2EL1740 : Algèbre & Cryptologie

CentraleSupélec - 2A

Challenge 5

27 septembre 2024

CentraleSupélec

Raphaël PAIN DIT HERMIER
Alexis LOMBARD-GAILLARD
Edward LUCYSZYN
Julian LOCK

Table des matières

Pour rappel, l'entièreté du challenge s'intéresse au problème du facteur commun approché. Pour λ un entier, notons $P = 2^{\lambda+\log_2 \lambda}$, $Q = 2^{\lambda \log_2 \lambda}$, $R = 2^{\lambda}$. On s'intéressera alors à la difficulté à distinguer un nombre de la forme $pq + r$ (avec $p \approx P$ fixé et secret, et $q \approx Q$, $r \approx R$) d'un nombre entier "aléatoire".

1 Question 1

Dans cette question on considère p un nombre premier secret très proche de P (dans le sens où $|p - P|$ est petit devant P), et des nombres connus $x_i = pq_i + r_i$ où $q_i \xleftarrow{\$} \{0, \dots, Q\}$ et $r_i \xleftarrow{\$} \{0, \dots, R\}$ de manière indépendante et uniforme (dans cette question uniquement, la loi de tirage des q_i et r_i aléatoirement sera sans conséquence dans les questions suivantes). L'objectif est de rendre p difficile à trouver, en trouvant une valeur de λ appropriée.

1.1 Premières remarques et difficulté souhaitée

On supposera dans un premier temps qu'il faut prendre λ relativement grand, sinon un attaquant pourra "brute force" des valeurs de p proches de P jusqu'à facilement tomber sur notre clé secrète. Dès lors on pourra supposer que $Q \gg P \gg R$, donc $Q \gg r_i$ et $p \gg r_i$ pour tout i.

Avant de continuer, il faut proposer une première définition de difficulté. La difficulté s'exprimera sous la forme d'un couple (ℓ, N) où :

- N est la capacité recherchée de x_i publics ($i \in \{1, \dots, n\}$).
- Le nombre de calculs nécessaires pour qu'un attaquant retrouve le premier secret p soit de l'ordre de 2^{ℓ}.

Pour pouvoir envoyer/recevoir assez de messages d'une longueur assez raisonnable (par exemple pour le chiffrement symétrique), on prendra $N = 2^{128}$ pour être large.
Pour une sécurité de 512 bits, on prendra $\ell = 512$.
Mettons-nous pour un instant dans la peau d'un attaquant (emoji qui fait peur). Nous disposons d'au moins 2 ressources pour trouver la clé secrète p :

- Vérifier si un p' candidat est bel et bien la clé secrète p ;
- Calculer des approximations de p secret.

Pour la première ressource, il suffira de calculer le reste de la division euclidienne de x_i par p' pour autant de i que souhaités. p' passe alors le test de vérification si ce reste r'_i est "proche" de $R = 2^{\lambda}$.
Pour la seconde ressource, une méthode d'approximation de p (qui ne repose pas sur P) étant donné une valeur de x_i est de calculer $\dfrac{x_i}{Q}$. En effet,

$$\frac{x_i}{Q} = \frac{pq_i}{Q} + \underbrace{\frac{r_i}{Q}}_{\ll 1} \approx p$$

car $q_i \approx Q$. Plus la valeur de q_i est proche de Q (ce qui est équivalent dans notre approche à "plus x_i est grand"), plus la valeur calculée est une bonne approximation de p. Deux méthodes sont alors possibles pour raffiner cette approximation de p pour un attaquant.

1.2 Première méthode d'approximation de p : choisir le x_i le plus grand

Cette méthode consiste à effectuer l'approximation décrite précédemment uniquement avec le x_i le plus grand, de manière à s'assurer que q_i est le plus proche possible de Q. Cette méthode possède l'avantage pour un attaquant d'être simple et efficace, avec l'inconvénient de reposer sur le fait que les q_i sont tirés aléatoirement de manière **uniforme** entre 1 et la borne maximale Q.

Ces hypothèses étant formulées, on va maintenant pouvoir s'intéresser au maximum des q_i, notons $q_{max} = \max\limits_{i \in \{1,\dots,N\}} q_i$ variable aléatoire sur $\{1, \dots, Q\}$. On sait dans tous les cas que $q_{max} \leq Q$.

