

Applications des réseaux euclidiens à la cryptanalyse



Plan

Résolution de programmes linéaires entiers

Cryptanalyse de RSA avec Petits exposants secrets

Cryptanalyse des générateurs linéaires congruentiels tronqués

Résolution de Subset-Sum

Chiffrement homomorphe sur les entiers

Algorithme de Coppersmith

Plan

Résolution de programmes linéaires entiers

Cryptanalyse de RSA avec Petits exposants secrets

Cryptanalyse des générateurs linéaires congruentiels tronqués

Résolution de Subset-Sum

Chiffrement homomorphe sur les entiers

Algorithme de Coppersmith

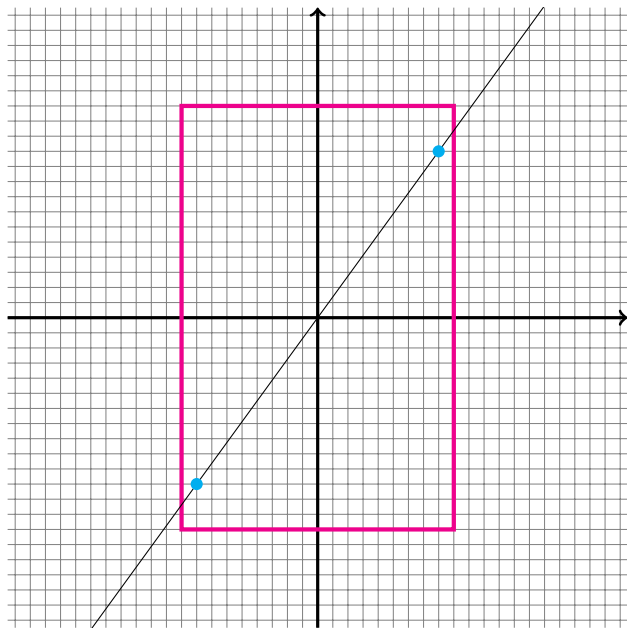
RSA avec petits exposants secrets

$$\left\{ \begin{array}{l} ed - k(N+1) + y = 0 \\ 0 \leq d \leq N^\alpha \\ 0 \leq k \leq N^\alpha \\ 0 \leq y \leq 2N^{\frac{1}{2}+\alpha} \end{array} \right.$$

Généralisation

$$(\mathcal{S}) \left\{ \begin{array}{l} \sum_{i=1}^n a_i x_i = 0 \\ |x_i| \leq B_i \quad (1 \leq i \leq n) \\ x \neq 0 \end{array} \right.$$

Géométriquement



« Programmation » linéaire avec des réseaux euclidiens

$$(\mathcal{S}) \left\{ \begin{array}{l} \sum_{i=1}^n a_i x_i = 0 \\ |x_i| \leq B_i \quad (1 \leq i \leq n) \end{array} \right. x \neq 0$$

$$M = \begin{pmatrix} 1 & & & a_1 \\ & 1 & & a_2 \\ & & \ddots & \vdots \\ & & & 1 & a_n \end{pmatrix}$$

- Idée : $xM = (x \mid 0)$
- Vecteur court dans $\mathcal{L}(M) \rightsquigarrow$ solution de (\mathcal{S}) ?

« Programmation » linéaire avec des réseaux euclidiens

$$(\mathcal{S}) \left\{ \begin{array}{l} \sum_{i=1}^n a_i x_i = 0 \\ |x_i| \leq B_i \quad (1 \leq i \leq n) \end{array} \right. x \neq 0$$

$$M = \begin{pmatrix} 1 & & & a_1 \\ & 1 & & a_2 \\ & & \ddots & \vdots \\ & & & 1 & a_n \end{pmatrix}$$

- Idée : $xM = (x \mid 0)$
- Vecteur court dans $\mathcal{L}(M) \rightsquigarrow$ solution de (\mathcal{S}) ?
- **PB #1** : $(\text{????} \mid y) \in \mathcal{L}(M)$ avec $y \neq 0$, de norme faible

« Programmation » linéaire avec des réseaux euclidiens

$$(\mathcal{S}) \left\{ \begin{array}{l} \sum_{i=1}^n a_i x_i = 0 \\ |x_i| \leq B_i \quad (1 \leq i \leq n) \end{array} \right. x \neq 0$$

$$M = \begin{pmatrix} 1 & & & \lambda a_1 \\ & 1 & & \lambda a_2 \\ & & \ddots & \vdots \\ & & & 1 & \lambda a_n \end{pmatrix}$$

- Idée : $xM = (x \mid 0)$
- Vecteur court dans $\mathcal{L}(M) \rightsquigarrow$ solution de (\mathcal{S}) ?
- **PB #1** : $(\text{????} \mid y) \in \mathcal{L}(M)$ avec $y \neq 0$, de norme faible
 - $y \neq 0 \implies \|(\text{????} \mid y)\| \geq \lambda$

« Programmation » linéaire avec des réseaux euclidiens

$$(\mathcal{S}) \left\{ \begin{array}{l} \sum_{i=1}^n a_i x_i = 0 \\ |x_i| \leq B_i \quad (1 \leq i \leq n) \end{array} \right. x \neq 0$$

$$M = \begin{pmatrix} 1 & & & \lambda a_1 \\ & 1 & & \lambda a_2 \\ & & \ddots & \vdots \\ & & & 1 & \lambda a_n \end{pmatrix}$$

- Idée : $xM = (x \mid 0)$
- Vecteur court dans $\mathcal{L}(M) \rightsquigarrow$ solution de (\mathcal{S}) ?
- **PB #1** : $(\text{????} \mid y) \in \mathcal{L}(M)$ avec $y \neq 0$, de norme faible
 - $y \neq 0 \implies \|(\text{????} \mid y)\| \geq \lambda$
- **PB #2** : $B_i \lll B_j$ et SVP « sacrifie » x_i pour réduire x_j

« Programmation » linéaire avec des réseaux euclidiens

$$(\mathcal{S}) \left\{ \begin{array}{l} \sum_{i=1}^n a_i x_i = 0 \\ |x_i| \leq B_i \quad (1 \leq i \leq n) \end{array} \right. x \neq 0$$

$$M = \begin{pmatrix} \frac{B}{B_1} & & & \lambda a_1 \\ & \frac{B}{B_2} & & \lambda a_2 \\ & & \ddots & \vdots \\ & & & \frac{B}{B_n} \lambda a_n \end{pmatrix} \quad B = \prod_i B_i$$

- Idée : $xM = (x \mid 0)$
- Vecteur court dans $\mathcal{L}(M) \rightsquigarrow$ solution de (\mathcal{S}) ?
- **PB #1** : $(\text{????} \mid y) \in \mathcal{L}(M)$ avec $y \neq 0$, de norme faible
 - $y \neq 0 \implies \|(\text{????} \mid y)\| \geq \lambda$
- **PB #2** : $B_i \lll B_j$ et SVP « sacrifie » x_i pour réduire x_j
 - $\rightsquigarrow xM \approx (B, B, \dots, B, 0)$

« Programmation » linéaire avec des réseaux euclidiens

$$(\mathcal{S}) \left\{ \begin{array}{l} \sum_{i=1}^n a_i x_i = 0 \\ |x_i| \leq B_i \quad (1 \leq i \leq n) \end{array} \right. x \neq 0$$

$$M = \begin{pmatrix} \frac{B}{B_1} & & & \lambda a_1 \\ & \frac{B}{B_2} & & \lambda a_2 \\ & & \ddots & \vdots \\ & & & \frac{B}{B_n} \lambda a_n \end{pmatrix} \quad B = \prod_i B_i$$

- ▶ Idée : $xM = (x \mid 0)$
- ▶ Vecteur court dans $\mathcal{L}(M) \rightsquigarrow$ solution de (\mathcal{S}) ?
- ▶ **PB #1** : $(\text{????} \mid y) \in \mathcal{L}(M)$ avec $y \neq 0$, de norme faible
 - ▶ $y \neq 0 \implies \|(\text{????} \mid y)\| \geq \lambda$
- ▶ **PB #2** : $B_i \lll B_j$ et SVP « sacrifie » x_i pour réduire x_j
 - $\rightsquigarrow xM \approx (B, B, \dots, B, 0)$ et $\|xM\| \leq \sqrt{n}B \rightarrow \lambda = 2\sqrt{n}B$ est OK

« Programmation » linéaire avec des réseaux euclidiens

$$(S) \begin{cases} \sum_{i=1}^n a_i x_i = 0 \\ |x_i| \leq B_i \quad (1 \leq i \leq n) \end{cases} x \neq 0$$

$$M = \begin{pmatrix} \frac{B}{B_1} & & & \lambda a_1 \\ & \frac{B}{B_2} & & \lambda a_2 \\ & & \ddots & \vdots \\ & & & \frac{B}{B_n} \lambda a_n \end{pmatrix} \quad B = \prod_i B_i, \quad \lambda = 2\sqrt{n}B$$

Algorithme

- ▶ Calculer $y \leftarrow \text{SHORTESTVECTOR}(M)$
- ▶ $y_{n+1} \neq 0$ ou bien $y_i > B \Rightarrow$ pas de solution
- ▶ Sinon renvoyer $x_i \leftarrow y_i \frac{B_i}{B} \quad (1 \leq i \leq n)$

