DSlWa3wOXpU7YhfJMF5YzZBK3eP0FZt65tSbhDBvodvwYmndTgCA4LxgGuF28iD
qR/kEfImvJRLZcebv4abuHtoa7Ss6Jpd/mkPN615cM7/xkkwxQ37wie5Ro9xJNEC
eLuZouObCplwlHC7PBCky9YodpNR6P8ZXybYKPxCefJXwUzTnnZ4HCqgNE6TkFzt
HaDqsJpUP5Cz8Sk5OBjiwcPAqK5jTk171YnPYUvahQZ+daux9Rj5WkP7IMVlNecw
fZh+w2f/yMLBmYFoDG/1APUvSL59nyR7uPi3CHNhASjjg8sJnLRoT6F40QigthtQ
hcxp
-----END CERTIFICATE-----
```

基于密钥库和数字证书的加密/解密和签名/验证操作

定义属性文件

```
cipher:
  safeKeyboardKeyStorePath: E:\\Java\\idea_workspaces\\SafeKeyboard\\safeKeyboard.keysto
e
  safeKeyboardCerPath: E:\\Java\\idea_workspaces\\SafeKeyboard\\safeKeyboard.cer
  alias: www.weiquding.com
  storePass: weiquding
  keyPass: weiquding
```

定义属性配置类

```
package com.weiquding.safeKeyboard.common.domain;
```

```
import lombok.Data;
import org.springframework.boot.context.properties.ConfigurationProperties;

/**
 * 密钥库证书相关属性文件
 * @author believeyourself
 */
@Data
@ConfigurationProperties(prefix = "cipher")
public class CipherPathProperties {


    /**
     * 安全键盘密钥库
     */
    private String safeKeyboardKeyStorePath;

    /**
     * 安全键盘证书
     */
    private String safeKeyboardCerPath;

    /**
     * 安全键盘密钥库密码
     */
    private String storePass;
    /**
     * 安全键盘密钥库KEY密码
     */
    private String keyPass;
    /**
     * key别名
     */
    private String alias;


}
```

定义Key装配配置类

```
package com.weiquding.safeKeyboard.common.config;

import com.weiquding.safeKeyboard.common.cache.KeyInstance;
import com.weiquding.safeKeyboard.common.domain.CipherPathProperties;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;

/**
 * 配置类
 * @author believeyourself
 */
@Configuration
public class KeystoreAutoConfig {
```

```
    @Bean
    public static CipherPathProperties cipherPathProperties(){
        return new CipherPathProperties();
    }


    @Bean
    public KeyInstance keyInstance(CipherPathProperties cipherPathProperties){
        return new KeyInstance(cipherPathProperties);
    }


}
```

定义Key缓存类

```
package com.weiquding.safeKeyboard.common.cache;

import com.weiquding.safeKeyboard.common.domain.CipherPathProperties;
import com.weiquding.safeKeyboard.common.util.KeystoreUtil;
import lombok.extern.slf4j.Slf4j;

import javax.annotation.PostConstruct;
import java.security.KeyStore;
import java.security.cert.Certificate;
import java.security.interfaces.RSAPrivateKey;
import java.security.interfaces.RSAPublicKey;

/**
 * 缓存key实例
 * @author believeyourself
 */
@Slf4j
public class KeyInstance {

    private  CipherPathProperties properties;

    public KeyInstance(CipherPathProperties properties){
        this.properties = properties;
    }

    /**
     * 密钥库
     */
    public static KeyStore KEY_STORE;
    /**
     * RSA私钥
     */
    public static RSAPrivateKey RSA_PRIVATE_KEY;
    /**
     * 服务端证书
     */
```

```
    public static Certificate SERVER_CERTIFICATE;
    /**
     * 客户端证书
     */
    public static Certificate CLIENT_CERTIFICATE;
    /**
     * 客户端公钥
     */
    public static RSAPublicKey RSA_PUBLIC_KEY;

    @PostConstruct
    public void init(){
        try{
            KEY_STORE = KeystoreUtil.loadKeyStore(properties.getSafeKeyboardKeyStorePath(), p
operties.getStorePass());
            RSA_PRIVATE_KEY = KeystoreUtil.getRSAPrivateKey(KEY_STORE, properties.getAlias(),
roperties.getKeyPass());
            SERVER_CERTIFICATE = KeystoreUtil.getCertificate(KEY_STORE, properties.getAlias());
            CLIENT_CERTIFICATE = KeystoreUtil.loadCertificate(properties.getSafeKeyboardCerPat
());
            RSA_PUBLIC_KEY = KeystoreUtil.getRSAPublicKey(CLIENT_CERTIFICATE);
        }catch (Exception e){
            log.error("load keystore error", e);
        }

    }
}
```

使用Guava Cache提供密钥缓存能力

```
package com.weiquding.safeKeyboard.common.cache;

import com.google.common.cache.Cache;
import com.google.common.cache.CacheBuilder;

import java.util.Map;
import java.util.concurrent.TimeUnit;

/**
 * 缓存
 * @author beliveyourself
 */
public class GuavaCache {

    /**
     * CLIENT CACHE
     */
    public static final Cache<String, Map<String, String>> CLIENT_CACHE = CacheBuilder.new
uilder()
                .maximumSize(1000)
                .expireAfterAccess(300,TimeUnit.SECONDS)
                .build();
```

```
    /**
     * SERVER CACHE
     */
    public static final Cache<String, Map<String, String>> SERVER_CACHE = CacheBuilder.ne
Builder()
            .maximumSize(1000)
            .expireAfterAccess(300,TimeUnit.SECONDS)
            .build();
}
```

定义密码相关异常

