RSA Blinding RUS

7 февраля 2020 г.

1 Атака RSA Blinding

1.1 Введение

RSA (названа в честь своих создателей Ronald Linn Rivest, Adi Shamir and Leonard Adleman) это пример асимметричной криптосистемы, которая может быть использована для безопасной передачи данных и создания подписей. рассмотрим базовые принципы работы криптосистемы.

RSA использует мультипликативную группу по модулю $N = pq$, где p и q - простые числа. Степень мультипликативной группы (по-сути, её мощность) может быть вычислена при помощи функции Эйлера для составного числа из двух простых: $\varphi(N) = (p-1)(q-1)$. Функция считает количество натуральных чисел меньше N, которые не кратны p или q. Поскольку все такие числа взаимно просты с N, они состоят в мультипликативной группу. Как мы знаем, если возвести любой элемент конечной мультипликативной группы в степень этой группы, то получим нейтральный элемент (единицу): $a^{\varphi(N)} = 1, a \in Z_N^*$. Поэтому в RSA используют два числа e (открытая экспонента) и d (закрытая экспонента), такие что $ed = 1 mod N$. Пара чисел (e, N) используется как открытый ключ, а (d, N) как закрытый. Вычисление d из открытого ключа является сверхполиномиальной задачей (NP), если не были сгенерированы слабые N, d или e. Одним из способов решения является факторизация N в произведение p и q.

Пусть дан открытый текст (число) $M, M < N$, открытый ключ (e, N) и закрытый ключ (d, N), шифрование и расшифрование осуществляются следующим образом:

Шифрование $C = M^e mod N$

Расшифрование

$M = C^d mod N$

Проверка корректности:

$C^d mod N = M^{ed} mod N = M^{edmod\varphi(N)} mod N = M^1 mod N = M mod N$

1.2 Preparation

Попробуем немного поработать с RSA. Если ещё не установили, установите Pycryptodome. На Linux и Windows должна сработать следующая команда (Предварительно надо установить python 3 и pip, но я надеюсь, что вы справились с этим самостоятельно):

[1]:
```
!python3 -m pip install pycryptodome
```

```
Collecting pycryptodome
  Using cached https://files.pythonhosted.org/packages/54/e4/72132c31a4cedc58848
615502c06cedcce1e1ff703b4c506a7171f005a75/pycryptodome-3.9.6-cp36-cp36m-manylinu
x1_x86_64.whl
Installing collected packages: pycryptodome
Successfully installed pycryptodome-3.9.6
```

После установки надо перезапустить ядро jupyter (круговая стрелка рядом с "Run"). Если возникнут проблемы, загляните в документацию: Pycryptodome installation.

1.3 Примитивная реализация RSA

Давайте сделаем простейшую версию RSA. Будем использовать открытую экспоненту $e = 65537$. Обычно используют эту константу, потому что она переаолняет модуль даже при открытом тексте $M = 2$ и у нее удобное двоичное представление $65537_{10} = 10000000000000001_2$, которое позволяет эффективно возводить число в степень, используя алгоритм "Square and multiply".

Сначала сгенерируем p и q. Функция getStrongPrime дает возможность выбрать количество бит в генерируемом простом числе и проверяет, что $(p-1, e) = 1$

[2]:
```
try:
    from Crypto.Util.number import getStrongPrime, inverse,bytes_to_long,␣
 ↪long_to_bytes
except ImportError:
    print ("Pycryptodome not installed")
```

[3]:
```
e=65537
p=getStrongPrime(1024,e=e)
q=getStrongPrime(1024,e=e)
```

[4]:
```
N=p*q
phi=(p-1)*(q-1)
d=inverse(e,phi)
public_key=(e,N)
private_key=(d,N)
```

Мы успешно сгенерировали ключи, теперь давайте зашифруем сообщение, расшифруем закрытый текст и проверим, что получили то же самое

[5]:
```
M=bytes_to_long(b'Hello, RSA!')
C=pow(M,e,N)
print ('C:',hex(C))
M1=pow(C,d,N)
assert M1==M
print ('M1:',long_to_bytes(M1))
```

```
C: 0x98f3405c8d1afb28d02379707a07779cfcc727bea6b7b0f0eb6968100ad29489940d4f69dbd
```

```
833a550f5c298972033ab7c321cb9e0744961683625d2717b8c426638cd2ec4293ef8b45e7cf675e
c8046149863ae935edb37054f44ed07a24539d4e09bec8eda18264895cd48c28a4ce077c0c24e2f4
95a367c790242ac1f6ccc50ec1984e71fd2fd103d66a7a2239e1dd06f3d5f899c20418a99d3650c9
391e7debf558351b958d0f00087195e3b2ef5c87c1f021480edfd9fc3fd00efd383ebac21a332021
533365be4d65684d6da7c54102da77c2ed4ed43aa8dbf9243621ece4f7f946323c146717b3d61dd6
5d0245c019c96b817010af8926f8aa3bf2e9c
M1: b'Hello, RSA!'
```

