
RSAextreme

RSA_Extreme

Headorteil



Table des matières

Le sujet	3
Analyse du problème	4
C'est parti	4
A l'attaque !!	5
L'implémentation	6
C'est déjà fini ;(7

Le sujet

On nous donne une commande : `ssh -i rsa_priv_key rsaextreme@157.159.40.163`, une clef publique a télécharger et un indice : « Tu connais les fractions continues ? »

Analyse du problème

On voit donc qu'on doit générer une clé privée à l'aide d'une clé publique. On a donc un N et un e et on cherche à obtenir un d . En effet, les RSA fonctionnent de la manière suivante : on génère deux nombres premiers p et q et on note leur produit N . On génère ensuite e et d tels que $ed \equiv 1[\varphi(N)]$ avec $\varphi(N) = (p-1)(q-1)$. La clé publique est alors le couple (N,e) et la clé privée (N,d) .

C'est parti

Tout d'abord on doit extraire notre N et notre e grâce à la commande :

```
openssl rsa -in rsa_pub_key -pubin -text -m
```

```
thomas@Tom: ~/Documents/tsp/Ctf/rsa3
thomas@Tom:~/Documents/tsp/Ctf/rsa3$ openssl rsa -in rsa_pub_key -pubin -text -m
Public-Key: (2048 bit)
Modulus:
 02:07:a7:df:9d:17:3f:59:69:ad:16:dc:31:84:96:
 b3:6b:e3:9f:e5:81:20:7e:6a:a3:18:d3:bf:be:22:
 c8:b4:05:68:0b:a9:81:1a:7b:de:cc:ed:5a:eb:79:
 a1:c2:c4:91:eb:6d:4f:39:82:06:57:b6:68:63:91:
 b8:54:74:17:2a:e5:04:f4:8f:02:f7:ee:3a:2a:b3:
 1f:ce:1c:f9:c2:2f:40:e9:19:96:5c:7f:67:a8:ac:
 bf:11:1e:e4:e7:e2:f3:21:7b:c9:a0:54:88:75:00:
 42:40:08:06:c0:e7:59:08:16:51:f6:e4:06:a9:a6:
 42:de:6e:8e:13:1c:b6:44:a1:2e:46:57:3b:d8:24:
 6d:c5:e0:67:d2:a4:f1:76:fe:f6:ee:c4:45:bf:a9:
 d8:88:0a:35:25:73:76:e6:71:09:fa:ab:e3:9b:0c:
 f8:af:e2:ca:12:3d:a8:31:4d:09:f2:40:49:22:fc:
 41:16:d6:82:a4:bd:ae:ec:b7:3f:59:c4:9d:b7:fa:
 12:a7:fc:5c:98:14:54:92:5c:94:e0:b5:47:2e:02:
 d9:24:da:d6:2c:2e:00:66:e0:7c:7d:3b:10:09:d0:
 47:5c:2c:06:6b:7f:94:55:3c:75:e8:56:e3:a2:a7:
 73:c6:c2:4d:5b:a6:40:55:eb:8f:ea:3e:57:b0:6b:
 04:a3
Exponent:
 00:f7:0b:3b:d7:48:01:a2:5e:cc:bd:e2:4e:01:b0:
 77:67:7e:29:83:91:d4:19:7b:09:9a:6f:96:12:44:
 f0:43:14:da:7d:e1:44:d0:69:a8:aa:84:68:6b:f4:
 d0:bd:14:a6:34:4b:bc:31:52:18:db:ba:f2:94:90:
 a4:4e:42:e5:c4:a2:a4:e7:6b:81:01:a5:ca:82:35:
 1c:07:b4:cf:d4:e0:80:38:c8:d5:57:3a:82:7b:22:
 7b:ce:51:5b:70:86:67:24:71:8e:c2:ac:03:35:96:
 14:cd:f4:3d:8b:f1:a:c7:ee:45:39:17:97:5a:12:
 c0:19:e6:20:a5:31:20:76:92:22:40:09:c7:5a:ae:
 f1:1e:13:0f:8e:54:cc:e3:1e:80:c8:4e:93:66:21:
 9a:e5:c2:58:85:3b:e1:45:ea:87:dc:f3:7a:47:ec:
 e0:a9:94:19:58:85:c2:1e:bc:db:fe:74:2d:f1:cd:
 13:70:c9:5b:66:84:ab:6c:37:e8:47:62:19:3c:27:
 dd:34:c3:cf:3f:5e:69:95:7b:83:38:f9:14:3a:00:
 52:c9:38:10:9e:2e:cb:9e:f5:04:c9:54:04:53:f5:
 76:32:79:5c:d4:4b:29:a4:b5:cb:a6:13:08:e4:85:
 da:6a:f2:df:c9:01:e4:58:68:cd:d5:00:69:13:f3:
 38:a3
Modulus=207a70f90173f5969a0180c3184968368e39f5e812076e8a31003bfe22c084856888a98
114780ecf605a8b79a1c2c491e6d4f3821865786686391805474172ae594f48f02f7ee3a2a831fc
E1CF9C22F40E919965C7F67A8ACBF11EE4E7E2F32178C9A054587500424D0806C0E759081651F6E
406A9A642DE6E8E131C644A12E46573B08246DC5E06702A4F176FEF6EEC4458FA908888A3525737
8E67109FA8E398CFAFE2CA12DA8314069F2464922FC41160682A40DAEECB3F59C49087FA12A
7F5C081454295C4E085472E020924D062C26866E07C7D3B1080D5475C2C06687F94553C75E85
6E3A2A773C6C24D58A64055E88FEA3E57806804A3
Writing RSA key
----BEGIN PUBLIC KEY-----
```

