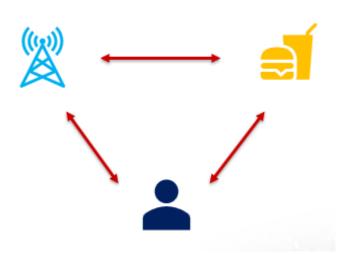
# 加密&隐私保护外卖模拟系统

### 1. 应用场景

本系统模拟了现实中的顾客-外卖平台-通讯运营商的简单交互过程。首先,该系统利用 OpenSSL 生成了根证书,然后为外卖与运营商生成 X509 证书,顾客发出请求时会将顾客发送的消息进行双重签名,使得外卖平台只能得到顾客的姓名与地址信息,无法得知顾客的真实电话号码,而通讯运营商可以得知顾客的真实电话号码,无法得知其订单信息,并且通讯运营商会为顾客与外卖平台之间生成一个虚拟号码的分机,该虚拟号码支持顾客与送外卖者的互相通信。



实际情况如下图中所示,我们可以看到,外卖商是可以知道我的地址消息和下单消息的,但是他们缺无法得知我的真实电话号码与完整的姓名信息;另一方面,移动运营商可以得知我的电话号码信息与姓名信息,同时,会为我和外卖商生成一个临时的转接云分机,这提供了一个虚拟号码,该号码可以单向或者双向拨通,这很

好的保护了我的隐私。

### 2. 所实现应用的功能

- 2.1 Client 与两个 Server 之间的通信
- 2.2 客户对服务器请求证书公钥信息,并且将收到的公钥信息保存到本地。
- 2.3 客户将自己的公钥信息发送给服务器, 服务器保存到本地
- 2.4 对客户发送的信息实现了 RSA 公钥加密, Base64 编码, 求取 散列哈希值, 以及 RSA 私钥签名。
- 2.5 服务器对收到信息实现了 RSA 私钥解密, Base64 解码, 散列对比, 以及 RSA 公钥验证签名。
- 2.6 电话号码虚拟化, 生成一个基于随机数的虚拟电话号码。