Создание подписи - обратная операция к шифрованию.

$$Sign(M) \equiv Dec(M), Check(S) \equiv Enc(S)$$

Таким образом любой, владеющий окрытым ключом, может проверить правильность подписи, а создать её может только сторона, у которой есть закрытый ключ. Поздравляю, теперь вы знаете, как шифровать и создавать подаписи при помощи RSA. Дальше рассмотрим одно из его интересных свойств.

1.4 RSA Blinding

RSA - это гомоморфное шифрование по отношению к операции умножения. Отношение является гомоморфизмом групп, если оно операцию одной группы переводит в опрацию другой группы ,операция и отношение элементов сохраняются. Если ничего не понятно, не беспокойтесь, я в первый раз, когда услышал, тоже ничего не понял. Что это значит на практике: пусть у вас есть два элемента группы G_1 (x, y) и вы применяете к ним гомоморфное отображение, они будут также связаны в новой группе G_2 (для RSA $G_1 = G_2$):

$$\varphi(x \cdot y) = \varphi(x) \times \varphi(y)$$

Для шифрования RSA:

$$Enc(M_1 \cdot M_2) = Enc(M_1) \times Enc(M_2) =$$

То же самое верно и для расшифрования:

$$Dec(C_1 \times C_2) = Dec(C_1) \cdot Dec(C_2)$$

Протестируем это свойство в python

[6]:
```python
class BasicRSA:
    def __init__(self, e,p,q):
        self.e=e
        self.p=p
        self.q=q
        self.N=p*q
        self.d=inverse(e,(p-1)*(q-1))

    def encryptNumber(self, m):
        return pow(m,self.e,self.N)

    def decryptNumber(self, c):
```

```
        return pow(c,self.d,self.N)

brsa=BasicRSA(e,p,q) #we created these parameters earlier
m1=2
m2=3
m3=m1*m2
c1=brsa.encryptNumber(m1)
c2=brsa.encryptNumber(m2)
print ('c1:',c1)
print ('c2:',c2)
c3=(c1*c2)%brsa.N
print('c3:',c3)
m3_dec=brsa.decryptNumber(c3)
print ('m3: %d, m3_dec: %d'%(m3,m3_dec))
assert m3_dec==m3
```

```
c1: 291778566264659889340067455651670375907165315768260678767010820082786377596536245310495922376821729112241097566044243314213114199248248618959956992032462919599718438285775304993919377278494552804663762740231802808110515664205585760900463999730713331550298743053939959338509067152779968561573645528781257355275761088259188708074880376213161888522413138339359729267418233144634478534506833079123369674038794425932197462162433424140909219556825284773362427014316576789857141643778868180502070621654051172327516657943656154420939828554765371153608188804664466657448115273050774558873869247289738338218566352074589324007646997002195329104740994463085131943782865476543540076933169448795171342031663903351603548956452664746326786229220537420972142343544311593325960815479390728514523375168836001256493954554992623899831438449860304545495690875266931303861537120379584454826740958490686564249031344818856209182785532042922377360302026194891709805859689116826624645527453785374641551212275411542387998661613871211165758081018419379877840156544751827114505281443224757302231619528819233602520309733745596830514777873357751582665826759698959828150978586440541750382198351900138726101940403214915906816362607295310798396042849519999748110705043c3: 102182635181994901793292654873459886071678541517457243369864102619913881231807155172334720742043329684009335325838881861120410816121733128168915687720675958572900029913908278876174028620743543865200977984439176391837938804901588570453789840636930991118360829362809701754916810714058142095317704820888512951490370769524631532472254566587440069169743402218718882227836888276597172352720848804246680982965254445261599515288677670733497783690492206760227435257605384675117902469951695025831031604596874821114145852147865212879251989509396296733804426752999701843523353582258367425946571243214674822853247051456409914599053c3: 6, m3_dec: 6
```

1.5 Атакуем сервер

Теперь попробуйте применить эти знания к уязвимому серверу. Вы можете приконнетиться, используя `nc cryptotraining.zone 1337` или при помощи питоновских сокетов.

