Cryptanalyse du RSA
Dans quelles limites le système de cryptographie RSA est-il sécurisé ?

Aloïs Simon
Candidat 22273

9 juin 2019

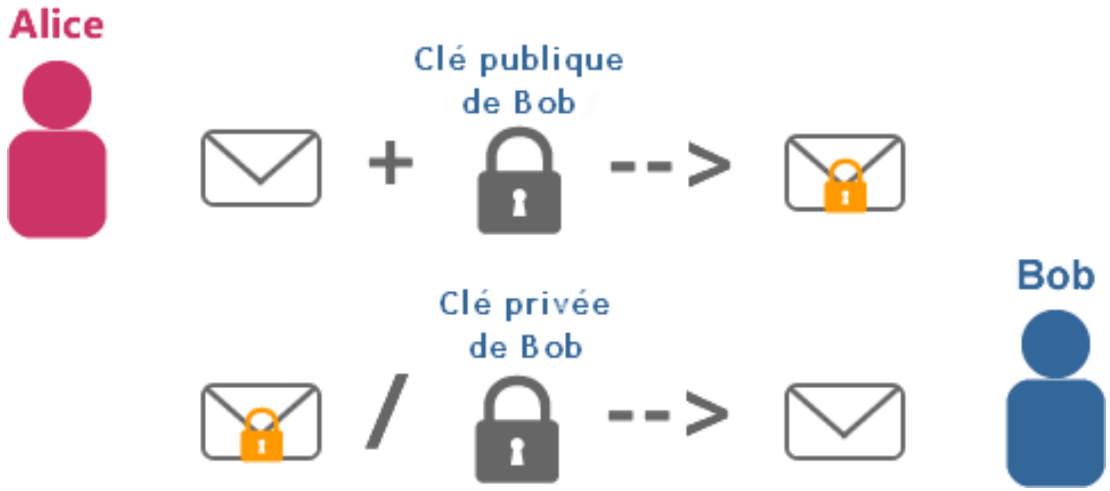
Sommaire

Sommaire

Principe

Définition

Le chiffrement RSA est un algorithme de cryptographie asymétrique.



Génération des clés

- $p, q \in \mathbb{P}$ distincts
- Module de chiffrement : $n = p \times q$
- Indicatrice d'Euler : $\varphi(n) = (p-1)(q-1)$
- Exposant de chiffrement : $e \in \mathbb{N}$ tel que $\begin{cases} e \wedge \varphi(n) = 1 \\ e < \varphi(n) \end{cases}$
- Exposant de déchiffrement : $d \in \mathbb{N}$ tel que $d \times e \equiv 1[\varphi(n)]$
- Clé publique : (n, e)
- Clé privée : (n, d)

Codage / Décodage

Soit M un entier représentant le message clair, C le message encodé.
Encodage : $C = M^e[n]$
Décodage : $M = C^d[n]$

```
1  def generateRSA(long = 500):
2
3    p = millerRabin(10, long)
4    q = millerRabin(10, long)
5
6    n = p * q
7    phi = (p - 1) * (q - 1)
8
9    e = min(p, q)
10   pgcd_e_phi = pgcd(e, phi)
11
12   while pgcd_e_phi != 1:
13     e += 1
14     pgcd_e_phi = pgcd(e, phi)
15
16   d = invModulo(e, phi)
17
18   return (e, n), (d, n)
```

Exemple I

Message : "Hello World !"

Message encodé :

134967637220626576885010373655381596550051634430747850639222
826338123866514840759360429084769725009252148868707790408518
135869093412296136446399084114663775554611717768163178746887
777663235928004008490727585766386138325819064099503939867422
951476940289698809722454115172732913477433296245204850962265
739656667898960958828728545254437726714513663228121008286288
860161868544764876581585760872163873859787300249291866197437
596598870788450293760280117925233465040861368144319717325273
024591344275795999498733612583947183803595871830002498637735
822883079765553441061181140985053747272061564493873534835801
391163475974960467892327513267599540085061434934766167666050

Exemple II

$$
\begin{aligned}
&9755839639254266665194384175941853655624678680674705717058221466987115405905374575080902855541028149153259047865719255142520140592185214350220939576600357103917942019368579456931802047935184079668109727400424494117804053101766855591487923771131536139024006315490832330815233633994362174517048327555967480791202384834502425678036857981154372 0
\end{aligned}
$$

Sommaire

Conditions

Il faut que p et q soient proches de $\sqrt{n}$, i.e. $|p - q| < cn^{\frac{1}{4}}$, où c est une constante assez petite.