# 3. 所实现应用的工作流程以及实现细节

- 3.1 生成根证书、各个交互方生成私钥、用根证书为站点分发 X509 证书。
- 3.2 客户向平台通信伊始, 会先请求平台的证书与通讯运营商的证书,

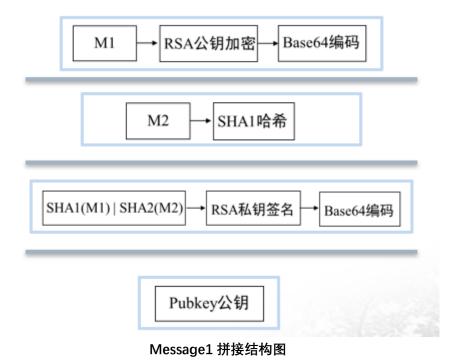
将对应的证书保存在本地。

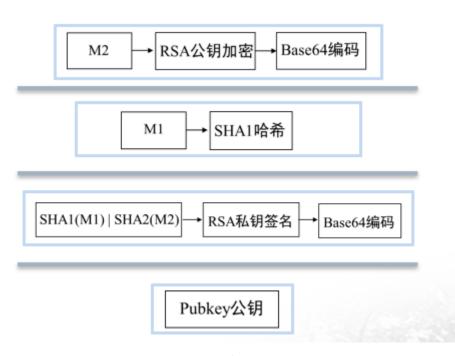
- 3.3 顾客对发送的信息分段哈希, 计算出 Hm1,Hm2,Sign(Hm12), 其中 M1 为订单信息, M2 为电话信息, Hm12 为 Hm1 与 Hm2 的拼接 的哈希值, 并且使用自己的私钥对 Hm12 进行签名, 由于 RSA 签 名后的数据为二进制数据, 需要进行 Base64 编码后才更适合信息 拼接与传输。
- 3.4 然后用外卖平台的证书对发送的信息 M1 进行 RSA 加密(实际生活中是需要使用 RSA 构建加密通道,协商加密对称加密的秘钥,然后对信息采用对称加密的方式来加密,原因是 RSA 加密信息的耗时非常大,并且大多接口对加密信息的长度也有限制。这里直接使用 RSA 加密信息的原因是客户模拟发送的信息很短,直接 RSA 加密不会带来很大开销),同时由于加密后为二进制数据,需要进行 Base64 编码后才更适合进行信息拼接传输。
- 3.5 顾客将自己的公钥信息附属在发送的信息后面,最后对外卖平台发送的信息为 Message1(Base64(RSA\_encode(M1)) | Hm2 | Base64(Sign(Hm12)) | pubkey)
- 3.6 顾客对于通讯运营商则不需要请求证书, 发送的信息为 Message2(Base64(RSA\_encode(M2)) | Hm1 | Base64(Sign(Hm12)) | pubkey)
- 3.7 外卖平台和运营商接收到消息后,需要进行以下步骤: Base64 解码, RSA 私钥解码,对顾客的公钥信息进行读取保存,使用顾客公钥进行签名验证,反馈消息。而运营商会根据电话号码生成虚

拟转接的电话号码(本系统只是产生了随机数,并没有申请虚拟号码的分机,因为穷!)

 说明:实际应用中,顾客只是对外卖平台发送了 Message1 和 Message2,而由外卖平台进行转发,这里为了方便观察与处理, 令顾客分别对外卖平台与运营商发送 Message1 与 Message2。