```
package com.weiquding.safeKeyboard.common.exception;

/**
 * 加解密相关异常
 *
 * @author believeyourself
 */
public class CipherRuntimeException extends RuntimeException {

    /**
     * Call the superior
     */
    public CipherRuntimeException() {
        super();
    }

    /**
     * Call the superior
     *
     * @param message the detail message
     */
    public CipherRuntimeException(String message) {
        super(message);
    }

    /**
     * Call the superior
     *
     * @param message the detail message (which is saved for later retrieval
     *                by the {@link #getMessage()} method).
     * @param cause   the cause (which is saved for later retrieval by the
     *                {@link #getCause()} method).  (A <tt>null</tt> value is
     *                permitted, and indicates that the cause is nonexistent or
     *                unknown.)
     * @since 1.4
     */
    public CipherRuntimeException(String message, Throwable cause) {
        super(message, cause);
    }

    /**
```

```
     * Call the superior
     *
     * @param cause the cause (which is saved for later retrieval by the
     *              {@link #getCause()} method).  (A <tt>null</tt> value is
     *              permitted, and indicates that the cause is nonexistent or
     *              unknown.)
     * @since 1.4
     */
    public CipherRuntimeException(Throwable cause) {
        super(cause);
    }

    /**
     * Call the superior
     *
     * @param message           the detail message.
     * @param cause             the cause.  (A {@code null} value is permitted,
     *                          and indicates that the cause is nonexistent or unknown.)
     * @param enableSuppression  whether or not suppression is enabled
     *                          or disabled
     * @param writableStackTrace whether or not the stack trace should
     *                          be writable
     * @since 1.7
     */
    protected CipherRuntimeException(String message, Throwable cause,
                                     boolean enableSuppression,
                                     boolean writableStackTrace) {
        super(message, cause, enableSuppression, writableStackTrace);
    }
}
```

定义Keystore操作工具类

了解更多请参见keystore API

```
package com.weiquding.safeKeyboard.common.util;

import com.weiquding.safeKeyboard.common.exception.CipherRuntimeException;

import java.io.BufferedInputStream;
import java.io.File;
import java.io.FileInputStream;
import java.io.InputStream;
import java.security.KeyStore;
import java.security.KeyStoreException;
import java.security.cert.Certificate;
import java.security.cert.CertificateFactory;
import java.security.interfaces.RSAPrivateKey;
import java.security.interfaces.RSAPublicKey;

/**
 * 密钥库及证书操作
 * @author believeyourself
 */
```

```
public class KeystoreUtil {

    /**
     * Changes the password used to protect the integrity of the keystore contents.
     */
    private static final String STORE_TYPE = "PKCS12";


    /**
     * Loading a Particular Keystore into Memory
     *
     * @param path      密钥库地址
     * @param storePass 密钥库密码
     * @return KeyStore
     */
    public static KeyStore loadKeyStore(String path, String storePass) {
        try (
                InputStream inputStream = new FileInputStream(new File(path));
        ) {
            KeyStore keyStore = KeyStore.getInstance(STORE_TYPE);
            keyStore.load(inputStream, storePass.toCharArray());
            return keyStore;
        } catch (Exception e) {
            throw new CipherRuntimeException("An error occurred while obtaining an instance of
keystore ", e);
        }
    }

    /**
     * 获取RSAPrivateKey实例
     *
     * @param keyStore 密钥库
     * @param alias    key别名
     * @param keyPass  key密钥
     * @return RSAPrivateKey
     */
    public static RSAPrivateKey getRSAPrivateKey(KeyStore keyStore, String alias, String keyPas
) {
        try {
            return (RSAPrivateKey) keyStore.getKey(alias, keyPass.toCharArray());
        } catch (Exception e) {
            throw new CipherRuntimeException("An error occurred while obtaining an instance of
RSAPrivateKey ", e);
        }
    }

    /**
     * 获取Certificate实例
     *
     * @param keyStore 密钥库
     * @param alias    key别名
     * @return Certificate实例
     */
    public static Certificate getCertificate(KeyStore keyStore, String alias) {
```

```
        try {
            return keyStore.getCertificate(alias);
        } catch (KeyStoreException e) {
            throw new CipherRuntimeException("An error occurred while obtaining an instance of
Certificate ", e);
        }
    }

    /**
     * 加载RSA证书
     *
     * @param path 证书路径
     * @return RSA证书
     */
    public static Certificate loadCertificate(String path) {
        try (
                FileInputStream fis = new FileInputStream(path);
                BufferedInputStream bis = new BufferedInputStream(fis);
        ) {
            CertificateFactory cf = CertificateFactory.getInstance("X.509");
            return cf.generateCertificate(bis);
        } catch (Exception e) {
            throw new CipherRuntimeException("An error occurred while loading Certificate ", e);
        }
    }

    /**
     * 获取RSA公钥
     *
     * @param certificate 证书
     * @return RSA公钥
     */
    public static RSAPublicKey getRSAPublicKey(Certificate certificate) {
        return (RSAPublicKey) certificate.getPublicKey();
    }


}
```

定义非对称加解密工具类

```
package com.weiquding.safeKeyboard.common.util;

import com.weiquding.safeKeyboard.common.exception.CipherRuntimeException;

import javax.crypto.Cipher;
import java.nio.charset.StandardCharsets;
import java.security.Signature;
import java.security.interfaces.RSAPrivateKey;
import java.security.interfaces.RSAPublicKey;

/**
 * RSA算法相关实现
 * 私钥签名 公钥验签
```

```
 * 公钥加密 公钥解密
 *
 * @author believeyourself
 */
public class RSAUtil {

    private static final String ALGORITHM = "RSA";

    /**
     * 签名算法
     */
    private static final String SIGNATURE_ALGORITHM = "SHA256withRSA";

    /**
     * RSA私钥签名
     *
     * @param privateKey 私钥
     * @param data       待签名文本字节
     * @return 签名
     */
    public static byte[] signByRSAPrivateKey(RSAPrivateKey privateKey, byte[] data) {
        try {
            Signature sign = Signature.getInstance(SIGNATURE_ALGORITHM);
            sign.initSign(privateKey);
            sign.update(data);
            return sign.sign();
        } catch (Exception e) {
            throw new CipherRuntimeException("An error occurred while signing with RSAPrivate
ey", e);
        }
    }

    /**
     * RSA公钥验签
     *
     * @param publicKey 公钥
     * @param data      原数据文本字节
     * @param sign      签名
     * @return 是否验证通过
     */
    public static boolean verifySignByRSAPublicKey(RSAPublicKey publicKey, byte[] data, byte[]
sign) {
        try {
            Signature signature = Signature.getInstance(SIGNATURE_ALGORITHM);
            signature.initVerify(publicKey);
            signature.update(data);
            return signature.verify(sign);
        } catch (Exception e) {
            throw new CipherRuntimeException("An error occurred while using RSAPublicKey to v
rify the signature", e);
        }
    }

    /**
```

```
     * RSA公钥加密
     *
     * @param publicKey 公钥
     * @param data      明文数据
     * @return 加密数据
     */
    public static byte[] encryptByRSAPublicKey(RSAPublicKey publicKey, byte[] data) {
        try {
            Cipher cipher = Cipher.getInstance(ALGORITHM);
            cipher.init(Cipher.ENCRYPT_MODE, publicKey);
            byte[] bytes = cipher.doFinal(data);
            return bytes;
        } catch (Exception e) {
            throw new CipherRuntimeException("An error occurred while using RSAPublicKey for
ncryption", e);
        }
    }

    /**
     * RSA私钥解密
     *
     * @param privateKey 私钥
     * @param data       明文数据
     * @return 解密数据
     */
    public static byte[] decryptByRSAPrivateKey(RSAPrivateKey privateKey, byte[] data) {
        try {
            Cipher cipher = Cipher.getInstance(ALGORITHM);
            cipher.init(Cipher.DECRYPT_MODE, privateKey);
            byte[] bytes = cipher.doFinal(data);
            return bytes;
        } catch (Exception e) {
            throw new CipherRuntimeException("An error occurred while decrypting with RSAPriv
teKey", e);
        }
    }

}
```

随机数生成组件

The SecureRandom class is an engine class (see Engine Classes and Algorithms) that provides ryptographically strong random numbers, either by accessing a pseudo-random number gene ator (PRNG), a deterministic algorithm that produces a pseudo-random sequence from an init al seed value, or by reading a native source of randomness (for example, /dev/random or a tr e random number generator). One example of a PRNG is the Deterministic Random Bits Gene ator (DRBG) as specified in NIST SP 800-90Ar1. Other implementations may produce true ran om numbers, and yet others may use a combination of both techniques. A cryptographically s rong random number minimally complies with the statistical random number generator tests pecified in FIPS 140-2, Security Requirements for Cryptographic Modules, section 4.9.1.

了解更多请参见The SecureRandom Class

```java
package com.weiquding.safeKeyboard.common.util;

import com.weiquding.safeKeyboard.common.cache.KeyInstance;

import java.security.SecureRandom;
import java.util.HashMap;
import java.util.Map;

/**
 * 随机数生成
 *
 * @author believeyourself
 */
public class RandomUtil {

    /**
     * 获取指定长度随机数
     *
     * @param length 长度
     * @return 随机数字节
     */
    public static byte[] generateRandomBytes(int length) {
        byte[] bytes = new byte[length];
        SecureRandom random = new SecureRandom();
        random.nextBytes(bytes);
        return bytes;
    }

    /**
     * 生成客户端随机数，先用RSA公钥加密，再base64编码
     * @return 加密客户端随机数
     */
    public static Map<String, String> generateRNCAndPMS(){
        byte[] RNC = RandomUtil.generateRandomBytes(32);
        byte[] PMS = RandomUtil.generateRandomBytes(48);
        byte[] seed = new byte[80];
        System.arraycopy(RNC, 0, seed, 0, RNC.length);
        System.arraycopy(PMS, 0, seed, RNC.length, PMS.length);
        byte[] bytes = RSAUtil.encryptByRSAPublicKey(KeyInstance.RSA_PUBLIC_KEY, seed);
        Map<String, String> map = new HashMap<>();
        map.put("RNC", Base64.getEncoder().encodeToString(RNC));
        map.put("PMS", Base64.getEncoder().encodeToString(PMS));
        map.put("cipherText", Base64.getEncoder().encodeToString(bytes));
        return map;
    }


    /***
     * 生成服务端随机数并进行base64,再生成对应签名
     * @return base64随机数及签名
     */
    public static Map<String, String> generateRNSAndSign(){
        Map<String,String> map = new HashMap<>();
        byte[] seed = RandomUtil.generateRandomBytes(32);
```

```
        byte[] base64 = Base64.getEncoder().encode(seed);
        byte[] sign = RSAUtil.signByRSAPrivateKey(KeyInstance.RSA_PRIVATE_KEY, base64);
        map.put("RNS", new String(base64));
        map.put("sign", Base64.getEncoder().encodeToString(sign));
        return map;
    }