[7]:
```
import socket
import re
class VulnServerClient:
    def __init__(self,show=True):
        """Ининциализация, подключаемся к серверу"""
        self.s=socket.socket(socket.AF_INET,socket.SOCK_STREAM)
        self.s.connect(('cryptotraining.zone',1337))
        if show:
            print (self.recv_until().decode())
    def recv_until(self,symb=b'\n>'):
        """Получение сообщения с сервера, по умолчанию до приглашения к вводу␣
 ↪команды"""
        data=b''
        while True:

            data+=self.s.recv(1)
            if data[-len(symb):]==symb:
                break
        return data
    def get_public_key(self,show=True):
        """Получение открытого ключа с сервера"""
        self.s.sendall('public\n'.encode())
        response=self.recv_until().decode()
        if show:
            print (response)
        e=int(re.search('(?<=e: )\d+',response).group(0))
        N=int(re.search('(?<=N: )\d+',response).group(0))
        self.num_len=len(long_to_bytes(N))
        return (e,N)

    def signBytes(self,m,show=True):
        """Получение подписи для выбранного сообщения в байтах с сервера"""
        try:
            num_len=self.num_len
        except KeyError:
            print ('You need to get the public key from the server first')
            return
        if len(m)>num_len:
            print ("The message is too long")
            return
        if len(m)<num_len:
            m=bytes((num_len-len(m))*[0x0]) +m
        hex_m=m.hex().encode()
        self.s.sendall(b'sign '+hex_m+b'\n')
        response=self.recv_until().decode()
        if show:
            print (response)
```

```
        if response.find('flag')!=-1:
            print('You tried to submit \'flag\'')
            return None
        signature_hex=re.search('(?<=Signature: )[0-9a-f]+',response).group(0)
        signature_bytes=bytes.fromhex(signature_hex)
        return bytes_to_long(signature_bytes)


    def signNumber(self,m,show=True):
        """Получение подписи с сервера для выбранного сообщения в числовом␣
↪представлении"""
        try:
            num_len=self.num_len
        except KeyError:
            print ('You need to get the public key from the server first')
            return
        return self.signBytes(long_to_bytes(m,num_len),show)

    def checkSignatureNumber(self,c,show=True):
        """Проверка сигнатуры (на сервере) для подписи в числовом␣
↪представлении"""
        try:
            num_len=self.num_len
        except KeyError:
            print ('You need to get the public key from the server first')
            return
        signature_bytes=long_to_bytes(c,num_len)
        self.checkSignatureBytes(signature_bytes,show)

    def checkSignatureBytes(self,c,show=True):
        """Проверка сигнатуры (на сервере) для подписи в байтовом␣
↪представлении"""
        try:
            num_len=self.num_len
        except KeyError:
            print ('You need to get the public key from the server first')
            return
        if len(c)>num_len:
            print ("The message is too long")
            return

        hex_c=c.hex().encode()
        self.s.sendall(b'flag '+hex_c+b'\n',)
        response=self.recv_until(b'\n').decode()

        if show:
            print (response)
```

```
        if response.find('Wrong')!=-1:
            print('Wrong signature')
            x=self.recv_until()
            if show:
                print (x)
            return
        flag=re.search('CRYPTOTRAINING\{.*\}',response).group(0)
        print ('FLAG: ',flag)

    def __del__(self):
        self.s.close()
```

[8]:
```
vs=VulnServerClient()
(e,N)=vs.get_public_key()
```

```
Welcome to RSA blinding task
Available commands:
help - print this help
public - show public key
sign <hex(data)> - sign data
flag <hex(signature(b'flag'))> - print flag
quit - quit
>
e: 65537
N: 20159717663186764200842482638329142432479376755681286432561400011207751568770
23937873504239055098886463647821209788938254180637863281345152201173477839435246
47506954302364591564396569321085369361070927857591871209155591733213020275252290
18106368725032056109022369913503577023942696069608771010384365856481001383579432
84411223121576763032862701509742254008778946240450869708632121399086803127321961
48979014368449994422593874530212706423955318848486976509334781242540719122324457
08062597679170291021925633789812405697682134528381868778865376836541179591638312
15247213631375725238476129368433608284013777398457594745906 1
>
```

Вы можете подписывать сообщения при помощи методов signNumber (подписать число) и signBytes (подписать сообщение из байтов)

Проверять подпись можете при помощи методов checkSignatureNumber и checkSignatureBytes.

Ваша цель - получить правильную подпись для сообщения 'flag'.

Помните, что RSA - это гомоморфизм и решите задание.

Удачи!

[ ]: