my secure communication system

2022年6月17日 19:45

目录：

1. 实现的密码算法

RSA算法：

RSA（Rivest-Shamir-Adleman）是一种广泛用于安全数据传输的公钥密码系统。RSA 用户根据两个大质数以及一个辅助值创建并发布一个公钥。素数是保密的。任何人都可以通过公钥加密消息，但只能由知道素数的人解码。

RSA 的安全性依赖于分解两个大素数乘积的实际困难，即“大整数分解问题”。如果使用足够大的密钥，目前没有公开的方法来攻破这个系统。

RSA 是一种相对较慢的算法。正因为如此，在本系统设计中，为了提高信息传递的效率，它并不用于直接加密用户数据，而是 用于传输对称密钥加密（在本系统中是AES）的共享密钥，然后用于批量加密-解密。

代码：

素性检验：

```
    # 素性检验
    def is_prime(num):
        if num == 2:
            return True
        if num < 2 or num % 2 == 0:
            return False
        for n in range(3, int(num**0.5)+2, 2):
            if num % n == 0:
                return False
        return True
```

生成随机16位素数：

```
def generateRandomPrim(self):
    while(1):
        ranPrime = random.randint(0,65536)
        if self.is_prime(ranPrime):
            return ranPrime
```

生成密钥：

```
def generate_keyPairs(self):
    self.p = self.generateRandomPrim()
    self.q = self.generateRandomPrim()

    self.n = self.p*self.q
    # print("n ",n)

    self.phi = (self.p-1) * (self.q-1)
    # print("phi ",phi)


    self.e = random.randint(1, self.phi)
    g = self.gcd(self.e,self.phi)
    while g != 1:
        self.e = random.randint(1, self.phi)
        g = self.gcd(self.e, self.phi)

    # print("e=",e," ","phi=",phi)

    self.d = self.egcd(self.e, self.phi)[1]

    # 保证d是正的
    self.d = self.d % self.phi
    if(d < 0):
        d += self.phi

    return ((self.e,self.n),(d,self.n))
```

加密与解密：

```python
def decrypt(self,private_key):
    try:
        key,n = private_key
        self.text = [chr(pow(char,key,n)) for char in self.ctext]
        return "".join(self.text)
    except TypeError as e:
        print(e)

def encrypt(self,public_key):
    key,n = public_key
    self.ctext = [pow(ord(char),key,n) for char in self.text]
    return self.ctext
```

实例：

```
Public:  (1313949701, 2927704867)
Private:  (1904122445, 2927704867)
encrypted  = [1877324118, 636907642, 2307856049, 2307856049, 1592848526, 379725380, 2464659111, 1592848526, 96614984, 2307856049, 227477280
8]
decrypted = Hello World
```

AES算法：

AES的全称是advanced encryption standard，意为高级加密标准。它的出现主要是为了取代在不断发展的科技水平下不再安全且高效的DES加密算法。1997年1月2日，美国国家标准技术研究所宣布希望征集高级加密标准，用以取代DES。有5个候选算法进入最后一轮：Rijndael，Serpent，Twofish，RC6和MARS。最终经过安全性分析、软硬件性能评估等严格的步骤，Rijndael算法获胜。