}
```

Base64组件的选择

基于JDK10的java.util.Base64打乱字符表映射，实现一个简化的自定义Base64，提供简单的单表映加密能力。

Base64算法主要是对给定的字符以与字符编码对应的十进制数为基准，做编码操作：

1. 将给定的字符串以字符为单位转换为对应的字符编码(如ASCII码)。
2. 将获得的字符编码转换为二进制码。
3. 对获得的二进制码做分组转换操作，每3个8进制码为一组，转换为每6个6位二进制码为一组（不足位时低位补0）。这是一个分组变化的过程，3个8位二进制码和4个6位二进制码的长度都是24位。
4. 对获得的4个6位二进制码补位，向6位二进制码添加2位高位0，组成4个8位二进制。
5. 将获得的4个8位二进制码转换为十进制码。
6. 将获得的十进制码转换为Base64字符表中对应的字符。

替换如下部分代码

```
        /**
         * This array is a lookup table that translates 6-bit positive integer
         * index values into their "Base64 Alphabet" equivalents as specified
         * in "Table 1: The Base64 Alphabet" of RFC 2045 (and RFC 4648).
         */
        private static final char[] toBase64 = {
                'A', '+', '9', '8', 'E', '6', 'G', 'H', '3', 'J', '1', 'L', 'M',
                'N', 'O', 'P', 'Q', 'u', 'S', 's', 'U', 'V', 'W', 'o', 'n', 'Z',
                'l', 'k', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'b', 'a', 'm',
                'Y', 'X', 'p', 'q', 'r', 'T', 't', 'R', 'v', 'w', 'x', 'y', 'z',
                '0', 'K', '2', 'I', '4', '5', 'F', '7', 'D', 'C', 'B', '/'
        };

        /**
         * It's the lookup table for "URL and Filename safe Base64" as specified
         * in Table 2 of the RFC 4648, with the '+' and '/' changed to '-' and
         * '_'. This table is used when BASE64_URL is specified.
         */
        private static final char[] toBase64URL = {
                'A', '-', '9', '8', 'E', '6', 'G', 'H', '3', 'J', '1', 'L', 'M',
                'N', 'O', 'P', 'Q', 'u', 'S', 's', 'U', 'V', 'W', 'o', 'n', 'Z',
                'l', 'k', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'b', 'a', 'm',
                'Y', 'X', 'p', 'q', 'r', 'T', 't', 'R', 'v', 'w', 'x', 'y', 'z',
```

```
            '0', 'K', '2', 'I', '4', '5', 'F', '7', 'D', 'C', 'B', '_'
        };
```

对称加密算法组件

对称算法使用AES-GCM-256,有关算法的实现请参考Recommendation for Block Cipher Modes of Operation:Galois/Counter Mode (GCM) and GMAC

```
package com.weiquding.safeKeyboard.common.util;

import com.weiquding.safeKeyboard.common.exception.CipherRuntimeException;
import lombok.extern.slf4j.Slf4j;

import javax.crypto.Cipher;
import javax.crypto.KeyGenerator;
import javax.crypto.SecretKey;
import javax.crypto.spec.GCMParameterSpec;
import javax.crypto.spec.SecretKeySpec;
import java.security.NoSuchAlgorithmException;
import java.security.SecureRandom;
import java.util.Arrays;

/**
 * AES工具
 *
 * @author believeyourself
 */
@Slf4j
public class AESUtil {

    private static final String ALGORITHM = "AES";

    private static final String ALGORITHM_PADDING = "AES/GCM/NoPadding";

    private static final int AES_KEY_SIZE = 256;

    private static final int GCM_IV_LENGTH = 12;

    private static final int GCM_TAG_LENGTH = 16;



    /**
     * 获取向量
     * @return 长度为12的向量
     */
    public static byte[] ivParameter(){
        return RandomUtil.generateRandomBytes(GCM_IV_LENGTH);
    }


    /**
     * 生成指定位数的AES密钥
```

```
     *
     * @param bitLen 位数
     * @return AES密钥
     */
    public static SecretKey generateAESKey(int bitLen) {
        try {
            KeyGenerator keyGenerator = KeyGenerator.getInstance(ALGORITHM);
            SecureRandom random = new SecureRandom();
            keyGenerator.init(bitLen, random);
            return keyGenerator.generateKey();
        } catch (NoSuchAlgorithmException e) {
            throw new CipherRuntimeException("An error occurred while generating the AES key",
e);
        }
    }


    /**
     * 使用AES/GCM/NoPadding进行解密
     *
     * @param encoded   密钥数据
     * @param iv        向量
     * @param data 数据
     * @return 解密后数据
     */
    public static byte[] decryptByAESKey(byte[] encoded, byte[] iv, byte[] data) {
        try {
            // Get Cipher Instance
            Cipher cipher = Cipher.getInstance(ALGORITHM_PADDING);
            // Create SecretKeySpec
            SecretKeySpec keySpec = new SecretKeySpec(encoded, ALGORITHM);
            // Create GCMParameterSpec
            GCMParameterSpec gcmParameterSpec = new GCMParameterSpec(GCM_TAG_LENG
H * 8, iv);
            log.info("decrypt key : [{}]", Arrays.toString(keySpec.getEncoded()));
            // Initialize Cipher for DECRYPT_MODE
            cipher.init(Cipher.DECRYPT_MODE, keySpec, gcmParameterSpec);
            // Perform Decryption
            return cipher.doFinal(data);
        } catch (Exception e) {
            throw new CipherRuntimeException("An error occurred while using the AES key for de
ryption", e);
        }
    }

    /**
     * 使用AES/GCM/NoPadding进行加密
     *
     * @param encoded    密钥
     * @param iv         向量
     * @param data 加密数据
     * @return 解密后数据
     */
    public static byte[] encryptByAESKey(byte[] encoded, byte[] iv, byte[] data) {
```