FIG. 1: Voilà le N et le e en hexadécimal

On obtient donc N : le « modulus » qu'on obtient en base 10 grâce à la commande :

```
echo 'ibase=16; <modulus>' | bc
```

On convertit ensuite le e : « exponent » en base 10 aussi grâce à un site dédié, les « : » et les retours à la ligne étant problématiques pour itérer la méthode précédente.

A l'attaque !!

On se retrouve donc a la recherche d'un d en ayant qu'un N et un e , c'est ici que l'indice intervient, on nous parle de fractions continues, on va donc implémenter une attaque de Wiener. Cette attaque consiste a approximer $\phi(N)$ par N , en effet on a $N = pq$ et $\varphi(N) = N - (p + q) + 1$, p et q étant très grands, on peut plus ou moins négliger $(p + q) + 1$ devant pq . Or on sait qu'il existe k tel que $ed = k\varphi(N) + 1$ puisque $ed \equiv 1[\varphi(N)]$.

On a donc $\frac{e}{\varphi(N)} - \frac{k}{d} = \frac{1}{d\varphi(N)}$: et donc $\frac{e}{N} - \frac{k}{d} = \frac{1}{dN}$ avec l'approximation qu'on s'est permise. $\frac{e}{N}$ et $\frac{k}{d}$ sont donc semblerait il très proches. On va donc tenter d'obtenir k et d en s'approchant de plus en plus précisément de $\frac{e}{N}$. Pour cela on va utiliser les fractions continues.

On utilise l'algorithme d'Euclide pour récupérer une suite de coefficients successifs afin de construire notre fraction continue.

Enfin on construit notre fraction et on essaye de construire une clé avec son dénominateur comme étant d . Si ça ne fonctionne pas, on doit ajouter un étage a notre fraction et retenter... Par exemple pour un $n = 53$ et $e = 17$

$e/n : e = 0 * n + 17$, la fraction est : 0 et on a alors $d = 0$, on essaye de générer une clef mais ça ne fonctionne pas, on itère ;

$n/17 : n = 3 * 17 + 2$, la fraction est : $0 + \frac{1}{3}$ et on a alors $d = 3$, on essaye de générer une clef mais ça ne fonctionne pas, on itère ;

$17/2 : 17 = 8 * 2 + 1$, la fraction est : $0 + \frac{1}{3 + \frac{1}{8}} = \frac{8}{25}$ et on a alors $d = 25$, on essaye de générer une clef mais ça ne fonctionne pas, on itère ; et cætera et cætera.

L'implémentation

On implémente le tout en python et le résultat est le suivant :

```
1  from Crypto.PublicKey import RSA
2  from fractions import Fraction
3  n = <le N>
4  e = <le e>
5  doc = open("/home/thomas/Documents/tsp/Ctf/rsaextreme/rsaextreme", "w")
6
7  def f(a, b, c, T):
8      if c == 0:
9          return(T)
10     temp = a%b
11     T.append(a/b)
12     return f(b, temp, c-1, T)
13
14  def frac(res, T):
15     if len(T) == 1:
16         return Fraction(1, res) + T[0]
17     return frac(Fraction(1, res) + T.pop(), T)
18
19  def final(a):
20     T = f(e, n, a, [])
21     for i in range(2, a):
22         N = T[:i]
23         d = (frac(N.pop(), N)).denominator
24         print(d)
25         try:
26             key = RSA.construct((n,e,d))
27             print("Youpi")
28             doc.write(key.exportKey())
29             return
30         except ValueError: None
31
32  final(500)
33
34  doc.close()
```

C'est déjà fini ;(

On obtient alors un magnifique « Youpi » sur le terminal après quelques minutes et une superbe clé privé dans notre fichier rsaextreme. Il ne reste plus qu'à l'envoyer au serveur avec la commande :

```
ssh -i rsaextreme rsaextreme@157.159.40.163
```

et l'afficher flag.txt avec la commande :

```
cat flag.txt
```

Et voila, on obtient le flag.