在密码标准征集中，所有AES候选提交方案都必须满足以下标准：

分组大小为128位的分组密码。

必须支持三种密码标准：128位、192位和256位。

AES算法是Rijndael的特别情况。相比之下，Rijndael本身的块和密钥大小可以是 32 位的任意倍数，最小为 128 位，最大为 256 位。

故虽然本系统要求16位的安全性，但本系统中实现的AES算法仍采用128位密钥。

128位密钥的AES需要经过10轮转换。每一轮都包含几个处理步骤，其中一个取决于加密密钥本身。一组反向循环用于使用相同的加密密钥将密文转换回原始明文。

算法的具体描述：

KeyExpansion – 轮密钥是使用AES 密钥调度从密码密钥派生的。AES 需要一个单独的 128 位轮密钥块，每轮添加。

初始轮密钥添加：

AddRoundKey – 状态的每个字节都使用按位异 或与轮密钥的一个字节组合。

第9 轮：

SubBytes – 一个非线性替换步骤，其中每个字节根据查找表替换为另一个。

ShiftRows – 一个转置步骤，其中状态的最后三行循环移动一定数量的步骤。

MixColumns – 一种线性混合操作，对状态的列进行操作，将每列中的四个字节组合在一起。

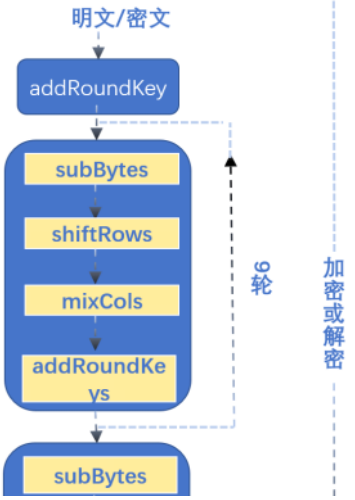
AddRoundKey

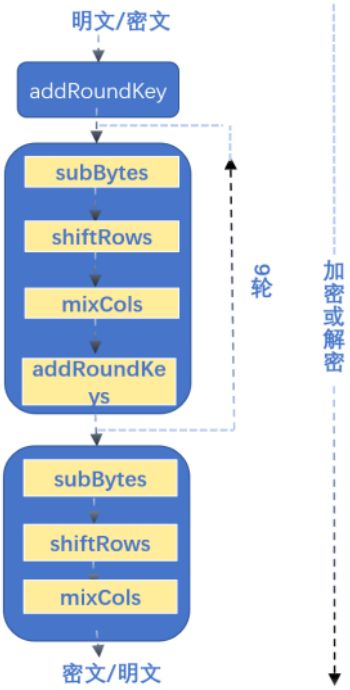
最后一轮：

SubBytes

ShiftRows

AddRoundKey





hash function：

由于系统安全性要求为16位，所以考虑采用平方取中法构造哈希函数。即将原消息截取后16位后平方取中间16位作为消息摘要。

代码：

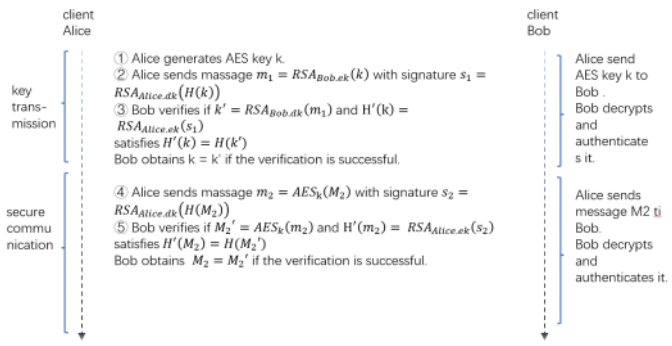
```
class hash:
    message = ""
    abstract = 0

    def makehash(self):
        binary_converted = int(''.join(format(ord(c), 'b') for c in self.message),2)
        binary_converted = binary_converted % 65536
        binary_converted = binary_converted*binary_converted
        binary_converted = binary_converted % 16777216 #2^24 = 16777216
        self.abstract = binary_converted // 256
        print(bin(self.abstract))
        print(self.abstract)
```

实例：

```
message: here is an example
abstract:
0b1000111011000011
36547
```

3. 系统的工作流程

如果Alice想要和Bob交流，他们会经过这样的工作流程：



key transmission process：

Alice 首先生成128位AES密钥k，然后用Bob的RSA公钥将k加密得到消息息m1，同时附上经过Alice的RSA私钥加密的k的哈希值s1作为签名。

Bob接收到m1和s1后，首先对消息完整性和可用性进行验证。如果通过Bob的私钥解密m1得到的k'经过哈希函数形成的摘要H(k')和通过Alice的公钥解密s1得到的H'(k)相同，则认为消息得到了验证，Bob成功得到对称加密所需的密钥k。

secure communication process：
Alice和Bob具有相同的对称密钥k，如果Alice要传输消息M2给Bob：
首先将M2用AES加密，然后将加密后的消息与Alice的签名一起传输给Bob
Bob验证签名后确认消息的完整性的可用性，再用AES的密钥将消息解密后得到原文。

3. 系统的安全性
根据木桶原理，整个通信系统的安全性取决于最易被攻破的一环。
假设该通信系统传递的明文均为随机字符串，攻击者只能截取或改变传输中的字符串，不能进行尝试攻击。不考虑字母频率分析、边信道攻击等非算法攻击方式。

通信系统主要包括3个部分：用于对称密钥传递和数字签名的rsa算法，用于生成消息摘要作为数字签名的hash函数，用于明文密文传递的AES算法。
其中rsa算法的p/q均为随机生成的16位质数，由于不存在多项式时间的大整数分解算法，系统此部分安全性可以达到16位（即理论上需要2^16次尝试可以破解）
hash function部分：若字符串长度远大于16位且随机，则通过基于平方取中法的哈希函数理论上需要2^16次尝试才能构造碰撞，安全性可以达到16位。
AES：密钥长度为128位的128位分组对称密码算法。在密钥不泄露的前提下，安全性远高于16位。
故综合分析，该系统的安全性满足16位。

4. 运行实例
main.py:

```python
if __name__ == "__main__":
    alice = client()
    bob = client()

    # transmit key
    m1 = bob.r.encrypt(alice.aeskey,bob.pub)
    s1 = alice.r.encrypt(alice.h.makehash(alice.aeskey),alice.pri)

    tmp1 = bob.h.makehash(bob.r.decrypt(m1,bob.pri))
    tmp2 = alice.r.decrypt(s1,alice.pub)

    if(tmp1 == tmp2):
        print("successfully transmitted aes key")
        bob.aeskey = bob.r.decrypt(m1,bob.pri)
        print("aes key: ",bob.aeskey)


    # secure communication
    M = "this is an example"
    alice.a.plainArray = M
    m2 = alice.a.encrypt()
    s2 = alice.r.encrypt(alice.h.makehash(alice.a.plainArray),alice.pri)

    tmp1 = bob.h.makehash(bob.a.decrypt(m2))
    tmp2 = alice.r.decrypt(s2,alice.pub)

    if(tmp1==tmp2):
        print("successfully commnunicated")
        bob.a.plainArray = bob.a.decrypt(m2)
        print("message: ",bob.a.plainArray)
```

```
successfully transmitted aes key
aes key:  0xed0919fa
successfully commnunicated
message:  this is an example
```