Principe

Soient x et y pairs tels que

$$4n = 4pq = (x+y)(x-y) = x^2 - y^2$$

On pose alors

$$p = \frac{x+y}{2}, q = \frac{x-y}{2}$$

Avec un pas de 2, on cherche x en posant $x_0 = \lceil 2\sqrt{n} \rceil$ puis $x_1 = \lceil 2\sqrt{n} \rceil + 2$, etc...
On pose k le nombre entier pour lequel $x_k = \lceil 2\sqrt{n} \rceil + k$ donne la factorisation de n.

Alors $x_k = p + q$ et on a, en supposant que $|p - q| < cn^{\frac{1}{4}}$:

$$\begin{aligned} k &= x_k - \lceil 2\sqrt{n} \rceil \\ &= p + q - \lceil 2\sqrt{n} \rceil \\ &< p + q - 2\sqrt{n} + 1 \\ &= \frac{(p+q)^2 - 4n}{p + q + 2\sqrt{n}} + 1 \\ &= \frac{(p-q)^2}{p + q + 2\sqrt{n}} + 1 \\ &< \frac{c^2\sqrt{n}}{2\sqrt{n}} + 1 \\ &< \frac{c^2}{2} + 1 \end{aligned} \tag{1}$$

```
1  def fermat(n):
2    x =  ceil(2 * sqrt(n))
3    y = sqrt(x ** 2 - 4 * n)
4
5    p = (x + y) / 2
6    q = (x - y) / 2
7
8    while p * q != n:
9      x =  ceil(2 * sqrt(n)) + 2
10     y = sqrt(x ** 2 - 4 * n)
11
12     p = (x + y) / 2
13     q = (x - y) / 2
14
15   return p, q
```

Exemple

Avec $n = 8868043$ on trouve $p = 2957$ et $q = 29999$.

Si on pose $\Delta = |p - q|$ et $\Delta = n^{\alpha}$, cette attaque est possible si $\Delta \leq cn^{\frac{1}{4}}$ et donc $\alpha \leq \frac{1}{4}$.

Sommaire

Définition

Définition

On appelle *fraction continue d'ordre n* toute expression de la forme

$$a_0 + \cfrac{1}{a_1 + \cfrac{1}{a_2 + \cfrac{1}{a_3 + \ldots + \cfrac{1}{a_n}}}}$$

où a_0 est un entier positif et où les $a_i, i \geq 1$ sont des entiers strictements positifs. On note cette expression

$$[a_0, a_1, a_2, \ldots, a_{n-1}, a_n]$$

k-ième réduites

Définition

Soit $[a_0, a_1, \ldots, a_n]$ une fraction continue. On définit la suite (p_k/q_k) par

$$\begin{cases} p_0 = a_0 \\ q_0 = 1 \end{cases} \quad \begin{cases} p_1 = a_0 a_1 + 1 \\ q_0 = a_1 \end{cases}$$

et pour $k \geq 2$,

$$\begin{cases} p_k = a_k p_{k-1} + p_{k-2} \\ q_k = a_k q_{k-1} + q_{k-2} \end{cases}$$

La fraction p_k/q_k s'appelle la *k-ième réduite* de la fraction continue.

Théorème

Soit x un réel positif non nul et soit p/q une fraction irréductible telle que :

$$\left| x - \frac{p}{q} \right| < \frac{1}{2q^2}$$

Alors p/q est une des réduites du développement en fraction continue de x.

Sommaire

Théorème

Si $d < \frac{1}{3} n^{\frac{1}{4}}$ et $q < p < 2q$
Alors on peut retrouver d grâce à un développement en fractions continues de $\frac{e}{n}$
Donc l'attaque de Wiener n'est possible que si $d = n^{\beta}$ tel que $\beta < \frac{1}{4}$.

Pourquoi ça marche ?

Comme $ed \equiv 1[\varphi(n)]$ alors : $\exists k \in \mathbb{Z} | ed - k\varphi(n) = 1$
Donc $|\frac{e}{\varphi(n)} - \frac{k}{d}| = \frac{1}{d\varphi(n)}$ (1)

- On utilise alors n comme approximation de $\varphi(n)$
  En effet $|n - \varphi(n)| = p + q - 1 < 3\sqrt{n}$ car $q < p < 2q$
- On obtient $|\frac{e}{n} - \frac{k}{d}| = |\frac{1-k(n-\varphi(n))}{nd}| < |\frac{3k\sqrt{n}}{nd}|$ (2)

- Or $ke < k\varphi(n) = de - 1 < de$ donc $k < d < \frac{1}{3}n^{\frac{1}{4}}$
- Donc $|\frac{e}{n} - \frac{k}{d}| < \frac{1}{2d^2}$ (3)

Théorème

Si $|x - \frac{p}{q}| < \frac{1}{2q^2}|$ alors $\frac{p}{q}$ est une des réduites du développement en fraction continue de x.

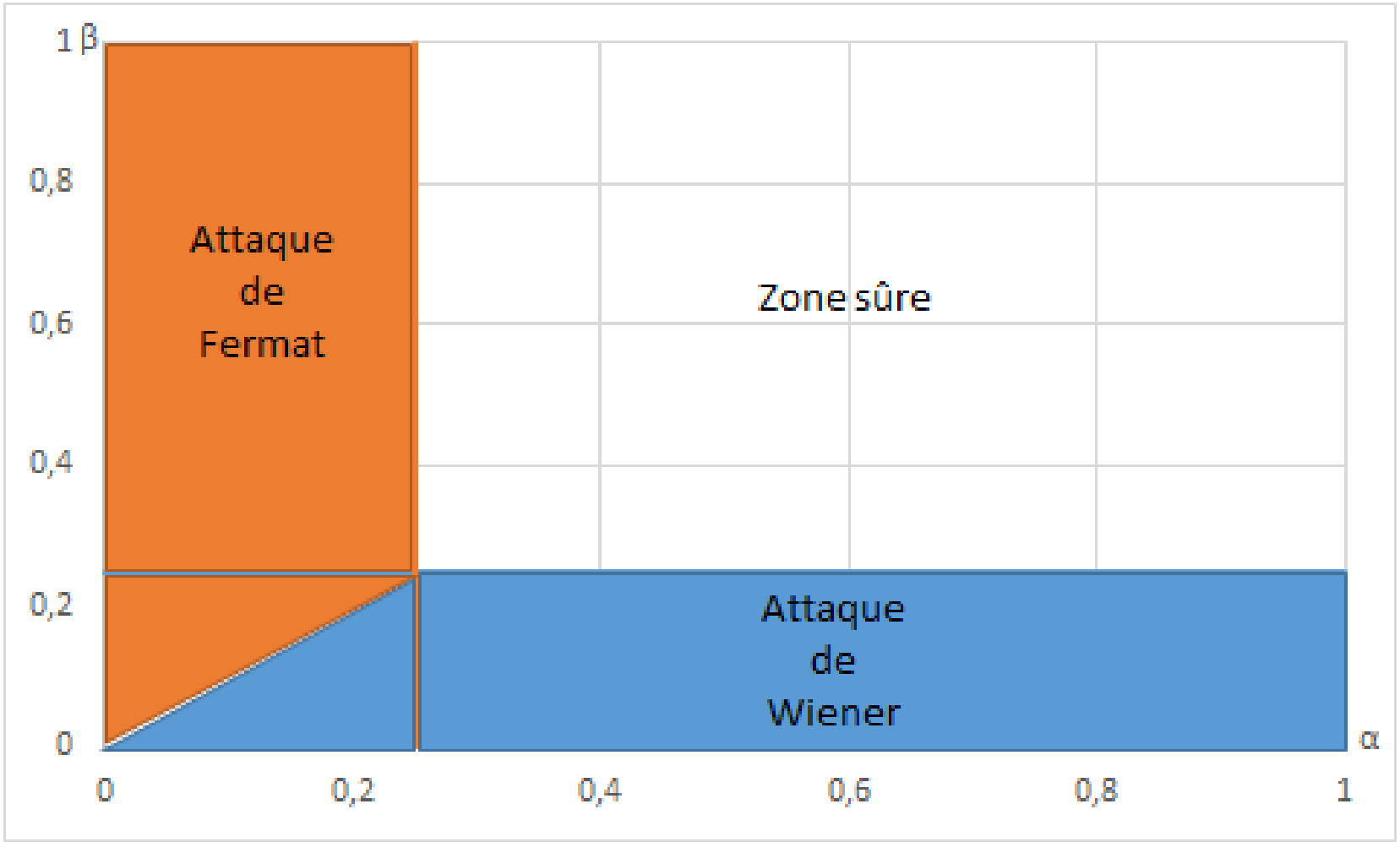
Equation (3) : $|\frac{e}{n} - \frac{k}{d}| < \frac{1}{2d^2}$
Donc $\frac{k}{d}$ est une des réduites de $\frac{e}{n}$

```
1  def wiener(n, e):
2  frac = reduite(e, n)
3  z = len(frac)
4  m = expoMod(12345, e, n)
5
6  for i in range(z):
7    d = frac[i][1]
8    if expoMod(m, d, n) == 12345:
9      return d
```

Exemple

$p = 14737$, $q = 509449$, $n = 7507749913$ et $e = 3217382455$
On trouve alors $d = 7$.