Message2 拼接结构图

而服务器在收到相关的 Message 后,后分别进行提取解码解密,最后进行验签,并给出对应的回馈。

## Demo 运行截图:

```
Accepted open client[ id:2 ip:127.0.0.1 socket:4 send:0 ]
                                                                                                                                                                                                  open tetenit totz tp:17.0.0.1 Socket-1 Stinds j
receive message:
HqFiSwp7u4Nr09H9+X82MQa2djaf]jZNSm2E6V4kPhVVVSRMMJQx5j0Bk
F0im+pqxd8ACQ0UyjU9LyAXO4hkm1yu1FAkn4H56eBqw5RaukhbKFBTpi
YZuQXSRikJfRe/vGI8DXFY0B7XCjtMB+tpYzJy0KYNC0rxn+0Qq7Lstqh
E0SyzVTp2tfZh0eTDouvULrmiv+j22oIOltuifopW8xvmru/V6PrudYKS
MIGFMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBQQCq8RcYTHXDuvUYr5fftk
 mn1 SHA1 digest translate:96252f20795cef8edc7765fd847206b69e409c34
hm2 SHA1 digest translate:284bdd1a36dc87765f04f4e6be9727913e446149
hm12 SHA1 digest translate:75aa463b709a48c596a85c03d576e26f8f716487
                                                                                                                                                                                                  83jbqloLDwfFnf9uM9GFEkxgcOsTZXQwrQAOg2bEMhZ5EdlTDaDK37us;
uOAW3piHxlYPnti1n1DYvjpqZE+VBswMMUqo0sMDj4kTgNNcT1IaO43vı
                                                                                                                                                                                                   WJr55uVOmcsb6iXjrQIDAOAB
                                                                                                                                                                                                          --END PUBLIC KÈY--
                                                                                                                                                                                                  M1 base64 =======:HqFiSwp7u4NrQ9H9+X82MQa2djafJjZN5m2E
1gA9wGackfZ9HnJjtW2JaF0im+pqxd8ACQOUyjU9LyAxO4nkm1yu1FAkr
messages[0]:jack
 verify success
 local pubkey transformed to string =====::
-----BEGIN PUBLIC KEY-----
----BEGIN PUBLIC KEY----
MIGFMAΘGCSqGSTB3DgDBAQUAA4GNADCB1QKBgQCqBRCYTHXDuvUYr5f1V7XvzVsS
83jbqloLDwfFnf9uM9GFEkxgcOsTZXQwrQAOg2bEMhZ5EdlTDaDK37us2ZbhH3BN
uOAW3piHxlYPntl1n1DYvjpqZE+VBswMMUqo0sMDj4kTgNNcT1IaO43vwndwTOcT
WJT5SUVOMcsb61XjrQIDAQAB
----END PUBLIC KEY----
                                                                                                                                                                                                  A216b
SHA1 digest translate:96252f20795cef8edc7765fd847206b69e
SHA1 digest translate:75aa463b709a48c596a85c03d576e26f8f
                                                                                                                                                                                                  answer of verify:1
close client[ id:0 ip:127.0.0.1 socket:4 ]
exit thread: 139890472412928
close client[ id:2 ip:127.0.0.1 socket:4 ]
exit thread: 139890455627520
Send to B:start send:
                                                                                                                                                                                                        oot@ceph5-ubuntu: /home/xin777/cyber/socketTest/Sim
                                                                                                                                                                                                        root@ceph5-ubuntu:/home/xin777/cyber/socketTest/Simple
                                                                                                                                                                                                        server serverB
root@ceph5-ubuntu:/home/xin777/cyber/socketTest/Simple
                                                                                                                                                                                                        serverB 5688
accept client[ id:0 ip:127.0.0.1 handle:4 ]
 receive decode:Server B verify success. your messages are:
                                                                                                                                                                                                       Accepted
jack
A216b
                                                                                                                                                                                                       Accepted open client[ id:0 ip:127.0.0.1 socket:4 send:0 ] receive message:
m4dXyCuQwOu8sutztLQSQ8liEyhnI5/gKP8Ze1xcckF9sXyqaqwUIX-
Epxez8dwSTVR7keTWKRb0410]Rc1ZKDHhQ3vncYURsq8UHG1KDaat7(
Q6mNM84X1/c=:96252f20795cef8edc7765fd847206b69e409c34:
.
CjtWB+tpYzJyOKYNG0rxn+0Qq7LstqWVwBNwHf62s7C0AD+LYDT19c
dEO5yzVTp2tfZhOeTDouvULrmiv+j22oIOltuifopW8xvmru/V6Pru
  7Tl6aRM=
end to C:start send:
                                                                                                                                                                                                        BLIC KEY
                                                                                                                                                                                                       MIGFMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQCq8RcYTHXDuvUYrS
83jbqloLDwfFnf9uM9GFEkxgcOsTZXQwrQAOg2bEMhZ5EdlTDaDK37
 send to Listart send:
gsCVtr7x6012quSAwPU6MkYubp4yBK7R2nKpLVhcxkeivdKlB4dB4veyQDtcM2P62IKd41gAtN7sqLlr2E0P0RHSeXrCTyn
ZvrnrFJd/bygRgBtGso23h4sKSpMtk6In1h0nGcBf16LqktanoXbUE33gbn7zUltkaydo7Tl6aRM=:96252f20795cef8ed
c7765fd847206b69e409c34:YZuqXSRtkJfRe/vG18DXFY0B7XCjtWB+tpYzJy0KYNG0rxn+0Qq7LstqWVwBNwHf62s7C0A
D+LYDT19c67RyL-sqkRowdNe9b0/1JyS9oZwdE0SyzVTp2tfZh0eTDouvULrmiv+j22oIOltuifopW8xvmru/V6PrudYKs8B
PLdP8=:----BEGIN PUBLIC KEY----
                                                                                                                                                                                                       UDAW3piHxlYPntiin1DYvjpqZE+VBswMMUqo0sMDj4kTgNNcT1IaO4
WJr55uVOmcsb6iXjrQIDAQAB
-----END PUBLIC KEY-----
                                                                                                                                                                                                       MI Dāše64 =======:m4dXyCuQwOu8sutztLQ5Q8liEyhnI5/gKF
VOxi7Xmhe6Wf5s36kXgpFEpxez8dWsTVR7keTWKRb0410jRc1ZKDHh(
FP7BM9cs4A+N9dajWp6XZQ6mNM84X1/c=
messages[0]:125324
SHA1 digest trace
PLOPB=:----BEGIN PUBLIC KEY----
MIGFMA0GCSQGSIb3DQEBAQUAA4GNADCBiQKBgQCQ8RcYTHXDuvUYr5fiV7XvzVsS
83jbqloLDwfFnf9uM9GFEkxgcOsTZXQwrQA0g2bEMhZ5EdlTDaDK37us2ZbhH3BN
uOAW3piHxlYPnti1n1DYvjpqZE+VBswMMUqo0sMDj4kTgNNcT1IaO43vwndwTOcT
WJr5SuVOmcsb6iXjrQIDAQAB
                                                                                                                                                                                                       SHA1 digest translate:284bdd1a36dc87765f04f4e6be972791
SHA1 digest translate:75aa463b709a48c596a85c03d576e26f
     --- END PUBLIC KEY--
                                                                                                                                                                                                        verify success
  eceive messages:Server C verify success. your messages are:
   25524
he virtual number is 12341234
hot&renh5-ubuntu:/home/xin777/cvber/socketTest/SimpleNetwork/example
                                                                                                                                                                                                        answer of verify:1
close client[ id:0 ip:127.0.0.1 socket:4 ]
```