Puis, pour $M \in \mathbb{N}^*$ une borne dite de "calculabilité" :

$$\begin{aligned}\mathbb{P}(q_{max} + M \geq Q) &= 1 - \mathbb{P}(q_{max} < Q - M) \\ &= 1 - \mathbb{P}(\forall i \in \{1, \dots, N\}, q_i < Q - M) \\ &= 1 - \prod_{i=1}^{N} \mathbb{P}(q_i < Q - M) \text{ (par indépendance des } q_i) \\ &= 1 - \left(\frac{Q - M}{Q + 1}\right)^N \text{ (loi uniforme)} \\ &= 1 - \left(1 - \frac{M + 1}{Q + 1}\right)^N.\end{aligned}$$

La stratégie de l'attaquant serait alors de tester toutes les valeurs de l'intervalle entier $\{Q - M, \dots, Q\}$ avec la première ressource. Cette stratégie repose sur la capacité de l'attaquant à effectuer un total de $\mathcal{O}(M)$ calculs. En ayant pris une sécurité de $\ell = 512$ bits, on prend $M = 2^{512}$ et on veut que la probabilité obtenue soit elle aussi négligeable à notre niveau de sécurité, i.e $\mathbb{P}(q_{max} + M \geq Q) \leq 2^{-512}$.
Sachant que $Q = 2^{\lambda \log_2(\lambda}$, par la majoration $\lambda \leq \lambda \log_2 \lambda$ on en déduit une première borne

$$\lambda \geq 512 - \log_2(1 - 2^{-2^{9-128}}) \approx 631.$$

1.3 Deuxième méthode d'approximation de p : moyenner les x_i

Cette méthode consiste à effectuer l'approximation décrite précédemment avec plusieurs x_i et d'effectuer leur moyenne. Comme $\mathbb{E}\left[\frac{x_i}{Q}\right] = \frac{p}{2}$, plus l'on a de valeurs de x_i à disposition, plus il est probable que le double de la moyenne des $\frac{x_i}{Q}$ soit une bonne approximation de p (conséquence de la loi forte des grands nombres).
Cette méthode prend autant de temps calculatoire que la première et a l'avantage de pouvoir s'adapter à d'autres lois de tirage des q_i, tant que cette dernière est connue (par exemple si $q_i = arr(a_i)$ où arr est la fonction "arrondir à l'entier le plus proche" et $a_i \sim \mathcal{N}(Q, \sigma^2)$, alors $\mathbb{E}\left[\frac{x_i}{Q}\right] = p$).

On note alors

$$V_N = \frac{1}{N} \sum_{i=1}^{N} \frac{x_i}{Q}.$$

C'est une variable aléatoire d'espérance et de variance

$$\mathbb{E}[V_N] = \frac{p}{2}, \quad \mathbb{V}[V_N] = \frac{1}{N}\mathbb{V}\left[\frac{x_1}{Q}\right] \approx \frac{P^2}{NQ^2}\frac{Q(Q+2)}{12} \approx \frac{P^2}{12N}$$

Car $p \approx P$ q_i suit une loi uniforme sur $\{0, \dots, Q\}$.
Ainsi, pour $M \in \mathbb{N}^*$ une borne dite de "calculabilité", on utilise l'inégalité de Bienaymé-Tchebychev :

$$\begin{aligned}\mathbb{P}(|2V_N - p| \leq M) &= 1 - \mathbb{P}(|V_N - \mathbb{E}[V_N]| > \frac{M}{2}) \\ &\geq 1 - \frac{4\mathbb{V}[V_N]}{M^2} \\ &\geq 1 - \frac{P^2}{3NM^2}\end{aligned}$$

Pour que p soit inatteignable avec une sécurité de ℓ bits (c'est-à-dire $M = 2^{512}$), on impose que $\mathbb{P}(|2V_N - p| \leq M)$ soit négligeable, et donc que l'inégalité de Bienaymé-Tchebychev soit saturée.
On propose alors que $\frac{P^2}{3NM^2} > 1$, c'est-à-dire que $P^2 > 3 \times 2^{1024} \times N$.

On avait pris $N = 2^{128}$. De plus, on majore 3 par 4 de sorte à avoir $P^2 \geq 2^{2+1024+128} = 2^{1154}$. En considérant que $\log_2(\lambda) \ll \lambda$, on prendra

$$\lambda \geq 1154.$$

En combinant les méthodes des deux approches, on prendra $\lambda = 1154$ pour une sécurité qui convient aux deux méthodes.

On garde la valeur de p suivante, probablement premier avec une probabilité supérieure à $1 - 2^{-540}$ d'après les tests de Miller-Rabin :

$$p = 25056185341070159655744468467386665701022029309313351290960034845048708217279266128603899469069 - 50524613256991001538930546461040982222280730637664947823027228101200812911846279295376719749773 - 49220810968318532927168872930418994518829420741149312204893572709752708607032818175191464000110 - 58427103522230242196464658937105954628921212157300497222511204842 3.$$

1.4 Résistance face à NFS

L'attaquant pourrait avoir une autre idée : étant donné une valeur de x_i, on cherche à factoriser $x_i - r'_i$ en prenant r'_i "proche" de $R = 2^\lambda$.
Or $x_i - r'_i$ est de l'ordre de $2^{\lambda + \log_2(\lambda) + \lambda \log_2(\lambda)} \geq 2^{12903}$, dont la complexité de NFS à cette grandeur est de l'ordre de 251 bits, ce qui est largement supérieur à 128 bits. Donc même si notre attaquant avait la chance inouïe de tomber sur la bonne valeur de r_i (qui est quand même à une probabilité de $2^{-\lambda} = 2^{-1154}$), l'algorithme NFS ne parviendrait pas à factoriser ce nombre, et ne lui permettrait pas de retrouver la clé. Donc notre sécurité est garantie face à une attaque "NFS".

2 Question 2

Pour un chiffré c_i donné, la valeur maximale de $2r_i + m_i$ est $2R + 1 \ll P$, le résultat de la question 1 s'applique donc et on peut affirmer que le chiffrement fonctionne.

Le déchiffrement fonctionne :

$$(c_i \bmod p) \bmod 2 = (pq_i + 2r_i + m_i \bmod p) \bmod 2 = (2r_i + m_i \bmod p) \bmod 2 = 2r_i + m_i \bmod 2 = m_i$$

L'avantage IND-CPA de l'attaquant vaut :

$$\mathrm{Adv}^{\text{IND-CPA}}(A) = \left| \mathbb{P}(A(X) = p | X = x) - \frac{1}{\pi(P)} \right| \simeq \left| \frac{1}{2^{128}} - \frac{\ln P}{P} \right| \simeq \left| \frac{1}{2^{128}} - \frac{1}{10^{348}} \right| \simeq \frac{1}{2^{128}}$$

où $\pi(P)$ est le nombre de premiers inférieurs ou égaux à P. Ainsi, le chiffrement est IND-CPA.

L'implémentation est la suivante :

```
import numpy as np

def question2_code_bit(bit, p, lambda_):
    random_q = int(np.random.uniform(2**(lambda_*np.log2(lambda_) - 1), \\
    3*2**(lambda_*np.log2(lambda_) - 1)))
    random_r = int(np.random.uniform(2**(lambda_ - 1), 3*2**(lambda_ - 1)))
    return p*random_q + 2*random_r + bit

def question2_code_number(number, p, lambda_):
    number_2 = bin(number)[2:]
    coded_number = []
    for element in number_2:
        coded_number.append(question2_code_bit(int(element), p, lambda_))
    return coded_number

def question2_uncode(L, p):
```

```
    number = ''
    for element in L:
        number += str(int((element%p)%2))
    return int(number, 2)
```

On observe une limitation principale de ce système : comment est-ce que l'émetteur et le destinaire peuvent se partager la clé secrète p ? Il est possible que l'émetteur et le destinaire ne puissent pas se rencontrer dans la vie réelle (ou alors que l'un des deux est sous surveillance), et une clé envoyée par Internet est vulnérable aux attaques. C'est le problème de la plupart des chiffrements à clé symétrique, et donc cela détruit l'utilité du chiffrement.

3 Question 3

Dans cette question, t désigne la taille de la clé publique **pk**. On supposera alors que $t < \frac{\lambda}{2} - 1$ pour ce chiffrement. Le déchiffrement fonctionne :

$$(c \bmod p) \bmod 2 = \left(m + \sum_i b_i x_i \bmod p\right) \bmod 2 = \left(m + \sum_i b_i \times 2r_i\right) \bmod 2 = m.$$

tant que

$$m + \sum_i b_i \times 2r_i < p$$

ce qui est vrai car $m + \sum_i b_i \times 2r_i \leq 1 + 2^{\lambda+1}t < p$.