```
        try {
            // Get Cipher Instance
            Cipher cipher = Cipher.getInstance(ALGORITHM_PADDING);

            // Create SecretKeySpec
            SecretKeySpec keySpec = new SecretKeySpec(encoded, ALGORITHM);

            // Create GCMParameterSpec
            GCMParameterSpec gcmParameterSpec = new GCMParameterSpec(GCM_TAG_LENG
H * 8, iv);

            log.info("encrypt key : [{}]", Arrays.toString(keySpec.getEncoded()));

            // Initialize Cipher for ENCRYPT_MODE
            cipher.init(Cipher.ENCRYPT_MODE, keySpec, gcmParameterSpec);

            // Perform Encryption
            return cipher.doFinal(data);
        } catch (Exception e) {
            throw new CipherRuntimeException("An error occurred while using the AES key for en
ryption", e);
        }
    }


}
```

摘要算法的实现

```
package com.weiquding.safeKeyboard.common.util;

import com.weiquding.safeKeyboard.common.exception.CipherRuntimeException;

import javax.crypto.Mac;
import javax.crypto.spec.SecretKeySpec;

/**
 * HMAC摘要工具类
 *
 * @author believeyourself
 */
public class HmacUtil {

    public static final String HMAC_SHA_256 = "HmacSHA256";

    /**
     * 获取mac实例
     * @param algorithm 算法
     * @param key 密钥
     * @return Mac
     */
    public static Mac getMacInstance(String algorithm, byte[] key) {
```

```
        if (key == null) {
            throw new IllegalArgumentException("Null key");
        } else {
            try {
                SecretKeySpec keySpec = new SecretKeySpec(key, algorithm);
                Mac mac = Mac.getInstance(algorithm);
                mac.init(keySpec);
                return mac;
            } catch (Exception e) {
                throw new CipherRuntimeException("An error occurred while getting a MAC instan
e", e);
            }
        }
    }
}
```

Pseudo-random function实现及并生成MS及keyblock

```
package com.weiquding.safeKeyboard.common.util;

import javax.crypto.Mac;

/**
 * The pseudo random function
 *
 * @author believeyourself
 */
public class PRFUtil {

    /**
     * master secret
     */
    private static final byte[] MASTER_LABEL = new byte[]{109, 97, 115, 116, 101, 114, 32, 115,
01, 99, 114, 101, 116};

    /**
     * key expansion
     */
    private static final byte[] KEY_BLOCK_LABEL = {107, 101, 121, 32, 101, 120, 112, 97, 110, 115,
105, 111, 110};

    /**
     * 生成伪随机数
     *
     * @param key       密钥
     * @param label     标签
     * @param seed      种子
     * @param numBlocks 迭代次数
     * @return 伪随机数
     */
    public static byte[] prf(byte[] key, byte[] label, byte[] seed, int numBlocks) {
        if (key == null
                || label == null
                || seed == null
```

```
            || numBlocks == 0
    ) {
        throw new IllegalArgumentException();
    }
    byte[] newSeed = new byte[label.length + seed.length];
    System.arraycopy(label, 0, newSeed, 0, label.length);
    System.arraycopy(seed, 0, newSeed, label.length, seed.length);
    Mac mac = HmacUtil.getMacInstance(HmacUtil.HMAC_SHA_256, key);
    byte[] digest = mac.doFinal(newSeed);
    byte[] output = new byte[0];
    for (int i = 0; i < numBlocks; i++) {
        byte[] toDigest = new byte[digest.length + newSeed.length];
        System.arraycopy(digest, 0, toDigest, 0, digest.length);
        System.arraycopy(newSeed, 0, toDigest, digest.length, newSeed.length);
        byte[] out = mac.doFinal(toDigest);

        byte[] newOutput = new byte[output.length + out.length];
        System.arraycopy(output, 0, newOutput, 0, output.length);
        System.arraycopy(out, 0, newOutput, output.length, out.length);
        output = newOutput;
        digest = mac.doFinal(digest);
    }
    return output;
}

/**
 * Compute the master secret from the premaster secret
 * 生成Master Secret
 *
 * @param PMS pre-master secret
 * @param RNC random number of client
 * @param RNS random number of server
 * @return Master Secret
 */
public static byte[] generateMasterSecret(byte[] PMS, byte[] RNC, byte[] RNS) {
    if (PMS == null
            || RNC == null
            || RNS == null
    ) {
        throw new IllegalArgumentException();
    }
    byte[] seed = new byte[RNC.length + RNS.length];
    System.arraycopy(RNC, 0, seed, 0, RNC.length);
    System.arraycopy(RNS, 0, seed, RNC.length, RNS.length);
    byte[] randoms = prf(PMS, MASTER_LABEL, seed, 2);
    byte[] ms = new byte[48];
    System.arraycopy(randoms, 0, ms, 0, ms.length);
    return ms;
}

/**
 * Generate the key block
 * 生成以下密钥数据
 * keyBlock[0]:客户端Mac摘要密钥：clientMacKey[32]
```