其中左边为客户发送的控制台输出(可以看到发送的完整流程,包括签名的值,加密的值,以及 base64 编码的值),右上为外卖平台的控制台输出,而右下为运营商的控制输出。最终在互相验证成功后,客户收到了返回的虚拟号码以及订单信息。

## 4. 安全分析

在发送的消息中,只有公钥以及 SHA1 哈希值是明文传输的,对于这两个信息还有 Sign(Hm12)签名来说,它们本身就是不需要加密的,可公开的信息量,对签名数据进行 Base64 是为了方便拼接处理与传输。而对于 M1,采用了 RSA 私钥加密。

在不考虑中间人攻击的情况下,对于这整个信息来说,是拥有机密性,完整性,认证性与不可否认性的。

其实理论上采用了 X509 证书认证的方式,是可以防止中间人攻击的手段的,功能主要是在客户收到服务器的证书后,进行验证,确认是可信任的根证书颁发即可。但是由于时间等原因,这一功能并没有选择添加。后续有待完善。

### 5. 遇到问题

#### 5.1 RSA 签名与验签

首先,密码学中,我们没有选择先签名,再哈希的原因是,攻击者可以直接篡改信息还有哈希值。而在实践实现中,我们经常会忘记哈希,选择直接对信息签名,而其实现有的 RSA 签名都是对哈希做签名的,一方面节省时间,一方面避免了特殊字符或者二进制无法识别且运算,这个小细节不注意有时会引起接收方验签错误。主要表现在 Hm1 与 Hm2 拼接后,需要再次哈希才能正确得到 Hm12,如果直接拼接进行签名会导致验签失败。

验证签名接口的参数,需要接收哈希方式,签名内容的哈希值,签名的长度,十六进制的签名,公钥信息。这里我在验签时总是失败,查找官网接口说明后,根据错误代码返回信息得知:1,签名本身为二进制,所以其长度不能够使用 strlen 等函数获取,比如 SHA256 长度很有可能为 128,具体长度需要在签名时就予以标注。2,验证签名与该签名的哈希值哈希方式密切相关,我使用了很多的哈希方式,都无法验证通过,最后去官网采用了官网提供的 OpenSSL 的 SHA1 哈希方式,终于通过。

### 5.2 Base64 解码与原信息不一致

在签名之后,会对签名值进行 Base64 编码,之后进行传输。原因很简单,签名之后的值是二进制特殊字符,无法进行消息拼接,也不是特别适合传输,所以进行 Base64 编码处理。但是问题