« Programmation » linéaire avec des réseaux euclidiens

$$(\mathcal{S}) \begin{cases} \sum_{i=1}^n a_i x_i = 0 \\ \sum_{i=1}^n b_i x_i = 0 \\ |x_i| \leq B_i \quad (1 \leq i \leq n) \end{cases} \mathbf{x} \neq \mathbf{0}$$

$$M = \begin{pmatrix} \frac{B}{B_1} & & & \lambda a_1 & \lambda b_1 \\ & \frac{B}{B_2} & & \lambda a_2 & \lambda b_2 \\ & & \ddots & \vdots & \\ & & & \frac{B}{B_n} & \lambda a_n & \lambda b_n \end{pmatrix} \quad B = \prod_i B_i, \quad \lambda = 2\sqrt{n}B$$

Algorithme

- ▶ Calculer $\mathbf{y} \leftarrow \text{SHORTESTVECTOR}(M)$
- ▶ $y_{n+1} \neq 0, y_{n+2} \neq 0$ ou bien $y_i > B \Rightarrow$ pas de solution
- ▶ Sinon renvoyer $x_i \leftarrow y_i \frac{B_i}{B} \quad (1 \leq i \leq n)$

« Programmation » linéaire avec des réseaux euclidiens

$$(S) \begin{cases} \mathbf{x}A = 0 \\ |\mathbf{x}_i| \leq B_i \quad (1 \leq i \leq n) \end{cases} \mathbf{x} \neq 0$$

$$M = \begin{pmatrix} \frac{B}{B_1} & & & \\ & \frac{B}{B_2} & & \\ & & \ddots & \\ & & & \frac{B}{B_n} \end{pmatrix} \begin{matrix} \boxed{\lambda A} \end{matrix} \quad B = \prod_i B_i, \quad \lambda = 2\sqrt{n}B$$

Algorithme

- ▶ Calculer $\mathbf{y} \leftarrow \text{SHORTESTVECTOR}(M)$
- ▶ $y_{n+i} \neq 0$ ou $y_i > B \Rightarrow$ pas de solution
- ▶ Sinon renvoyer $\mathbf{x}_i \leftarrow y_i \frac{B_i}{B} \quad (1 \leq i \leq n)$

« Programmation » linéaire avec des réseaux euclidiens

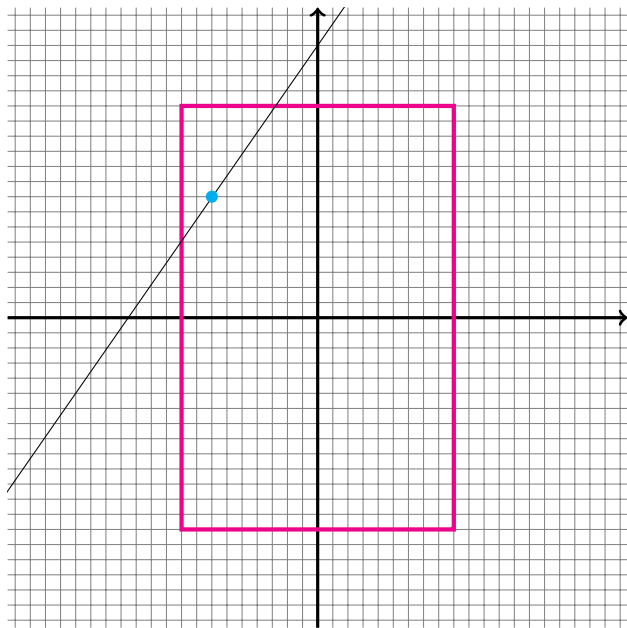
$$(S) \begin{cases} \mathbf{x}A = \mathbf{b} \\ |\mathbf{x}_i| \leq B_i \quad (1 \leq i \leq n) \end{cases} \mathbf{x} \neq 0$$

$$M = \begin{pmatrix} \frac{B}{B_1} & & & & \\ & \frac{B}{B_2} & & & \\ & & \ddots & & \\ & & & \frac{B}{B_n} & \\ & & & & \boxed{\lambda A} \end{pmatrix} \quad B = \prod_i B_i, \quad \lambda = 2\sqrt{n}B$$
$$z = \begin{pmatrix} 0 & 0 & \dots & 0 & \boxed{\lambda \mathbf{b}} \end{pmatrix}$$

Algorithme

- ▶ Calculer $\mathbf{y} \leftarrow \text{CLOSESTVECTOR}(M, z)$
- ▶ $y_{n+i} \neq \mathbf{b}_i$ ou $y_i > B \Rightarrow$ pas de solution
- ▶ Sinon renvoyer $\mathbf{x}_i \leftarrow y_i \frac{B_i}{B} \quad (1 \leq i \leq n)$

Géométriquement



« Programmation » linéaire avec des réseaux euclidiens

$$(S) \begin{cases} \mathbf{x}A = \mathbf{b} \pmod{p} \\ |\mathbf{x}_i| \leq B_i \quad (1 \leq i \leq n) \end{cases} \mathbf{x} \neq 0$$

$$M = \begin{pmatrix} \frac{B}{B_1} & & & & & \\ & \frac{B}{B_2} & & & & \\ & & \ddots & & & \\ & & & \frac{B}{B_n} & & \\ & & & & \boxed{\lambda A} & \\ & & & & & \lambda p \\ & & & & & \ddots \\ & & & & & & \lambda p \end{pmatrix}$$

$$\mathbf{z} = \begin{pmatrix} 0 & 0 & \dots & 0 & \boxed{\lambda \mathbf{b}} \end{pmatrix}$$

$$B = \prod_i B_i, \quad \lambda = 2\sqrt{n}B$$

Algorithme

idem

Plan

Résolution de programmes linéaires entiers

Cryptanalyse de RSA avec Petits exposants secrets

Cryptanalyse des générateurs linéaires congruentiels tronqués

Résolution de Subset-Sum

Chiffrement homomorphe sur les entiers

Algorithme de Coppersmith

Signature RSA

Génération de clef classique

1. Choisir $p, q \approx 2^{n/2}$ aléatoires, premiers.
2. Calculer $N = pq$ et $\phi(N) = (p - 1)(q - 1)$
3. Choisir $e = 3$
4. Si e n'est pas inversible modulo $\phi(N)$, retourner en 1.
5. Calculer $d \leftarrow e^{-1} \bmod \phi(N)$

À la fin, $ed \equiv 1 \bmod \phi(N)$

Bilan : $N \approx 2^n$, $e = 3$, $d \approx 2^n$.

- ▶ Signature = $2n$ multiplications $\bmod N \rightsquigarrow \mathcal{O}(n^3)$.
- ▶ Vérification = 2 multiplications $\bmod N \rightsquigarrow \mathcal{O}(n^2)$.

Signer est plus lent que vérifier

La CB signe, le terminal vérifie... dommage !
Peut-on faire l'inverse ?

Génération de clef avec petit exposant secret

1. Choisir $p, q \approx 2^{n/2}$ aléatoires, premiers
2. Calculer $N = pq$ et $\phi(N) = (p - 1)(q - 1)$
3. Choisir ~~$d \approx 3$~~ $d \approx 2^{128}$ aléatoire
4. Si d n'est pas inversible modulo $\phi(N)$, retourner en 1.
5. Calculer $e \leftarrow d^{-1} \bmod \phi(N)$ À la fin, $ed \equiv 1 \bmod \phi(N)$

Bilan : $N \approx 2^n$, $e \approx 2^n$, $d \approx 2^{128}$.

- ▶ Signature = 256 multiplications mod $N \rightsquigarrow \mathcal{O}(n^2)$.
- ▶ Vérification = $2n$ multiplications mod $N \rightsquigarrow \mathcal{O}(n^3)$.

C'est $16\times$ plus rapide, mais...

Est-ce sûr ?

Application #1



RSA avec petit exposant secret

Attaque de Michael J. Wiener, 1989

Situation

- ▶ ENTRÉE : e, N
- ▶ PROMESSES : $ed = 1 + k \cdot \phi(N)$ et $d \leq N^\alpha$ (« petit »)
- ▶ BUT : retrouver d ou $\phi(N)$.

$$B = \begin{pmatrix} \sqrt{N} & -e \\ 0 & N \end{pmatrix} \quad \text{[Pourquoi?} \rightarrow \text{poly]}$$

Vecteur (inconnu) « intéressant » : $\mathbf{x} = (d, k)B = (d\sqrt{N}, kN - ed)$

Applications des diapos précédentes

- ▶ Volume du réseau : $\text{Vol}(\mathcal{L}) = |\det B| = N\sqrt{N} = N^{3/2}$.
- ▶ Heuristique gaussienne suggère $\lambda_1(\mathcal{L}) \approx \sqrt{\text{Vol}(\mathcal{L})} \approx N^{3/4}$.

\Rightarrow Si $\|\mathbf{x}\| \lll N^{3/4}$, on retrouve \mathbf{x} avec SVP $\Rightarrow d$

RSA avec petit exposant secret (suite & fin)

Michael J. Wiener, 1989

Situation

$$\blacktriangleright ed = 1 + k \cdot \phi(N)$$

$$\blacktriangleright \mathbf{x} = (d, k)B = (d\sqrt{N}, kN - ed)$$

$$\blacktriangleright B = \begin{pmatrix} \sqrt{N} & -e \\ 0 & N \end{pmatrix}$$

CLAIM : $\|\mathbf{x}\| \approx d\sqrt{N}$

Il suffit de montrer $kN - ed \approx d\sqrt{N}$.

$$1. \quad kN - ed = k(N - \phi(N)) - 1.$$

$$2. \quad k = (ed - 1)/\phi(N) < ed/\phi(N) < d$$

$$3. \quad N - \phi(N) = N - (p-1)(q-1) = p + q - 1 \approx \sqrt{N}. \text{ CQFD}$$

Conclusion : si $d < N^{1/4}$, alors...

$$\blacktriangleright \|\mathbf{x}\| < N^{3/4} \rightsquigarrow \mathbf{x} = \text{plus court vecteur}$$

\blacktriangleright Résoudre SVP en dim. 2 révèle d

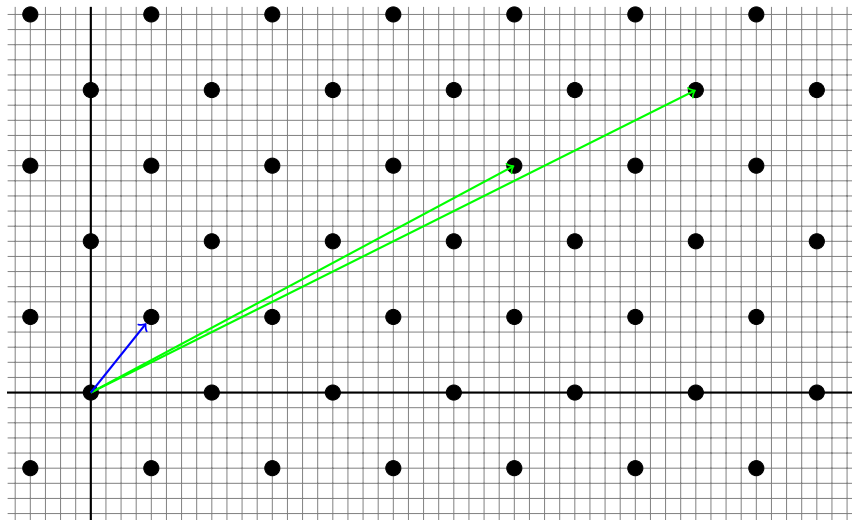
Debriefing et retour sur les réseaux

- ▶ Paramètres proposés ($N \approx 2048$ bits, $d \approx 128$ bits) **cassés** !

```
import fpylll                                # cf. https://github.com/fplll/fpylll
def attack(N, e):                             # temps d'exécution : 1.5ms
    M = fpylll.IntegerMatrix(2, 2)
    s = int(sqrt(N))
    M[0, 0] = s
    M[0, 1] = -e
    M[1, 1] = N
    x = fpylll.SVP.shortest_vector(M)
    return abs(x[0] // s)
```

- ▶ On peut construire une **mauvaise** base du réseau avec les informations publiques (vecteurs de taille $\approx N$)
- ⇒ Le plus court vecteur est de taille $\lll N^{3/4}$
- ▶ Une **super** base contient directement les vecteurs courts...

Partir sur de bonnes bases (SVP)



Plan

Résolution de programmes linéaires entiers

Cryptanalyse de RSA avec Petits exposants secrets

Cryptanalyse des générateurs linéaires congruentiels tronqués

Résolution de Subset-Sum

Chiffrement homomorphe sur les entiers

Algorithme de Coppersmith

Norme POSIX : spécification (obligatoire) de `rand48()`

```
uint64_t rand48_state;

void srand48(uint32_t seed) {
    rand48_state = seed;
    rand48_state = 0x330e + (rand48_state << 16);
}

uint32_t rand48() { /* renvoie 32 bits ``aléatoires'' */
    rand48_state = (0x5deece66d * rand48_state + 11) & 0xffffffffffff;
    return (rand48_state >> 16);
}
```

Spécification de C : suggestion d'implantation de `rand()`

```
static unsigned long int next = 1;

void srand(unsigned int seed) {
    next = seed;
}

int rand(void) { /* renvoie 15 bits ``aléatoires'' */
    next = next * 1103515245 + 12345;
    return ((unsigned)(next/65536) % 32768);
}
```

Utilisation typique des réseaux en cryptanalyse

Procédure habituelle #1

1. Infos publiques \rightsquigarrow base d'un réseau
2. vecteur spécial \mathbf{x} est « anormalement court »
3. Résoudre **SVP** \rightsquigarrow obtenir \mathbf{x}
4. Extraire **secrets** de \mathbf{x}

Procédure habituelle #2

1. Infos publiques \rightsquigarrow base d'un réseau et \mathbf{y}
2. vecteur spécial \mathbf{x} est « anormalement proche » de \mathbf{y}
3. Résoudre **CVP** sur \mathbf{y} \rightsquigarrow obtenir \mathbf{x}
4. Extraire **secrets** de \mathbf{x}

- ▶ PB #1 : être sûr que \mathbf{x} est bien le plus court/proche
- ▶ PB #2 : résoudre **SVP/CVP** (potentiellement difficile)

Utilisation typique des réseaux en cryptanalyse

Problème #1

Comment être sûr que \mathbf{x} est bien le plus **court**?

Heuristique gaussienne

$$\lambda_1(\mathcal{L}) \approx \sqrt{n} (\text{Vol}(\mathcal{L}))^{\frac{1}{n}}.$$

Si on **sait d'avance** que le réseau contient un vecteur spécial \mathbf{x} **sensiblement plus court** que ce que « prédit » l'heuristique gaussienne, alors c'est **probablement** le plus court du réseau.

Application #2



Générateur congruentiel linéaire **tronqué** (Knuth, 1980)

- ▶ Graine = X_0
- ▶ $X_{i+1} = aX_i \bmod m$ et $Y_i = \lfloor X_i/k \rfloor$
- ▶ Sortie = Y_0, Y_1, Y_2, \dots

Les bits de poids faibles des X_i sont **masqués**

Propriété attendue

Sortie indistinguishable de bits « vraiment » aléatoires

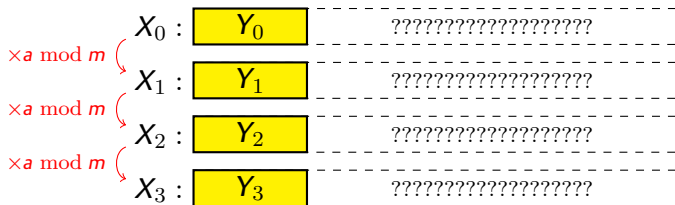
Attaque

- ▶ Si on peut **prédire** la sortie, alors on a un distingueur
 - ▶ (Il suffit de vérifier si la prédiction est correcte)
- ▶ Si on **recupère la graine**, on peut prédire la sortie...

Générateur congruentiel linéaire tronqué (Knuth, 1980)

- ▶ Graine = X_0
- ▶ $X_{i+1} = aX_i \bmod m$ et $Y_i = \lfloor X_i/k \rfloor$
- ▶ Sortie = Y_0, Y_1, Y_2, \dots

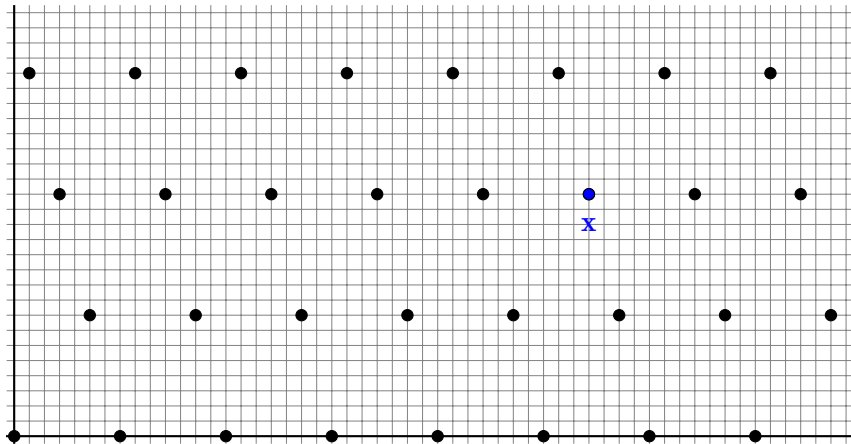
Hypothèses : a, m connus, $b = 0$



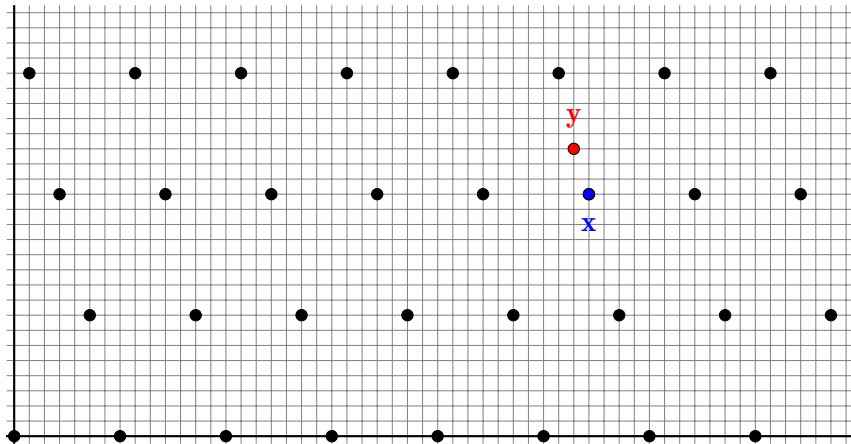
Stratégie : **construire un réseau qui contient X_0**