```
     * keyBlock[1]:服务端Mac摘要密钥：serverMacKey[32]
     * keyBlock[2]:客户端AES对称加密密钥：clientWriteKey[32]
     * keyBlock[3]:服务端AES对称加密密钥：serverWriteKey[32]
     *
     * @param MS  master secret
     * @param RNC random number of client
     * @param RNS random number of server
     * @return key block
     */
    public static byte[][] generateKeyBlock(byte[] MS, byte[] RNC, byte[] RNS) {
        if (MS == null
                || RNC == null
                || RNS == null
        ) {
            throw new IllegalArgumentException();
        }
        byte[] seed = new byte[RNC.length + RNS.length];
        System.arraycopy(RNC, 0, seed, 0, RNC.length);
        System.arraycopy(RNS, 0, seed, RNC.length, RNS.length);
        byte[] randoms = prf(MS, KEY_BLOCK_LABEL, seed, 4);
        byte[][] keyBlock = new byte[4][32];
        System.arraycopy(randoms, 0, keyBlock[0], 0, 32);
        System.arraycopy(randoms, 32, keyBlock[1], 0, 32);
        System.arraycopy(randoms, 64, keyBlock[2], 0, 32);
        System.arraycopy(randoms, 96, keyBlock[3], 0, 32);
        return keyBlock;
    }

    /**
     * 生成key block
     *
     * @param PMS base64 pms
     * @param RNC base64 rnc
     * @param RNS base64 rns
     * @return
     */
    public static byte[][] generateKeyBlock(String PMS, String RNC, String RNS) {
        byte[] RNCBytes = Base64.getDecoder().decode(RNC);
        byte[] RNSBytes = Base64.getDecoder().decode(RNS);
        byte[] MS = generateMasterSecret(Base64.getDecoder().decode(PMS), RNCBytes, RNSBy
es);
        return generateKeyBlock(MS, RNCBytes, RNSBytes);
    }

}
```

客户端操作

1. 根据时序图，产生随机数过程中客户端生成RNC和PMS，服务端生成RNS，并进行交换。
2. 提供密码加密能力

```
package com.weiquding.safeKeyboard.controller;
```

```
import com.weiquding.safeKeyboard.common.cache.GuavaCache;
import com.weiquding.safeKeyboard.common.cache.KeyInstance;
import com.weiquding.safeKeyboard.common.exception.CipherRuntimeException;
import com.weiquding.safeKeyboard.common.util.*;
import lombok.extern.slf4j.Slf4j;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.HttpEntity;
import org.springframework.http.HttpHeaders;
import org.springframework.http.MediaType;
import org.springframework.util.LinkedMultiValueMap;
import org.springframework.util.MultiValueMap;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.bind.annotation.RestController;
import org.springframework.web.client.RestTemplate;

import javax.crypto.Mac;
import java.net.URLEncoder;
import java.nio.charset.StandardCharsets;
import java.util.Arrays;
import java.util.HashMap;
import java.util.Map;

/**
 * 模拟获取密钥与密文
 *
 * @author believeyourself
 */
@Slf4j
@RestController
public class ClientController {

    @Autowired
    private RestTemplate restTemplate;

    /**
     *
     */
    @RequestMapping("/generateRNC")
    public Map<String, String> generateRNC(String sessionId) {
        Map<String, String> rncAndPMS = RandomUtil.generateRNCAndPMS();
        HttpHeaders headers = new HttpHeaders();
        headers.setContentType(MediaType.APPLICATION_FORM_URLENCODED);
        MultiValueMap<String, String> map = new LinkedMultiValueMap<>();
        map.add("RNC", rncAndPMS.get("cipherText"));
        map.add("sessionId", sessionId);
        HttpEntity<MultiValueMap<String, String>> request = new HttpEntity<>(map, headers);

        Map<String, String> retVal = restTemplate.postForObject("http://localhost:8082/genera
eRNS", request, Map.class);
        String RNS = retVal.get("RNS");
        String sign = retVal.get("sign");
        boolean verify = RSAUtil.verifySignByRSAPublicKey(KeyInstance.RSA_PUBLIC_KEY, RNS.g
tBytes(), Base64.getDecoder().decode(sign));
```

```
        if (!verify) {
            throw new CipherRuntimeException("Signature corrupted");
        }
        rncAndPMS.put("RNS", RNS);
        rncAndPMS.remove("cipherText");
        GuavaCache.CLIENT_CACHE.put(sessionId, rncAndPMS);
        // 测试用
        return rncAndPMS;
    }

    /**
     * 获取密文
     *
     * @param password
     * @return
     */
    @RequestMapping("/getEncryptedPassword")
    public Map<String, String> getEncryptedPassword(@RequestParam("password") String pa
sword, String sessionId) {
        Map<String, String> map = new HashMap<>(1);
        byte[] ivParameter = AESUtil.ivParameter();
        Map<String, String> session = GuavaCache.CLIENT_CACHE.getIfPresent(sessionId);
        String PMS = session.get("PMS");
        String RNC = session.get("RNC");
        String RNS = session.get("RNS");
        byte[][] keyBlock = PRFUtil.generateKeyBlock(PMS, RNC, RNS);
        byte[] pwdBytes = password.getBytes(StandardCharsets.UTF_8);
        byte[] randomBytes = RandomUtil.generateRandomBytes(32);
        // 拼接randomBytes
        byte[] confusionPwd = new byte[randomBytes.length + pwdBytes.length];
        System.arraycopy(randomBytes, 0, confusionPwd, 0, randomBytes.length);
        System.arraycopy(pwdBytes, 0, confusionPwd, randomBytes.length, pwdBytes.length);
        // 摘要
        Mac macInstance = HmacUtil.getMacInstance(HmacUtil.HMAC_SHA_256, keyBlock[0]);
        byte[] macDigest = macInstance.doFinal(confusionPwd);
        // 对称加密
        byte[] encryptedPwd = AESUtil.encryptByAESKey(keyBlock[2], ivParameter, confusionPwd
;
        log.debug("clientMacKey:{}", Arrays.toString(keyBlock[0]));
        log.debug("clientWriteKey:{}", Arrays.toString(keyBlock[2]));
        log.debug("ivParameter:{}", Arrays.toString(ivParameter));
        log.debug("encryptedPwd:{}", Arrays.toString(encryptedPwd));
        log.debug("macDigest:{}", Arrays.toString(macDigest));