就在于接收方 Base64 解码之后,与原来的签名值不一致,导致的验证签名一直未通过。





原 base64 编码值

错误解码后再次进行编码值

而且主要表现在其余数据都相同,只有尾值不同,原签名数据编码后以 = 结尾,错误解码后再进行编码在会将 = 变为 A。

原始字符	н						e								1								1							o							!								
ASCII码十进制值	72									101							108								108										13	11							33		
二进制值	0	1	0	0	1	0	0	0	0	1	1	0	0	1	0	1	0	1	1	0	1	1 6	9	9 6	) 1	1	0	1	1	0	0	0	1	1	0	1	1	1	1	0	0	1	0	0	0
Base64码十进制值	18							6					21					44					27						6				60								33				
Base64编码后字符	ase64编码后字符 S							G					V						s					b							(	G				8						h			
原始字符					!																																								
ASCII码十进制值				3	3																																								
二进制值	0	0	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0 (	9 6	9	)																					
Base64码十进制值	8							16				0			)			0					1																						
Base64编码后字符		I						Q					A						Α																										

在深入了解 Base64 编码之后,得知该错误是由于编码时,结 尾的两个字符 2\*8 = 16bit 数据,需要在填充后转化为 4\*6 = 24bit 的数据,

10111101 10010000 to 101111 | 01 1001 | 0000 00 |

\*\*\*\*\*

星号会转化为 = , 而 000000 会转化为 A

而这个时候, 解码就会出现问题

101111 | 01 1001 | 0000 00 | \*\*\*\*\* to 10111101 |

10010000 | 00 \*\*\*\*\* |

可以看到在前两个字符解码还是正确的,而最后一个字符转化时,00\*\*\*\*\*是按照000000 来转化呢?还是按照\*\*\*\*\*转化呢?

网上的大多数代码都忽略了这样一个问题,将其按照 000000 转化,导致多出一个字符,从而致使解码出现 bug (也包括一些企业的平台也出现了这个漏洞)。

最后将填充尾值丢弃, 修改完代码后, 解码成功。

# 实验细节

#### 1. 应用说明:

本应用使用了 C++语言来进行实现。实验环境是 Ubuntu16。使用的加解密接口为 OpenSSL 开放的底层 RSA 加解密和底层哈希接口。

运行 client 时,需要输入对应的参数,在默认 help 中设置了格式提示。另外,因为没有采用主动申请证书的机制,所以需要提前手动在 client 中和两个 server 生成各自的证书,并且提取出 prikey.pem 私钥文件。但是本应用有证书传递功能,因此不需要将 server 的证书提前内置在 client 中。(代码注意截图,截图中有运行提示)

## 2. 证书相关操作

### 2.1 创建根证私钥

openssl genrsa -out root-key.key 1024

### 2.2 创建根证书请求文件

openssl req -new -out root-req.csr -key root-key.key -keyform PEM

## 2.3 自签根证书

openssl x509 -req -extfile /etc/ssl/openssl.cnf -extensions v3\_req -in root-req.csr -out root-cert.cer -signkey root-key.key -CAcreateserial

-days 3650

其中/etc/ssl/openssl.cnf 是系统自带的 openssl 配置文件, 若没找到, 自行 find 命令寻找配置文件路径

#### 2.4 使用根证书签发客户端证书

#### 2.4.1 生成客户端 key

openssl genrsa -out client-key.key 1024

#### 2.4.2 生成客户端请求文件

openssl req -new -out client-req.csr -key client-key.key

#### 2.5 生成客户端证书, 使用根证书签名

root@ceph5-ubuntu:/home/xin777/cyber/client/secure# openssl x509 -req -extfile /etc/ssl/openssl.cnf -extensions v3\_req -in client-req.csr -out client-cert.cer -signkey client-key.key -CA ../../secure/rootsite/root-cert.cer -CAkey ../../secure/rootsite/root-key.key -CAcreateserial -days 3650