- ▶ $\mathbf{x} = (X_0, X_1, \dots)$ (inconnu)
- ▶ $\mathbf{y} = k \times (Y_0, Y_1, \dots)$ (connu, « approximation » de \mathbf{x})

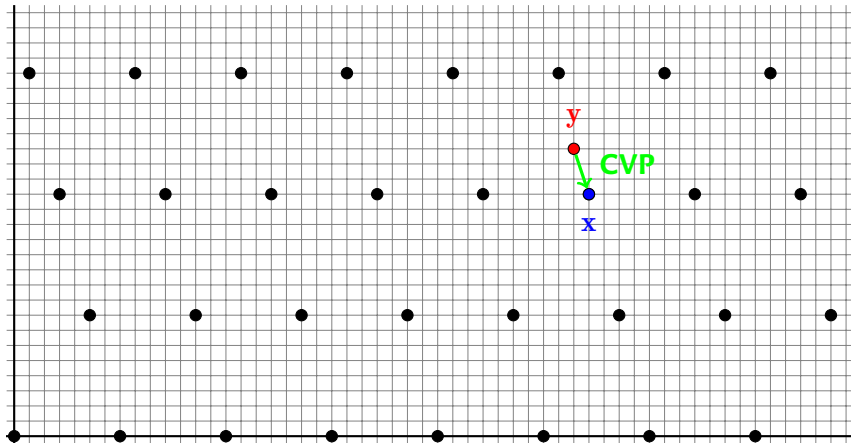
Application de la stratégie standard # 2



Application de la stratégie standard # 2



Application de la stratégie standard # 2



Stratégie : **construire un réseau qui contient X_0**

- ▶ $\mathbf{x} = (X_0, X_1, \dots)$ (inconnu)
- ▶ $\mathbf{y} = k \times (Y_0, Y_1, \dots)$ (connu, « approximation » de \mathbf{x})

$$X_{i+1} = aX_i \bmod m \quad \Longleftrightarrow \quad X_{i+1} = a^i X_0 + q_i m$$

Par conséquent :

$$(X_0, q_1, \dots, q_{n-1}) \begin{pmatrix} 1 & a & a^2 & \dots & a^{n-1} \\ & m & & & \\ & & m & & \\ & & & \ddots & \\ & & & & m \end{pmatrix} = (X_0, X_1, \dots, X_{n-1})$$

(technique générale pour ramener les problèmes $\bmod m$ à des problèmes sur les entiers)

Situation

► $\mathbf{x} = (X_0, X_1, \dots)$

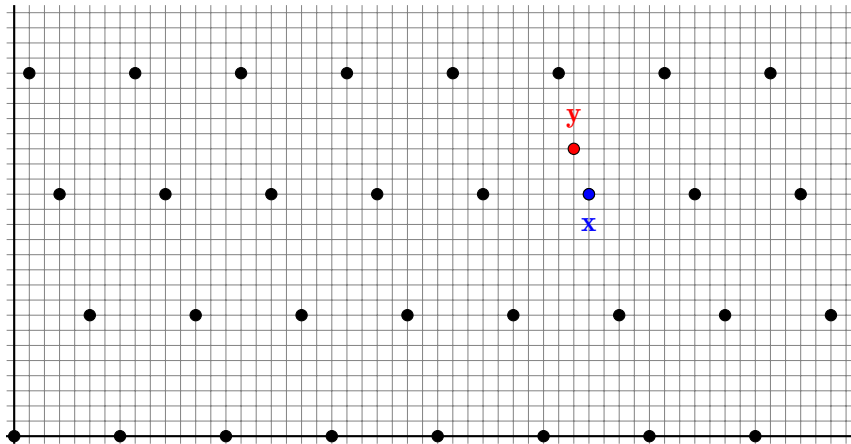
► $\mathbf{y} = k \times (Y_0, Y_1, \dots)$

►
$$\begin{pmatrix} 1 & a & a^2 & \dots & a^{n-1} \\ & m & & & \\ & & m & & \\ & & & \ddots & \\ & & & & m \end{pmatrix}$$

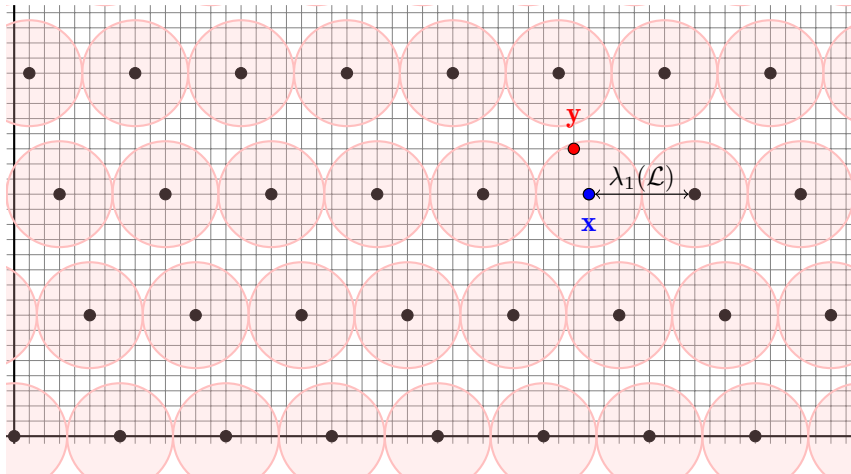
Problème potentiel

Est-ce que \mathbf{x} est le plus proche vecteur de \mathbf{y} dans le réseau ?

Application de la stratégie standard # 2



Application de la stratégie standard # 2



Situation

► $\mathbf{x} = (X_0, X_1, \dots)$

► $\mathbf{y} = k \times (Y_0, Y_1, \dots)$

►
$$\begin{pmatrix} 1 & a & a^2 & \dots & a^{n-1} \\ & m & & & \\ & & m & & \\ & & & \ddots & \\ & & & & m \end{pmatrix}$$

Problème potentiel

Est-ce que \mathbf{x} est le plus proche vecteur de \mathbf{y} dans le réseau ?

Solution

- Si $\|\mathbf{y} - \mathbf{x}\| < \frac{1}{2}\lambda_1(\mathcal{L})$, c'est **garanti**
- Heuristique gaussienne $\rightsquigarrow \lambda_1(\mathcal{L})$

Situation

- ▶ $Y_i = \lfloor X_i/k \rfloor$
- ▶ $\mathbf{x} = (X_0, X_1, \dots)$
- ▶ $\mathbf{y} = k \times (Y_0, Y_1, \dots)$
- ▶ $\|\mathbf{y} - \mathbf{x}\| < \frac{1}{2}\lambda_1(\mathcal{L})$?

$$\rightarrow \begin{pmatrix} 1 & a & a^2 & \dots & a^{n-1} \\ & m & & & \\ & & m & & \\ & & & \ddots & \\ & & & & m \end{pmatrix}$$

Primo : $\|\mathbf{x} - \mathbf{y}\| < k\sqrt{n}$

$$X_i - kY_i = X_i \bmod k < k, \text{ donc } \|\mathbf{x} - \mathbf{y}\| < \sqrt{k^2 + \dots + k^2} = k\sqrt{n}$$

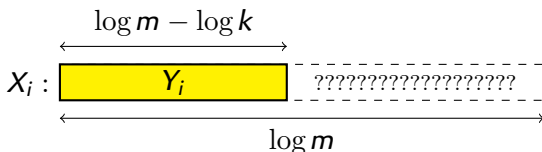
Secundo : $\lambda_1(\mathcal{L}) \approx m^{1-1/n}\sqrt{n}$

$\text{Vol}(\mathcal{L}) = m^{n-1}$. Heuristique gaussienne : $\lambda_1(\mathcal{L}) \approx \sqrt{n} \text{Vol}(\mathcal{L})^{1/n}$.

Calculer CVP sur \mathbf{y} renvoie bien \mathbf{x} si...

$$k\sqrt{n} < m^{1-1/n}\sqrt{n}, \text{ donc } n > \frac{\log m}{\log m - \log k}$$

Discussion



- ▶ Observer n sorties du PRG puis résoudre CVP en dim. n
- ▶ $n > \log m / (\log m - \log k)$
 - ▶ X_i grand \rightarrow plus dur
 - ▶ Y_i petit \rightarrow plus dur
- ▶ Paramètres proposés ($m = 2^{128}, k = 2^{120}$) $\rightsquigarrow n > 16$.
- ▶ CVP en dimension ≈ 20 = facile \implies cassé!

```
import fpylll                # cf. https://github.com/fpylll/fpylll
def attack(y):                # temps d'exécution : 8ms
    # [----REDACTED-----]
    fpylll.LLL.reduction(M)    # requis par CVP.closest_vector()
    x = fpylll.CVP.closest_vector(M, y)
    return x[0]
```

- ▶ CVP en grande dimension?

Plan

Résolution de programmes linéaires entiers

Cryptanalyse de RSA avec Petits exposants secrets

Cryptanalyse des générateurs linéaires congruentiels tronqués

Résolution de Subset-Sum

Chiffrement homomorphe sur les entiers

Algorithme de Coppersmith

Fonction à sens unique *subset sum*

I Génère n entiers (A_i) **aléatoires** sur m bits

► Pas de trucs supercroissants bizarres

V Sur une entrée de n bits (x_1, x_2, \dots, x_n) , calcule :

$$\mathcal{F}_{(A_i)}(x) = \sum_{i=1}^n x_i A_i \mod 2^m$$

But (rappel : c'est **NP-dur**)

Étant donné $y \leq 2^m$, trouver $x \in \{0, 1\}^n$ tel que $F(x) = y$

Fonction à sens unique *subset sum*

I Génère n entiers (A_i) **aléatoires** sur m bits

► Pas de trucs supercroissants bizarres

V Sur une entrée de n bits (x_1, x_2, \dots, x_n) , calcule :

$$\mathcal{F}_{(A_i)}(x) = \sum_{i=1}^n x_i A_i \mod 2^m$$

But (rappel : c'est **NP-dur**)

Étant donné $y \leq 2^m$, trouver $x \in \{0, 1\}^n$ tel que $F(x) = y$

Question à 1000 points

► $n = 128$ (suffisamment grand)

► Est-ce dur quelle que soit la taille des A_i (m bits)?

Application #3



$$y = F(\mathbf{x}) = \sum_{i=1}^n x_i A_i \pmod{2^m}, \quad \text{donc :}$$

$$(k, x_1, x_2, \dots, x_n) \cdot \begin{pmatrix} 2^m & & & \\ A_1 & 2 & & \\ A_2 & & 2 & \\ \vdots & & & \ddots \\ A_n & & & & 2 \end{pmatrix} = (y, 2x_1, 2x_2, \dots, 2x_n)$$

- ▶ Le vecteur $(y, 2x_1, 2x_2, \dots, 2x_n)$ appartient au réseau engendré par les lignes de la matrice, et il est proche de $(y, 1, 1, \dots, 1)$
- ▶ La distance entre les deux est \sqrt{n} (très, très faible)
- ⇒ $(y, 2x_1, \dots, 2x_n)$ certainement le + proche de $(y, 1, \dots, 1)$
- ▶ Résoudre CVP dans le réseau $\rightsquigarrow x_1, x_2, \dots, x_n \dots$

Obstacle

Le réseau est de **grande dimension** ($n = 128$). CVP est infaisable.

Utilisation de CVP approché

Situation

- ▶ cible $\mathbf{y} = (y, 1, 1, \dots, 1)$
 - ▶ $\mathbf{x} = (y, 2x_1, \dots, 2x_n)$
 - ▶ $\|\mathbf{y} - \mathbf{x}\| = \sqrt{n}$
- ▶ $\begin{pmatrix} 2^m & & & \\ A_1 & 2 & & \\ \vdots & & \ddots & \\ A_n & & & 2 \end{pmatrix}$

Algorithmes pour APPROX-CVP $_{\gamma}$

- ▶ On sait qu'on récupère \mathbf{x}' t.q. $\|\mathbf{y} - \mathbf{x}'\| \leq \gamma \|\mathbf{y} - \mathbf{x}\| \leq \gamma \sqrt{n}$.
- ▶ Que faire avec \mathbf{x}' ?

Utilisation de CVP approché

Situation

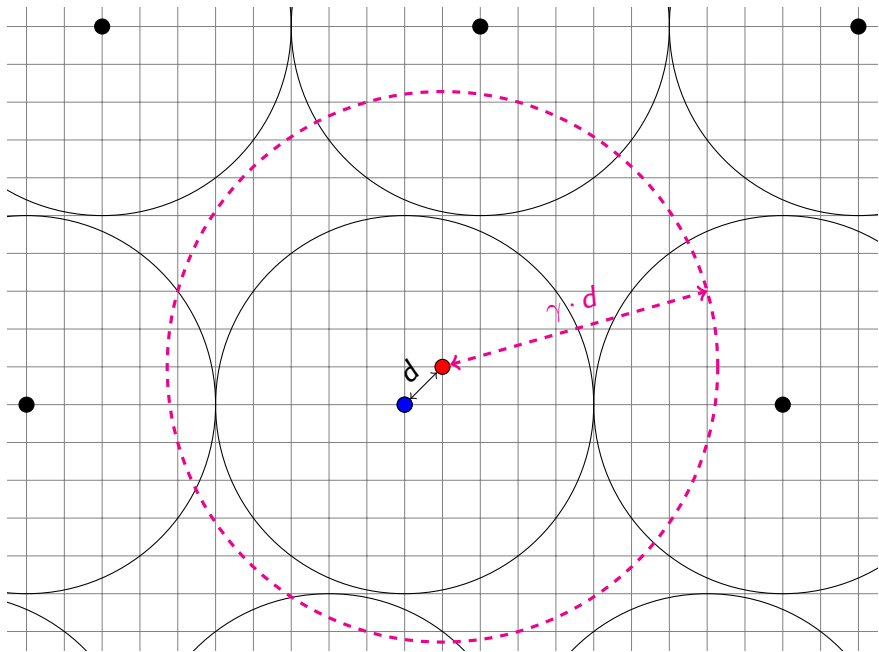
- ▶ cible $\mathbf{y} = (\mathbf{y}, 1, 1, \dots, 1)$
 - ▶ $\mathbf{x} = (\mathbf{y}, 2\mathbf{x}_1, \dots, 2\mathbf{x}_n)$
 - ▶ $\|\mathbf{y} - \mathbf{x}\| = \sqrt{n}$
- ▶ $\begin{pmatrix} 2^m & & & \\ A_1 & 2 & & \\ \vdots & & \ddots & \\ A_n & & & 2 \end{pmatrix}$

Algorithmes pour APPROX-CVP $_{\gamma}$

- ▶ On sait qu'on récupère \mathbf{x}' t.q. $\|\mathbf{y} - \mathbf{x}'\| \leq \gamma \|\mathbf{y} - \mathbf{x}\| \leq \gamma \sqrt{n}$.
- ▶ Que faire avec \mathbf{x}' ?

The power of approximation

- ▶ Si $\gamma \|\mathbf{y} - \mathbf{x}\| \ll \lambda_1(\mathcal{L})$, alors CVP $_{\gamma}(\mathbf{y})$ renvoie $\mathbf{x}' = \mathbf{x}$.
- ▶ Précisément, $(\gamma + 1) \|\mathbf{y} - \mathbf{x}\| < \lambda_1(\mathcal{L})$ suffit



Utilisation de CVP approché

Situation

- ▶ cible $\mathbf{y} = (\mathbf{y}, 1, \dots, 1)$
- ▶ $\mathbf{x} = (\mathbf{y}, \mathbf{x}_1, \dots, \mathbf{x}_n)$
- ▶ **Banco** si $\gamma\sqrt{n} \lll \lambda_1(\mathcal{L})$

$$\rightarrow \begin{pmatrix} 2^m & & & \\ A_1 & 2 & & \\ \vdots & & \ddots & \\ A_n & & & 2 \end{pmatrix}$$

Primo : plus court vecteur

- ▶ $\text{Vol } \mathcal{L} = 2^{m+n}$
- ▶ Heuristique gaussienne $\rightsquigarrow \lambda_1(\mathcal{L}) \approx 2^{(m+n)/(n+1)} \sqrt{n+1}$

Secundo : facteur d'approximation

Méthode en temps polynomial : $\gamma = 2^{cn}$.





$$\text{Banco} \iff \gamma\sqrt{n} \lll \lambda_1(\mathcal{L}) \iff cn^2 \lll m$$

Fonction à sens unique *subset sum*

I Génère n entiers (A_i) **aléatoires** sur m bits

✓ Sur une entrée de n bits (x_1, x_2, \dots, x_n) , calcule :

$$\mathcal{F}_{(A_i)}(x) = \sum_{i=1}^n x_i A_i \mod 2^m$$

- ▶  **cassé**  (en temps polynomial) si m **trop grand**
 - ▶ Dépend du facteur d'approximation pour (H)SVP
- ▶ Facteur d'approximation $\gamma = 2^{cn} \rightsquigarrow$  **cassé**  si $m \geq cn^2$
- ▶ En pratique, plongement + LLL $\rightsquigarrow \gamma \approx 1.0219^n$
 $\Rightarrow n = 128? \quad \longrightarrow \quad m \approx 512 \text{ bits}...$
- ▶ 128 entiers de ≈ 128 bits semble vraiment sûr

Plan

Résolution de programmes linéaires entiers

Cryptanalyse de RSA avec Petits exposants secrets

Cryptanalyse des générateurs linéaires congruentiels tronqués

Résolution de Subset-Sum

Chiffrement homomorphe sur les entiers

Algorithme de Coppersmith

Syntaxe du chiffrement symétrique

Trois algorithmes

1. **Paramètre de sécurité** $n \in \mathbb{N}$.
2. G (key **G**eneration) : $k \leftarrow G(1^n)$.
3. E (**E**ncryption) et D (**D**ecryption)

G forcément probabiliste. E généralement aussi.

Correction

Si $k \leftarrow G(1^n)$, alors pour tout $m \in \{0, 1\}^*$

$$D(k, E(k, m)) = m.$$

Sécurité? Plus tard

Chiffrement symétrique le plus simple possible

One-time Pad

$$E(k, m) = k \oplus m.$$

Problèmes bien connus (« one-time »).

Chiffrement symétrique le plus simple possible

One-time Pad

$$E(k, m) = k \oplus m.$$

Problèmes bien connus (« one-time »).

One-time Pad avec clef recyclable

- ▶ $G(1^n) = x \approx 2^n$ aléatoire
- ▶ $\mathcal{M} = \{0, 1\}^*$, $\mathcal{C} = \mathbb{Z}_x$.
- ▶ $E(x, m) = m + kx$ avec $k \approx 2^{\alpha n}$ aléatoire
- ▶ $D(x, c) = c \bmod x$

Idée sous-jacente : Masques = kx avec x fixé et k aléatoire.

Chiffrement symétrique le plus simple possible

One-time Pad avec clef recyclable

- ▶ $G(1^n) = x \approx 2^n$ aléatoire
- ▶ $E(x, m) = m + kx$ avec $k \approx 2^{\alpha n}$ aléatoire
- ▶ $D(x, c) = c \bmod x$

Problème #1 : si x est pair

$c \equiv m + 2k(x/2)$, donc $m \equiv c \bmod 2\ldots$ (on déchiffre sans la clef!)

Chiffrement symétrique le plus simple possible

One-time Pad avec clef recyclable

- ▶ $G(1^n) = x \approx 2^n$ aléatoire **impair**
- ▶ $E(x, m) = m + kx$ avec $k \approx 2^{\alpha n}$ aléatoire
- ▶ $D(x, c) = c \bmod x$

Problème #2 : factorisation

- ▶ Un chiffré : $c_1 = m_1 + k_1x$
- ▶ « Deviner » $m_1 \rightsquigarrow c_1 - m_1 = k_1x$.
- ▶ Factoriser k_1x ?

Chiffrement symétrique le plus simple possible

One-time Pad avec clef recyclable

- ▶ $G(1^n) = x \approx 2^n$ aléatoire **impair**
- ▶ $E(x, m) = m + kx$ avec $k \approx 2^{\alpha n}$ aléatoire
- ▶ $D(x, c) = c \bmod x$

Problème #2 : factorisation

- ▶ Un chiffré : $c_1 = m_1 + k_1x$
- ▶ « Deviner » $m_1 \rightsquigarrow c_1 - m_1 = k_1x$.
- ▶ Factoriser k_1x ? \rightsquigarrow Prendre $n > 1000$ et $\alpha \geq 1$ est suffisant.

Chiffrement symétrique le plus simple possible

One-time Pad avec clef recyclable

- ▶ $G(1^n) = x \approx 2^n$ aléatoire **impair**
- ▶ $E(x, m) = m + kx$ avec $k \approx 2^{\alpha n}$ aléatoire
- ▶ $D(x, c) = c \bmod x$

Problème #3 : PGCD

- ▶ Deux chiffrés : $c_1 = m_1 + k_1x$ et $c_2 = m_2 + k_2x$
- ▶ « Deviner » $m_1, m_2 \rightsquigarrow k_1x$ et k_2x
- ▶ $\text{PGCD}(k_1x, k_2x) \rightsquigarrow x...$

Chiffrement symétrique le plus simple possible

One-time Pad avec clef recyclable (2010, récent!)

- ▶ $G(1^n) = x \approx 2^n$ aléatoire impair
- ▶ $E(x, m) = m + 2r + kx$ avec $k \approx 2^{\alpha n}$, $r \approx 2^{\beta n}$ aléatoires
- ▶ $D(x, c) = (c \bmod x) \bmod 2$

Ajout d'un « bruit » r . Déchiffrement correct si $r < x/2$.

BONUS!

- ▶ $c_1 + c_2$ chiffré de $m_1 + m_2 \bmod 2$.
- ▶ $c_1 c_2$ chiffré de $m_1 m_2$.

+ Augmente la taille du « bruit » de 1 bit.

× Double la taille du « bruit ».

x grand et β petit \Rightarrow calculs (limités) sur les chiffrés!

Chiffrement symétrique le plus simple possible

One-time Pad avec clef recyclable (2010, récent!)

- ▶ $G(1^n) = x \approx 2^n$ (impair) aléatoire
- ▶ $E(x, m) = m + 2r + kx$ avec $k \approx 2^{\alpha n}$, $r \approx 2^{\beta n}$ aléatoires
- ▶ $D(x, c) = (c \bmod x) \bmod 2$

Exemple : $x \approx 1024$ bits, $k \approx 1024$ bits, $r \approx 128$ bits

Question

Est-ce sûr ? (et qu'est-ce que ça veut dire ?)

Chiffrement symétrique le plus simple possible

One-time Pad avec clef recyclable (2010, récent!)

- ▶ $G(1^n) = x \approx 2^n$ (impair) aléatoire
- ▶ $E(x, m) = m + 2r + kx$ avec $k \approx 2^{\alpha n}$, $r \approx 2^{\beta n}$ aléatoires
- ▶ $D(x, c) = (c \bmod x) \bmod 2$

Exemple : $x \approx 1024$ bits, $k \approx 1024$ bits, $r \approx 128$ bits

Question

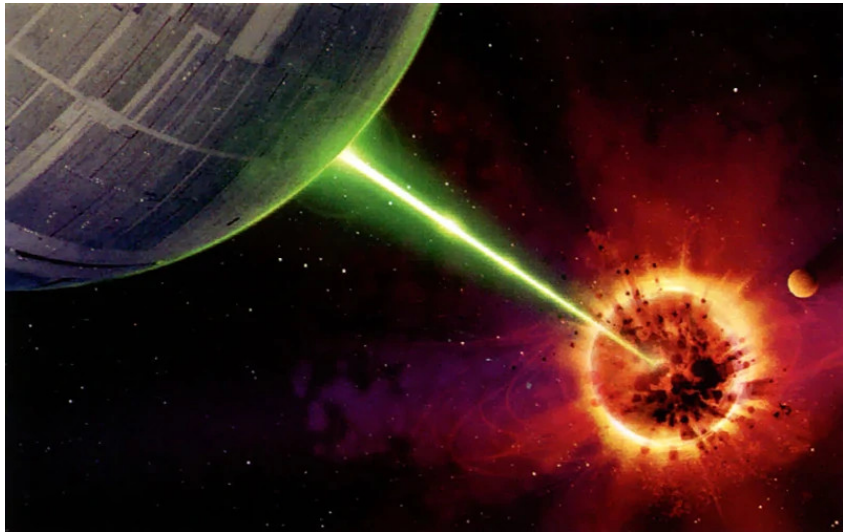
Est-ce sûr ? (et qu'est-ce que ça veut dire ?)

Problème sous-jacent : PGCD approché

Retrouver x à partir de multiples « bruités » de x .

- ▶ ENTRÉE : **multiples bruités** : $k_1x + r'_1, \dots, k_\ell x + r'_\ell$
bornes sur les tailles de k_i, r'_i, x .
- ▶ SORTIE : x

Application #4



Chiffrement homomorphe sur les entiers

Problème sous-jacent : PGCD approché

Retrouver x à partir de multiples « bruités » de x .

- ▶ ENTRÉE : **multiples bruités** $c_1 = k_1x + r_1, \dots, c_\ell = k_\ell x + r_\ell$
bornes $|x| < 2^n, |k_i| < X^\alpha, |r_i| < X^\beta$ (avec $\beta < 1$).
- ▶ SORTIE : x

Exemple : $x \approx 1024$ bits, $k_i \approx 1024$ bits, $r_i \approx 128$ bits

Retrouver les k (ou les r) permet de retrouver x

Stratégie

- ▶ « Bruit » petit : $\frac{c_i}{c_0} = \frac{k_i x + r_i}{k_0 x + r_0} \approx \frac{k_i}{k_0} \implies c_i k_0 \approx c_0 k_i$
- ▶ Donc $|c_i k_0 - c_0 k_i|$ est « petit »
- ▶ $|c_i k_0 - c_0 k_i| = |\cancel{x k_i k_0} + r_i k_0 - \cancel{k_0 k_i x} - r_0 k_i| < X^{\alpha+\beta}$
 - ▶ Comparer avec $|c_i k_0| \approx X^{1+2\alpha}$

Maintenant vous devez commencer à avoir l'habitude

► On pose :

$$\begin{aligned}\mathbf{x} &= (k_0, k_1, \dots, k_{\ell-1}) \begin{pmatrix} 2X^\beta & c_1 & c_2 & \dots & c_{\ell-1} \\ & -c_0 & & & \\ & & -c_0 & & \\ & & & \ddots & \\ & & & & -c_0 \end{pmatrix} \\ &= \left(2X^\beta k_0, k_0 r_1 - k_1 r_0, \dots, k_0 r_{\ell-1} - k_{\ell-1} r_0 \right)\end{aligned}$$

► $|c_i k_0 - c_0 k_i| < 2X^{\alpha+\beta}$, donc $\|\mathbf{x}\| < 2X^{\alpha+\beta} \sqrt{\ell}$

Stratégie habituelle #1

On aimerait récupérer \mathbf{x} en résolvant SVP dans le réseau engendré par les lignes de la matrice

Situation

► $\mathbf{x} = (2X^\beta k_0, \dots)$

► $\|\mathbf{x}\| < 2X^{\alpha+\beta} \sqrt{\ell}$

►
$$\begin{pmatrix} 2X^\beta & c_1 & \dots & c_{\ell-1} \\ & -c_0 & & \\ & & \ddots & \\ & & & -c_0 \end{pmatrix}$$

Plus court vecteur ?

► $\text{Vol } \mathcal{L} = 2X^\beta |c_0|^{\ell-1} \approx 2X^{\beta+(1+\alpha)(\ell-1)}$

► Heuristique gaussienne $\rightsquigarrow \lambda_1(\mathcal{L}) \approx X^{\beta/\ell+(1+\alpha)(\ell-1)/\ell} \sqrt{\ell}$

\mathbf{x} devrait être le plus court vecteur si...

$$X^{\alpha+\beta} \sqrt{\ell} \lll X^{\beta/\ell+(1+\alpha)(\ell-1)/\ell} \sqrt{\ell}$$

Qui se simplifie en $\ell > 1 + \alpha/(1 - \beta)$

Discussion

- ▶ $\mathbf{x} = (2X^\beta k_0, \dots)$ est le plus court vecteur du réseau si :

$$\ell > 1 + \alpha / (1 - \beta)$$

- ▶ Plus les multiples sont grands (α élevé), plus c'est dur
- ▶ Plus le bruit est important (β proche de 1), plus c'est dur

- ▶ ℓ petit (< 50) : on résout SVP exactement

- ▶ Paramètres proposés : $K, X \approx 1024$ bits, $R \approx 128$ bits

⇒ $\ell = 3$. **Cassé!**

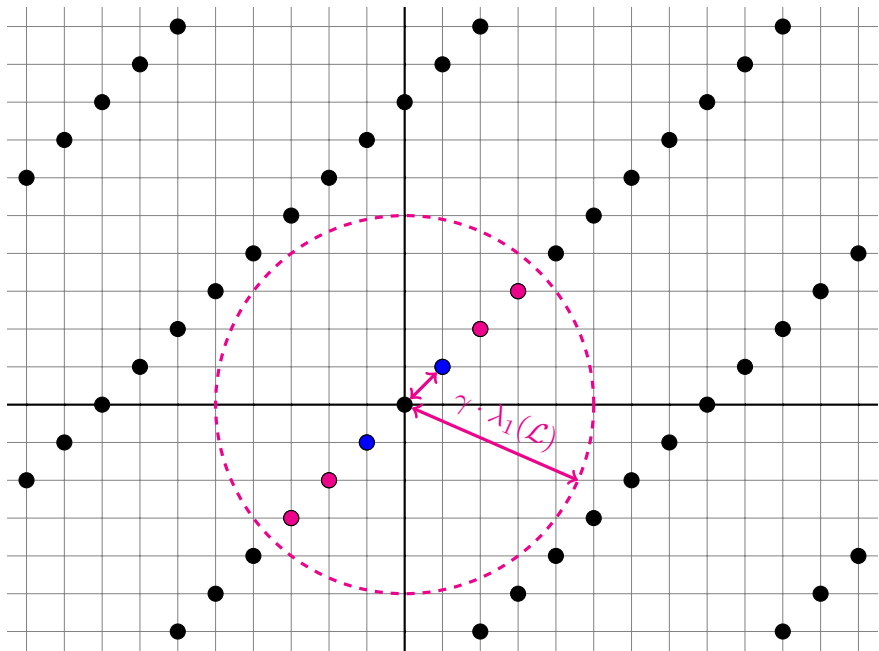
```
import fpylll                                # cf. https://github.com/fpylll/fpylll
def attack(C):                               # temps d'exécution : 0.6ms
    M = fpylll.IntegerMatrix(1, 1)
    M[0, 0] = 2*R
    for i in range(1, 1):
        M[0, i] = C[i]
        M[i, i] = -C[0]
    s = fpylll.SVP.shortest_vector(M)
    k0 = abs(s[0] // (2*R))
    return C[0] // k0
```

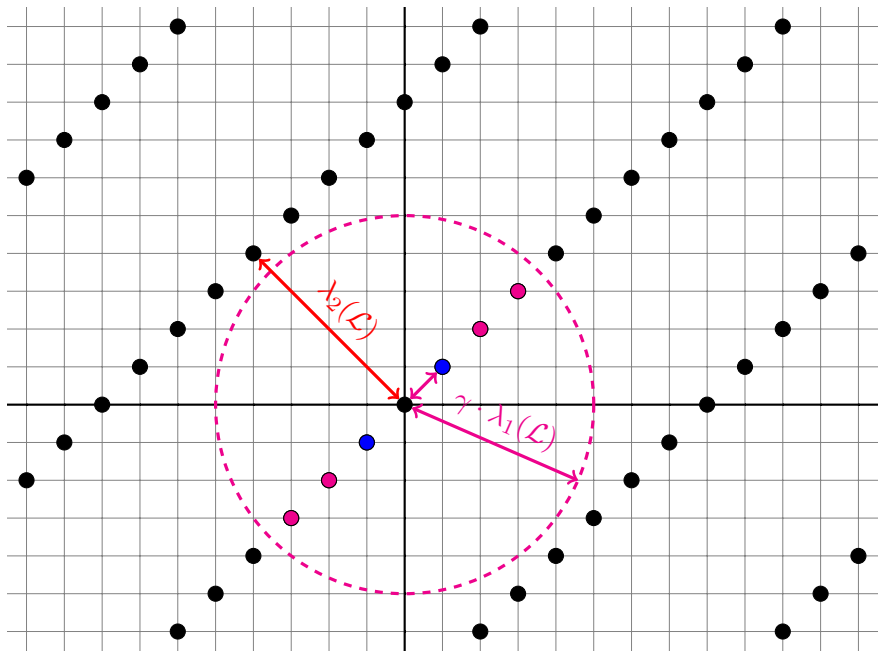
Discussion



Nouveaux paramètres (plus sûrs !)

- ▶ $x \approx 1024$ bits, $k_i \approx 65\,536$ bits, $r_i \approx 580$ bits
- ▶ $\ell > 200$. Très dur de résoudre SVP exactement. LLL ?





Situation

- ▶ $\mathbf{x} = (2X^\beta k_0, \dots)$, de norme $< 2X^{\alpha+\beta}$
- ▶ \mathbf{x} est le plus court vecteur du réseau si :

$$\ell > 1 + \alpha/(1 - \beta)$$

- ▶ $\text{Vol } \mathcal{L} \approx X^{\beta+(1+\alpha)(\ell-1)}$
- ▶ LLL permet d'approcher SVP avec facteur $\gamma = 2^{\ell/2}$
- ▶ LLL récupère (un multiple de) \mathbf{x} si $2^{\ell/2} \|\mathbf{x}\| < \lambda_2(\mathcal{L})$
- ▶ $\lambda_2(\mathcal{L})$ = norme du 2ème vecteur le plus court

Hypothèse : il y a **un** vecteur spécial, et c'est \mathbf{x}

Heuristique gaussienne : le **prochain** est de taille $\approx \text{Vol}(\mathcal{L})^{1/\ell} \sqrt{\ell}$

⇒ LLL récupère \mathbf{x} si $2^{\ell/2} X^{\alpha+\beta} \lll X^{\beta/\ell+(1+\alpha)(\ell-1)/\ell}$

Situation

- ▶ $\mathbf{x} = (2X^\beta k_0, \dots)$, de norme $< 2X^{\alpha+\beta}$
- ▶ \mathbf{x} est le plus court vecteur du réseau si $\ell > 1 + \alpha/(1 - \beta)$

On utilise LLL pour approcher SVP

- ▶ LLL récupère \mathbf{x} si $2^{\ell/2} X^{\alpha+\beta} \lll X^{\beta/\ell + (1+\alpha)(\ell-1)/\ell}$
- ▶ Ceci se simplifie en

$$\ell^2/2 - \ell n(1 - \beta) + n(1 - \beta + \alpha) < 0$$

Nouveau !

- ▶ ℓ trop grand = ça ne marche plus !
- ▶ Augmenter ℓ accroît l'écart entre $\lambda_1(\mathcal{L})$ et $\lambda_2(\mathcal{L})$
- ▶ Le facteur d'approximation pour SVP augmente plus vite...