        // 拼接
        byte[] cipherText = new byte[ivParameter.length + encryptedPwd.length + macDigest.le
gth];
        System.arraycopy(ivParameter, 0, cipherText, 0, ivParameter.length);
        System.arraycopy(encryptedPwd, 0, cipherText, ivParameter.length, encryptedPwd.length
;
        System.arraycopy(macDigest, 0, cipherText, ivParameter.length + encryptedPwd.length,
acDigest.length);

        String encryptedPwdString = Base64.getEncoder().encodeToString(cipherText);
```

```
        map.put("password", URLEncoder.encode(encryptedPwdString, StandardCharsets.UTF_8))

        return map;
    }

    /**
     * 提交密文
     *
     * @param password
     * @return
     */
    @RequestMapping("/submitEncryptedPassword")
    public Map<String, String> submitEncryptedPassword(@RequestParam("password") String
password, String sessionId) {
        HttpHeaders headers = new HttpHeaders();
        headers.setContentType(MediaType.APPLICATION_FORM_URLENCODED);
        MultiValueMap<String, String> map = new LinkedMultiValueMap<>();
        map.add("password", URLEncoder.encode(password, StandardCharsets.UTF_8));
        map.add("sessionId", sessionId);
        HttpEntity<MultiValueMap<String, String>> request = new HttpEntity<>(map, headers);

        Map<String, String> retVal = restTemplate.postForObject("http://localhost:8082/submit
ncryptedPassword", request, Map.class);

        return retVal;

    }

}
```

服务端操作

```
package com.weiquding.safeKeyboard.controller;

import com.weiquding.safeKeyboard.common.cache.GuavaCache;
import com.weiquding.safeKeyboard.common.cache.KeyInstance;
import com.weiquding.safeKeyboard.common.exception.CipherRuntimeException;
import com.weiquding.safeKeyboard.common.util.*;
import lombok.extern.slf4j.Slf4j;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.bind.annotation.RestController;

import java.net.URLDecoder;
import java.nio.charset.StandardCharsets;
import java.util.Arrays;
import java.util.HashMap;
import java.util.Map;

/**
 * 模拟生成密钥与密文
 *
```

```
 * @author believeyourself
 */
@Slf4j
@RestController
public class ServerController {

    @RequestMapping(value = "/generateRNS", method = RequestMethod.POST)
    public Map<String, String> generateRNC(@RequestParam("RNC") String cipherText, String
sessionId) {
        byte[] RNCAndPMS = RSAUtil.decryptByRSAPrivateKey(KeyInstance.RSA_PRIVATE_KEY, B
se64.getDecoder().decode(cipherText));
        byte[] RNC = new byte[32];
        byte[] PMS = new byte[48];
        System.arraycopy(RNCAndPMS, 0, RNC, 0, RNC.length);
        System.arraycopy(RNCAndPMS, RNC.length, PMS, 0, PMS.length);
        Map<String, String> map = RandomUtil.generateRNSAndSign();
        Map<String, String> session = new HashMap<>();
        session.put("RNC", Base64.getEncoder().encodeToString(RNC));
        session.put("PMS", Base64.getEncoder().encodeToString(PMS));
        session.put("RNS", map.get("RNS"));
        GuavaCache.SERVER_CACHE.put(sessionId, session);
        return map;
    }

    @RequestMapping(value = "/submitEncryptedPassword", method = RequestMethod.POST)
    public Map<String, String> submitEncryptedPassword(@RequestParam("password") String
password, String sessionId) {
        password = URLDecoder.decode(password, StandardCharsets.UTF_8);
        // 密钥生成
        Map<String, String> session = GuavaCache.SERVER_CACHE.getIfPresent(sessionId);
        String PMS = session.get("PMS");
        String RNC = session.get("RNC");
        String RNS = session.get("RNS");
        byte[][] keyBlock = PRFUtil.generateKeyBlock(PMS, RNC, RNS);
        // 切分iv[12] + cipherText[?] + macSign[32]
        byte[] base64DecodeBytes = Base64.getDecoder().decode(password);
        byte[] iv = new byte[12];
        byte[] macDigest = new byte[32];
        byte[] cipherText = new byte[base64DecodeBytes.length - iv.length - macDigest.length];
        System.arraycopy(base64DecodeBytes, 0, iv, 0, iv.length);
        System.arraycopy(base64DecodeBytes, iv.length, cipherText, 0, cipherText.length);
        System.arraycopy(base64DecodeBytes, iv.length + cipherText.length, macDigest, 0, macD
gest.length);
        log.debug("clientMacKey:{}", Arrays.toString(keyBlock[0]));
        log.debug("clientWriteKey:{}", Arrays.toString(keyBlock[2]));
        log.debug("ivParameter:{}", Arrays.toString(iv));
        log.debug("encryptedPwd:{}", Arrays.toString(cipherText));
        log.debug("macDigest:{}", Arrays.toString(macDigest));
        // 对称解密
        byte[] encryptedPwd = AESUtil.decryptByAESKey(keyBlock[2], iv, cipherText);
        // 验证摘要
        byte[] serverMacDigest = HmacUtil.getMacInstance(HmacUtil.HMAC_SHA_256, keyBlock
0]).doFinal(encryptedPwd);
        if (!Arrays.equals(macDigest, serverMacDigest)) {
```