# 2.6 查看证书

openssl x509 -in client-cert.cer -text -noout

# 2.7 查看证书的公钥

openssl rsa -in prikey.pem -pubout -out pubkey.pem

### 3. 完整实验演示

首先需要运行两个服务端(运营商和外卖商),然后运行客户端,采取命令行输入命令。其中 client 程序需要接收指定的几个参数,分别是 IP,端口,-n name,-t tel,-a address。之后可以看到完整的通信流程(我将主要的编解码或加解密的值输出在了控制台上),之后也可以在对应的服务端看到认证成功与否的消息,如下:

./client -i 127.0.0.1 -p 5687 -n jack -t 125324 -a "A building,216b"

```
root@ceph5-ubuntu:/home/xin777/cyber/socketTest/SimpleNetwork/example-client# .
client -i 127.0.0.1 -p 5687 -n jack -t 125324 -a "A216b"
name:jack
tel:125324
addressA216b
start send:
pubkey1.pem
receive file finish: pubkey1.pem
start send:
pubkey2.pem
Encrypted M1 Base64 encode: HiOyqX9H33hELroeFp7DAInjht3oiK3KNZV7y3pos0XQ0V48z/mYv
XQ//+1x9Rjlmyd5+pA1/NDYQzT8j/lWPJiBmIyggTqH923cZuU2WPRbcw2fZax7OTtwvBJ59xUhGtHXFqiIr3kfyN8kOcwDZWVJYw7KtK58c8L28qMpLp8=
Hm1 SHA1 digest translate:96252f20795cef8edc7765fd847206b69e409c34
Hm2 SHA1 digest translate:284bdd1a36dc87765f04f4e6be9727913e446149
Hm12 SHA1 digest translate:75aa463b709a48c596a85c03d576e26f8f716487
critical sha length:
128
verify success
local pubkey transformed to string ======:
----BEGIN PUBLIC KEY-----
MIGFMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQCq8RcYTHXDuvUYr5fiV7XvzVsS
83jbqloLDwfFnf9uM9GFEkxgcOsTZXQwrQAOg2bEMhZ5EdlTDaDK37us2ZbhH3BN
uOAW3piHxlYPnti1n1DYvjpqZE+VBswMMUqo0sMDj4kTgNNcT1IaO43vwndwTOcT
WJr55uVOmcsb6iXjrQIDAQAB
 ----END PUBLIC KEY---
start send:
.HiOyqX9H33hELroeFp7DAInjht3oiK3KNZV7y3pos0XQOV48z/mYwXQ//+1x9Rjlmyd5+pA1/NDYQzT8
j/lWPJiBmIyggTqH923cZuU2WPRbcw2fZax7OTtwvBJ59xUhGtHXRqiIr3kfyN8kOcwDZWVJYw7KtK58
c8L28qMpLp8=:284bdd1a36dc87765f04f4e6be9727913e446149:YZuQXSRikJfRe/vGI8DXFY0B7
.CjtWB+tpYzJy0KYNG0rxn+00q7LstqWVwBNwHf62s7C0AD+LYDTI9c67RyLs4pK0wdNe9bo/1lyS9oZi
dEO5yzVTp2tfZhOeTDouvULrmiv+j22oIOltuifopW8xvmru/V6PrudYKs8BPLdP8=:----BEGIN PU
MIGFMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQCq8RcYTHXDuvUYr5fiV7XvzVsS
83jbqloLDwfFnf9uM9GFEkxqcOsTZXOwrQAOq2bEMhZ5EdlTDaDK37us2ZbhH3BN
uOAW3piHxlYPnti1n1DYvjpqZE+VBswMMUqoOsMDj4kTgNNcT1IaO43vwndwTOcT
WJr55uVOmcsb6iXjrQIDAQAB
----END PUBLIC KEY---
receive decode:Server B verify success. your messages are:
jack
A216h
please wait, your order is being delievered
```