L'implémentation en Python est la suivante.

```
def question3_code_bit(bit, p, pk):
    S = bit
    quantity = np.random.choice(len(pk))
    L = np.random.choice(len(pk), size=quantity, replace=False)
    for index in L:
        S += np.random.choice(1)*pk[index]
    return S

def question3_code_number(number, p, pk):
    number_2 = bin(number)[2:]
    coded_number = []
    for element in number_2:
        coded_number.append(question3_code_bit(int(element), p, pk))
    return coded_number

def question3_uncode(L, p, pk):
    number = ''
    for element in L:
        number += str(int((element%p)%2))
    return int(number, 2)
```

On observe une limitation potentielle pour ce chiffrement : un bit m ne peut être chiffré que de 2^t manières différentes (car $(b_0, ..., b_{t-1}) \in \{0,1\}^t$). Si ce nombre est petit, alors un attaquant peut comparer chaque chiffré envoyé avec les manières possibles de chiffrer le bit 0. Si le chiffré concorde avec une de ces manières, alors l'attaquant sait que le chiffré envoyé correspond au bit 0, sinon il correspond au bit 1.

Heureusement on peut prendre n'importe quel valeur de t inférieure stricte à $\dfrac{\lambda}{2} - 1$, donc par exemple $t = 512$ promet 2^{512} manières de chiffrer chaque bit, et ainsi le message chiffré ne peut a posteriori pas être déchiffré sans connaître la clé secrète p.

4 Question 4

Étant donnés x_0, x_1, on a que $|q_0x_1 - q_1x_0| = |q_0r_1 - q_1r_0| < 2QR \ll \min(x_0, x_1)$ de manière très probable. On remarquera de même que $2^{\lambda+1}q_0 < 2QR \ll \min(x_0, x_1)$. Généralisons cette observation : posons

$$L = \begin{pmatrix} 2^{\lambda+1} & x_1 & x_2 & \cdots & x_k \\ 0 & -x_0 & 0 & \cdots & 0 \\ 0 & 0 & -x_0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & \cdots & \cdots & \cdots & -x_0 \end{pmatrix}$$

Alors en posant $\vec{v} := (q_0, q_1, \ldots, q_k) \cdot L = (v_0, ..., v_k)$ où

$$v_0 = 2^{\lambda+1}q_0; \quad \forall i \in \{1, \ldots, k\},\ v_k = q_0x_i - q_ix_0.$$

En regardant ce vecteur par rapport à la norme infinie ($\|\vec{z}\|_\infty = \max_{i\in\{0,\ldots,k\}} |q_i|$), le raisonnement au début de la question s'applique et $\|\vec{v}\|_\infty$ est très petit devant $\min(x_0, ..., x_k)$ très probablement pour des raisons d'ordre de grandeur.

Si on retrouve ce vecteur $\vec{v}$ de "petite" norme, alors on peut calculer $(q_0, \ldots, q_n) = \vec{v} \cdot L^{-1}$ avec

$$L^{-1} = \begin{pmatrix} 2^{-\lambda-1} & 2^{-\lambda-1}\frac{x_1}{x_0} & 2^{-\lambda-1}\frac{x_2}{x_0} & \cdots & 2^{-\lambda-1}\frac{x_k}{x_0} \\ 0 & -\frac{1}{x_0} & 0 & \cdots & 0 \\ 0 & 0 & -\frac{1}{x_0} & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & \cdots & \cdots & \cdots & -\frac{1}{x_0} \end{pmatrix}.$$

Ainsi on retrouve $p = arr\left(\frac{x_i}{q_i}\right)$.

Le code que nous avons essayé sur Sagemath est le suivant. (Il ne marche pas, nous n'avons pas eu le temps de beaucoup essayer.) Pour $\lambda = 30$, nous voulions prendre un nombre premier de l'ordre de 2^{35} ; nous avions donc choisi $p = 34359738421$ qui est premier avec certitude.

```
from sage.all import *

def create_matrix_L(k, lambd, x):
    L = matrix(QQ, k, k)
    L[0, 0] = 2^(lambd + 1)
    for j in range(1, k):
        L[0, j] = x[j]
    for i in range(1, k):
        L[i, i] = -x[0]
    return L

lambd = 30
p = 34359738421 # Secret

found_private_key = False
k = 2  # Commence avec k = 2

while not found_private_key:
    # Creer une liste x avec k elements
    x = [(2^(150) + i)*p + 2^30 for i in range(k)]

    L = create_matrix_L(k, lambd, x)
    L_reduced = L.LLL()
```

```
v = L_reduced[0]
print(v)

vector_p = [x[i]/v[i] for i in range(k) if (v[i] != 0) else 1]
for element in vector_p:
    if np.abs(element - p) < 1:
        found_private_key = True
else:
    k += 1
```

5 Question 5

Soient deux messages m et n. En notant C la fonction de chiffrement et D la fonction de déchiffrement, vérifions que $D(C(m) + C(n)) = m + n$ et que $D(C(m)C(n)) = mn$.
Notons $C(m) = pq_m + 2r_m + m$, $C(n) = pq_n + 2r_n + n$. Alors

$$\begin{aligned} D(C(m) + C(n)) &= D(pq_m + 2r_m + m + pq_n + 2r_n + n) \\ &= (pq_m + 2r_m + m + pq_n + 2r_n + n \bmod p) \bmod 2 \\ &= (2r_m + m + 2r_n + n) \bmod 2 \\ &= m \oplus n. \quad \text{(au sens de l'addition dans } \mathbb{F}_2) \end{aligned}$$

$$\begin{aligned} D(C(m)C(n)) &= (pq_m + 2r_m + m) \times (pq_n + 2r_n + n) \bmod p \bmod 2 \\ &= 4r_m r_n + 2r_m n + 2r_n m + nm \bmod p \bmod 2 \\ &= 4r_m r_n + 2r_m n + 2r_n m + nm \bmod 2 \\ &= nm. \end{aligned}$$

Ces deux résultats sont valides si :

- $2r_n + 2r_m < p$ dans le cas de la somme,
- $4r_m r_n + 2r_m n + 2r_n < p$ dans le cas du produit.

Sous ces conditions on a bien un chiffrement homomorphe pour la somme et le produit mais si elles ne sont pas respectées, le déchiffrement peut devenir impossible :
Prenons l'exemple trivial de $p = 5$, on a dans ce cas : $6 \bmod p \bmod 2 = 1$ tandis que $6 \bmod 2 = 0$.

Ainsi, pour permettre 10 multiplications et 10 additions (quelle que soit l'interprétation que l'on fait de cette formulation), prenons de la marge et cherchons une condition pour permettre 20 multiplications. Au vu des calculs précédemment menés, une condition suffisante est $2^{40}R^{20} \ll P$.
Le facteur 2^{40} provient du coefficient 2^{20} devant le terme en $r_1 r_2 \dots r_{20}$ multiplié au nombre 2^{20} de termes qui ne sont pas des multiples de p à considérer - les autres termes étant des multiples de p, ils seront annulés par le modulo p.
On peut donc choisir le nouvelles valeurs de P, Q, R en conséquence en gardant $R = 2^\lambda$ et en prenant les nouvelles valeurs pour P et Q afin de respecter les conditions précédentes. :

$$P = 2^{20\lambda + 20\log_2(\lambda) + 40}, \quad Q = 2^{20\lambda \log_2(\lambda)}.$$

En effet :

$$\begin{aligned} \frac{2^{40}R^{20}}{P} &= \frac{2^{40}2^{\log_2(R^{20})}}{P} \\ &= \frac{2^{40+20\log_2(R)}}{P} \\ &= \frac{2^{40+20\log_2(2^\lambda)}}{P} \\ &= \frac{2^{40+20\lambda}}{2^{20\lambda+20\log_2(\lambda)+40}} \\ &= 2^{-20\log_2(\lambda)} \ll 1 \text{ puisque } \lambda \text{ est grand.} \end{aligned}$$

Pour finir, on peut vérifier qu'avec ces nouvelles valeurs de P et Q, les majorations telles que $Q \gg P \gg R$ et $2QR \ll x$ sont toujours valides, ce qui ne change pas les résultats des questions précédentes.