Sommaire


1 β
0,8
0,6
0,4
0,2
0
Attaque de Fermat
Zone sûre
Attaque de Wiener
α
0
0,2
0,4
0,6
0,8
1

Sommaire

```
1
2  from random import randint, randrange
3  from math import sqrt, ceil
4
5  def calculArg(p):
6
7    s = 0
8    d = p - 1
9
10   while d % 2 == 0:
11     s = s + 1
12     d = d // 2
13
14   return s, d
15
16 def temoinMiller(a, p):
17
18   s, d = calculArg(p)
```

```
19    x = expoMod(a, d, p)
20
21    if x == 1 or x == p - 1:
22      return False
23
24    while s > 1:
25      x = x ** 2 % p
26      if x == p - 1:
27        return False
28      s = s - 1
29
30    return True
31
32  def millerRabin(k, long):
33    premier = randint(10**(long-1), 10**long - 1)
34    if premier % 2 == 0:
35      premier += 1
36    i = 0
37    a = randint(2, premier - 2)
```

```
38
39    while i <= k:
40      if temoinMiller(a, premier):
41        i = 0
42        premier += 2
43      else:
44        i += 1
45        a = randint(1, premier)
46
47    return premier
48
49  def generateRSA(long = 500):
50
51    p = millerRabin(10, long)
52    q = millerRabin(10, long)
53
54    n = p * q
55    phi = (p - 1) * (q - 1)
56
```

```
57    e = min(p, q)
58    pgcd_e_phi = pgcd(e, phi)
59
60    while pgcd_e_phi != 1:
61      e += 1
62      pgcd_e_phi = pgcd(e, phi)
63
64    d = invModulo(e, phi)
65
66    return (e, n), (d, n)
67
68  def encodeRSA(message, cle_pub):
69    e, n = cle_pub
70    message_code = ""
71    debut = False
72
73    for i in range(len(message)):
74      lettre = str(ord(message[i]))
75
```

```
76      if len(lettre) < 3 :
77        lettre = "0" + lettre
78        if i == 0 : debut = True
79
80      message_code += lettre
81
82    message_code = expoMod(int(message_code), e, n)
83
84    if debut:
85      message_code = int("0" + str(message_code))
86
87    return message_code
88
89  def decodeRSA(message, cle_priv):
90    d, n = cle_priv
91    debut = False
92
93    message_decode_int = expoMod(message, d, n)
94    message_decode_str = str(message_decode_int)
```

```
95    message_decode = ""
96
97    for i in range(0, len(message_decode_str), 3):
98      if message_decode_str[i] == 0:
99        lettre = int(message_decode_str[i+1:i + 3])
100     else:
101       lettre = int(message_decode_str[i:i + 3])
102     lettre_decode = chr(lettre)
103     message_decode += lettre_decode
104
105   return message_decode
106
107 def pgcd(a,b):
108
109   while a % b != 0:
110     a, b = b, a % b
111   return b
112
113 def expoMod(x, n, mod):
```

```
114
115     if n == 0:
116       return 1
117     if n % 2 == 0:
118       return (expoMod(x, n // 2, mod) ** 2) % mod
119     else:
120       return (x * expoMod(x, (n - 1) // 2, mod) ** 2) %
                                        mod
121
122 def euclide(a, b):
123
124     if b == 0:
125       return (a, 1, 0)
126     else:
127       q, r = divmod(a, b)
128       d, u, v = euclide(b, r)
129       return d, v, u - q * v
130
131 def invModulo(x, n):
```

```
132
133     pgcd_temp, u, v = euclide(x, n)
134
135     return u
136
137 def fermat(n):
138     x =  ceil(2 * sqrt(n))
139     y = sqrt(x ** 2 - 4 * n)
140
141     p = (x + y) / 2
142     q = (x - y) / 2
143
144     while p * q != n:
145         x =  ceil(2 * sqrt(n)) + 2
146         y = sqrt(x ** 2 - 4 * n)
147
148         p = (x + y) / 2
149         q = (x - y) / 2
150
```

```
151     return p, q
152
153 def dfc(a, b):
154     p = a
155     q = b
156     liste = []
157
158     while q != 0:
159         liste.append(p//q)
160         p, q = q, p % q
161
162     return liste
163
164 def reduite(a, b):
165     frac_continue = dfc(a, b)
166     n = len(frac_continue)
167     frac_reduit = [(frac_continue[0], 1), (frac_continue
                                [0]*frac_continue[1] + 1,
                                frac_continue[1])]
```

```
168
169     for i in range(2, n):
170         p = frac_continue[i]*frac_reduit[i - 1][0] +
                                    frac_reduit[i - 2][0]
171         q = frac_continue[i]*frac_reduit[i - 1][1] +
                                    frac_reduit[i - 2][1]
172
173         frac_reduit.append((p, q))
174
175     return frac_reduit
176
177 def wiener(n, e):
178     # p = 14737, q = 509449, n = 7507749913, e =
                                    3217382455, d = 7
179     # exemple : wiener(7507749913,3217382455)
180     frac = reduite(e, n)
181     z = len(frac)
182     m = expoMod(12345, e, n)
183
```

```
184     for i in range(z):
185         d = frac[i][1]
186         if expoMod(m, d, n) == 12345:
187             return d
```