```
            throw new CipherRuntimeException("Password digest authentication failed");
        }
        // 取出真实的密码
        byte[] pwdBytes = new byte[encryptedPwd.length - 32];
        System.arraycopy(encryptedPwd, 32, pwdBytes, 0, pwdBytes.length);
        // 真实的密码
        String pwd = new String(pwdBytes, StandardCharsets.UTF_8);
        log.info("The real password is [{}]", pwd);
        Map<String, String> retMap = new HashMap<>();
        retMap.put("pwd", pwd);
        return retMap;
    }
}
```

完整代码实现

完整的代码实现见Github仓库SafeKeyboard

测试结果

Postman 测试报文如下：


```
{
    "variables": [],
    "info": {
        "name": "safeKeyboard",
        "_postman_id": "ffbfa9dc-47cb-9420-dee0-bda343503f57",
        "description": "",
        "schema": "https://schema.getpostman.com/json/collection/v2.0.0/collection.json"
    },
    "item": [
        {
            "name": "http://localhost:8081/generateRNC",
            "request": {
                "url": {
                    "raw": "http://localhost:8081/generateRNC?sessionId=testID",
                    "protocol": "http",
                    "host": [
                        "localhost"
                    ],
                    "port": "8081",
                    "path": [
                        "generateRNC"
                    ],
                    "query": [
                        {
                            "key": "sessionId",
                            "value": "testID"
                        }
                    ],
                    "variable": []
                },
```

"method": "POST",
"header": [],
"body": {
"mode": "raw",
"raw": ""
},
"description": "生成随机数"
},
"response": []
},
{
"name": "http://localhost:8081/getEncryptedPassword?password=123456",
"request": {
"url": {
"raw": "http://localhost:8081/getEncryptedPassword?password=123456&sessionI
=testID",
"protocol": "http",
"host": [
"localhost"
],
"port": "8081",
"path": [
"getEncryptedPassword"
],
"query": [
{
"key": "password",
"value": "123456"
},
{
"key": "sessionId",
"value": "testID"
}
],
"variable": []
},
"method": "POST",
"header": [],
"body": {
"mode": "raw",
"raw": ""
},
"description": "获取密文"
},
"response": []
},
{
"name": "http://localhost:8081/submitEncryptedPassword?password=wmwcsZo20c+u
VA7ll1RBc4bp4zjNMsdAc2gxz1RRlx6EH7EQXvRB5XdN7tJxZBhL7M9Yq0p+6yELEm6xhXlKVp
mFFx4J93fSZlWtviGMSi413CjB3A3kGdThCf+J7nPLY=",
"request": {
"url": {
"raw": "http://localhost:8081/submitEncryptedPassword?password=qY%2F3Q5W
AxEp8XCFZm%2FowDYBGj60a68jDmecWAotHC1%2FNWDZXE%2BfueX8yf8QACvfKw2eyBpp

WRusHfUzdRKiz21oAQtHnbYAJIEGagby7AaUsk7Tul5E31sARAuqte6XNb%3D&sessionId=testD",
"protocol": "http",
"host": [
"localhost"
],
"port": "8081",
"path": [
"submitEncryptedPassword"
],
"query": [
{
"key": "password",
"value": "qY%2F3Q5WfAxEp8XCFZm%2FowDYBGj60a68jDmecWAotHC1%2NWDZXE%2BfueX8yf8QACvfKw2eyBppWWRusHfUzdRKiz21oAQtHnbYAJIEGagby7AaUsk7TulE31sARAuqte6XNb%3D",
"equals": true,
"description": ""
},
{
"key": "sessionId",
"value": "testID",
"equals": true,
"description": ""
}
],
"variable": []
},
"method": "POST",
"header": [],
"body": {},
"description": "提交密文"
},
"response": []
}
]
}

参考资料

1. Ssl_handshake_with_two_way_authentication_with_certificates-1.pdf
2. Java Platform, Standard Edition Security Developer’s Guide
3. TLS, Pre-Master Secrets and Master Secrets
4. The Transport Layer Security (TLS) Protocol Version 1.2
5. Crypto in practice: What happens when you connect to a secure web erver
6. 【原创】浅析密码学在互联网支付中的应用|RSA,Hash,AES,DES,3DES,SHA1,SHA256,MD5,SSL,Priate Key,Public Key
7. Cryptography in Mobile Apps
8. Encryption

9. Public-key_cryptography
10. Google Pay for Payments 针对商家的付款数据加密
11. Dynamic Rule Encryption for Mobile Payment
12. Mobile Payment Method Based on Public-Key Cryptography
13. Java加密与解密的艺术.第二版
14. Pseudo-random function: prf.go
15. prf_test
16. TLS
17. 通过 HTTPS 和 SSL 确保安全
18. 梆梆SDKs详细分析（2） - 安全键盘SDK揭秘
19. 安全软键盘SDK
20. 密码键盘
21. PDonkey/SafeKeyboard
22. StomHong/CustomizeKeyboard
23. Android密码安全输入键盘
24. How is the Premaster secret used in TLS generated?
25. Method and system for securing a payment transaction with trusted code base
26. Online Payment Fraud Prevention Using Cryptographic Algorithm TDES
27. Securing Sensitive Data Using Payload Encryption
28. A MULTI-FACTOR SECURITY PROTOCOL FOR WIRELESS PAYMENT- SECURE WEB AUTHENICATION USING MOBILE DEVICES
29. Recommendation for Block Cipher Modes of Operation:Galois/Counter Mode (GCM) and MAC
30. Java AES 256 GCM Encryption and Decryption Example | JCE Unlimited Strength
31. NIST Special Publication 800-63B: Digital Identity Guidelines: Authentication and Lifecycle Management
32. Java Platform, Standard Edition Tools Reference:keytool
33. PBKDF2