Situation

- ▶ $\mathbf{x} = (2X^\beta k_0, \dots)$, de norme $< 2X^{\alpha+\beta}$
- ▶ \mathbf{x} est le plus court vecteur du réseau si $\ell > 1 + \alpha/(1 - \beta)$
- ▶ LLL retrouve \mathbf{x} si $\ell^2/2 - \ell n(1 - \beta) + n(1 - \beta + \alpha) < 0$

- ▶ Discriminant : $\Delta = n^2(1 - \beta)^2 - 2n(1 - \beta + \alpha)$
- ▶ Il faut que $\Delta > 0$ pour qu'une solution existe, donc

$$n > 2 \frac{1 - \beta + \alpha}{(1 - \beta)^2}$$

- ▶ Alors, on peut prendre $\ell = 1 + \frac{\alpha}{1-\beta} + \mathcal{O}(n^{-1})$

Discussion

- ▶ LLL casse le système si $n > 2(1 - \beta + \alpha)/(1 - \beta)^2$
- ▶ Réseau de dimension $\ell = 1 + \alpha/(1 - \beta)$
- ▶ Paramètres proposés : $x \approx 1024$ bits, $k_i \approx 65\,536$ bits, $r_i \approx 580$ bits

~> $\ell = 200$ et **condition satisfaite** \Rightarrow **Cassé?**

```
import fpylll                                # cf. https://github.com/fpylll/fpylll
def attack(C):                               # temps d'exécution : A LOT !
    M = fpylll.IntegerMatrix(1, 1)
    M[0, 0] = 2*R
    for i in range(1, 1):
        M[0, i] = C[i]
        M[i, i] = -C[0]
    fpylll.LLL.reduction(M)
    k0 = abs(s[0] // (2*R))
    return C[0] // k0
```

- ▶ Jeu : trouvez des paramètres pas cassés !

Plan

Résolution de programmes linéaires entiers

Cryptanalyse de RSA avec Petits exposants secrets

Cryptanalyse des générateurs linéaires congruentiels tronqués

Résolution de Subset-Sum

Chiffrement homomorphe sur les entiers

Algorithme de Coppersmith

Polynômes bivariés sur \mathbb{Z}

- ▶ INPUT : $P(X, Y)$ à coefficients dans \mathbb{Z} .
- ▶ SORTIE : liste de tous les $(x, y) \in \mathbb{Z}^2$ t.q. $P(x, y) = 0$
 - ▶ (ou \perp s'il y en a une infinité).

Polynômes univariés modulo N

- ▶ INPUT : N et $P(X)$ à coefficients modulo N .
- ▶ SORTIE : liste de tous les $x \in \mathbb{Z}_N$ t.q. $P(x) = 0 \bmod N$
 - ▶ (ou \perp s'il y en a une infinité).

Racines de polynômes

$$P(X) = X^d + a_{d-1}X^{d-1} + \cdots + x_2X^2 + a_1X + a_0$$

Racines de P ?

- ▶ **FACILE** sur $\mathbb{C}, \mathbb{R}, \mathbb{Q}, \mathbb{Z}$
- ▶ **FACILE** modulo p (premier)
- ▶ **DUR** modulo N (factorisation inconnue)

Mais...

Algorithme de Coppersmith (1996)

- ▶ Il existe un algorithme polynomial qui trouve les *petites* racines de P modulo N (sans connaître la factorisation de N)
- ▶ « petites » : taille $\approx N^{\frac{1}{d}}$

Certains cas sont faciles (RSA avec $e = 3$)

RSA « textbook » avec $e = 3$

- ▶ Messages stéréotypés (« votre code secret est ... »)
- ▶ Padding aléatoire affine
- ▶ ... bref, toutes les situation où seule une fraction $< 1/3$ du message clair est inconnue.

$$P(X) = (aX + b)^3 - C$$

- ▶ On retrouve tout en temps polynomial

$$P(X) = X^d + a_{d-1}X^{d-1} + \cdots + a_2X^2 + a_1X + a_0$$

1. Si P a de **petits** coefficients :

- ▶ Petite racine mod $N \rightsquigarrow$ petite racine sur \mathbb{Z} (facile à trouver)
- ▶ Le modulo n'agit pas
- ▶ équivalent à : $(a_0, a_1, \dots, a_{d-1})$ est un vecteur de **norme faible**

2. Sinon, on fabrique un **nouveau** polynôme $Q(X)$ t.q.

- ▶ $P(x) = 0 \bmod N \iff Q(x) = 0$ sur \mathbb{Z} pour tout $|x| \leq \text{Borne}$
- ▶ (on cherche les racines de Q sur \mathbb{Z} et c'est réglé)
- ▶ Comment fabriquer Q ?
 - a) Augmenter le modulo
 - b) Utiliser **LLL**

Step 1 : Augmenter le modulo

$$P(X) = X^d + a_{d-1}X^{d-1} + \cdots + x_2X^2 + a_1X + a_0$$

On fixe (par la méthode de la pensée magique) un entier k .

Si $P(x) \equiv 0 \pmod{N}$, alors

► $N^i \cdot x^j \cdot P(x)^{k-i} \equiv 0 \pmod{N^k} \quad (i=0, \dots, k, j=0, \dots, d-1)$

► Autrement dit :

$$\begin{array}{cccccc} N^k, & N^k X, & N^k, & \dots, & N^k X^{d-1}, \\ N^{k-1} P, & N^{k-1} X P, & N^{k-1} X^2 P, & \dots, & N^{k-1} X^{d-1} P, \\ N^{k-2} P^2, & N^{k-2} X P^2, & N^{k-2} X^2 P^2, & \dots, & N^{k-2} X^{d-1} P^2, \\ \vdots & & & & \\ N P^{k-1}, & N X P^{k-1}, & N X^2 P^{k-1}, & \dots, & N X^{d-1} P^{k-1} \end{array}$$

S'annulent sur x modulo N^k

Step 2 : Lemme de Howgrave-Graham

Petits coefficients + petite racine mod N^k = racine sur \mathbb{Z}

Theorem (Howgrave-Graham, 1997)

$g(X) = \sum_i c_i X^i$ possède n monômes. B est une borne. Si :

1. $g(x_0) \equiv 0 \pmod{N^k}$ et $|x_0| \leq B$
2. $\|g(BX)\| < N^k / \sqrt{n}$ (norme du vecteur formé par les coefficients du polynome)

Alors $g(x_0) = 0$ (sur \mathbb{Z}).

Démonstration.

$g(BX) = \sum_i c_i B^i X^i$ donc $\|g(BX)\| = \|(c_0, c_1 B, c_2 B^2, \dots)\|$.

$$|g(x_0)| = \left| \sum_i c_i x_0^i \right| \leq \sum_i |c_i x_0^i| \leq \sum_i |c_i| B^i \leq \sqrt{n} \|g(BX)\| < N^k$$

Mais $g(x_0)$ est un multiple de N^k , donc c'est zéro...



Equivalence des normes

- ▶ $\|x\|_1 = |x_1| + \dots + |x_n|$
- ▶ $\|x\|_2 = \sqrt{x_1^2 + \dots + x_n^2}$

On a :

$$\|x\|_1 \leq \sqrt{n} \|x\|_2$$

Démonstration.

- ▶ Il faut prouver : $|x_1| + \dots + |x_n| \leq \sqrt{n} \sqrt{x_1^2 + \dots + x_n^2}$.
- ▶ Conséquence directe de Cauchy-Schwartz 😄.

Equivalence des normes

- ▶ $\|x\|_1 = |x_1| + \dots + |x_n|$
- ▶ $\|x\|_2 = \sqrt{x_1^2 + \dots + x_n^2}$

On a :

$$\|x\|_1 \leq \sqrt{n} \|x\|_2$$

Démonstration.

- ▶ Il faut prouver : $|x_1| + \dots + |x_n| \leq \sqrt{n} \sqrt{x_1^2 + \dots + x_n^2}$.
- ▶ Conséquence directe de Cauchy-Schwartz 😊.
- ▶ Inégalité de Cauchy-Schwartz : $|\mathbf{u} \cdot \mathbf{v}| \leq \|\mathbf{u}\| \cdot \|\mathbf{v}\|$.
- ▶ Dans notre cas $\mathbf{u} = (1, 1, \dots, 1)$ et $\mathbf{v} = (|x_1|, \dots, |x_n|)$.
- ▶ Borne atteinte par $x = (1/n, 1/n, \dots, 1/n)$.



Step 3 : Former un polynôme à petits coefficients

On cherche les racines de P de taille $\leq B$.

- ▶ On prend nos polynômes $Q_{ij} = N^i \cdot x^j \cdot P(x)^{k-i} \equiv 0 \pmod{N^k}$
- ▶ On écrit les vecteurs des coefficients de $Q_{ij}(BX)$
 - ▶ Dans la base $1, X, X^2, X^3, \dots$
 - ▶ Les coefficients sur la colonne de X^i sont des multiples de B^i .
- ▶ On lance LLL sur le réseau qu'ils engendrent
- ▶ Ca donne un vecteur de norme faible, dont on peut déduire un polynôme à petits coefficients qui s'annule sur $x \pmod{N^k}$.