Cryptography with One-Way Functions

Plan

Fonctions à sens unique

Générateurs Pseudo-Aléatoires

Fonctions pseudo-aléatoires

Plan

Fonctions à sens unique

Générateurs Pseudo-Aléatoires

Fonctions pseudo-aléatoires

« *Facile à évaluer, difficile à inverser* »

Exemples

- $(x, y) \mapsto x \times y$
- $x \mapsto x^e \bmod N$ avec $N = pq$ (factorisation inconnue)
- $x \mapsto g^x \bmod p$ (p premier)
- $x \mapsto \mathtt{SHA256}(x)$

Facile ? Difficile ?

- Évaluation « efficace » $\rightsquigarrow$ polynomiale
- Inversion « difficile » $\rightsquigarrow$ exponentielle
- ... en fonction de quel paramètre ?

Première définition

$f : \{0,1\}^* \to \{0,1\}^*$ est une fonction **à sens unique** si

- Il existe $\mathcal{A}$ (polynomial déterministe) tel que $\mathcal{A}(x) = f(x)$.
- Pour tout algorithme polynomial randomisé $\mathcal{B}$

$$\Pr\left[x \xleftarrow{\$} \{0,1\}^n, y \leftarrow f(x), f(\mathcal{B}(y, 1^n)) = y\right] \quad \text{est négligeable}$$

Remarques :

- $\mathcal{B}$ reçoit $1^n = \underbrace{1111\ldots111}_{n \text{ fois}}$ pour avoir un temps d'exécution polynomial en n (même si y est plus petit)

Existe-t-il des fonctions à sens unique ?

Theorem

Soit f une fonction à sens unique.

Existe-t-il des fonctions à sens unique ?

Theorem

Soit f une fonction à sens unique. Alors $\mathbf{P} \neq \mathbf{NP}$. 🤯

Existe-t-il des fonctions à sens unique ?

Theorem

Soit f une fonction à sens unique. Alors $\mathbf{P} \neq \mathbf{NP}$. 🤯

Démonstration.

- On prouve $\mathbf{P} = \mathbf{NP} \Longrightarrow f$ n'est pas à sens unique
- $L = \{(1^n, x_0, y) : \exists x. |x| = n, f(x) = y, x_0$ est un préfixe de $x\}$
- $L \in \mathbf{NP}$ (x fait office de témoin)
- Donc il existe $\mathcal{D}$ qui décide l'appartenance à L en tps poly.
- Pour inverser $f(y)$:
  - $x_0 \leftarrow \epsilon$ (invariant : x_0 est le préfixe d'une préimage)
  - **for** $i = 1, \ldots, n$:
    - **if** $\mathcal{D}(1^n, x_0 \| 0, y) = 1$ **then** $x_0 \leftarrow x_0 \| 0$ **else** $x_0 \leftarrow x_0 \| 1$
  - **return** x_0

□

Prédicat *hard-core*

« Prédicat de x impossible à calculer à partir de $f(x)$ ».

Définition

Un prédicat $b : \{0,1\}^* \to \{0,1\}$ calculable en temps polynomial est **hard-core** pour f si pour tout algorithme polynomial $\mathcal{C}$,

$$\underbrace{\left| \Pr\left[x \xleftarrow{\$} \{0,1\}^n, \mathcal{C}(f(x)) = b(x) \right] - \frac{1}{2} \right|}_{\text{avantage de } \mathcal{C}} \quad \text{est négligeable}$$

Exemples

- Bit de poids faible pour fonction RSA, fonction de Rabin
- Bit de poids fort pour exponentielle mod p
- ...

Il y a toujours des prédicats *hard-core*

Theorem (Goldreich-Levin 1989)

Soit f une fonction à sens unique. Alors :

- $g(x, r) = (f(x), r)$ *est une fonction à sens unique*
- *Le produit scalaire* $\langle x, r \rangle \pmod 2$ *est hard-core pour* g

Éléments de preuve

- Inverser g permet d'inverser f
  - f à sens unique $\Longrightarrow g$ à sens unique
- But : b pas *hard-core* $\Longrightarrow g$ pas à sens unique
- Cas facile : $\mathcal{C}(f(x), r)$ renvoie $\langle x, r \rangle$ *avec probabilité 1*
  - $\mathcal{C}(f(x), e_i)$ renvoie le i-ème bit de x 😆
- Cas général (assez difficile) :
  - $\mathcal{C}$ peut répondre au hasard sur une bonne partie des r
  - $\mathcal{C}$ peut se tromper avec proba proche de $1/2$ sur les autres

Fonctions à sens-unique : pas seulement des chaînes de bits

- Familles de fonction (on peut choisir N, p, g, ...)
- $\mathcal{F} = \{\mathcal{F}_i : D_i \to \{0,1\}^*\}$,
  - $i =$ indice de la fonction
  - paramètres nécessaires à l'évaluation
  - « clef » de la fonction (généralement publique)
- D_i = domaine de la fonction

Syntaxe d'une famille de fonctions $\mathcal{F}$:

Trois algorithmes polynomiaux probabilistes :

- I (**I**ndexation) : $i \leftarrow I(1^n)$.
- D (échantillonnage) : $D(i) \in D_i$.
- V (é**V**aluation) : $\mathcal{F}_i(x) = V(i, x)$.

I tire au hasard une fonction de $\mathcal{F}$ de « paramètre de sécurité » n.
Le domaine D_i de $\mathcal{F}_i$ dépend de i. Taille $\approx 2^n$.
Pas de trappe (=inversion efficace en connaissant un secret).

Fonctions à sens-unique : sécurité

Définition avec un « jeu »

1. Le challenger génère un indice $i \leftarrow I(1^n)$ ainsi qu'un élément $x \leftarrow D(i)$ dans le domaine de $\mathcal{F}_i$. Il calcule $y \leftarrow V(i,x)$
2. Le challenger transmet (i,y) à l'adversaire
3. L'adversaire renvoie $\hat{x}$ et il gagne si $V(i,\hat{x}) = y$

Sécurité (asymptotique)

La famille $\mathcal{F}$ est **à sens unique** (*One-Way*) si tout adversaire polynomial n'a qu'une probabilité de succès négligeable (= plus petit que l'inverse de n'importe quel polynôme en n).

Sécurité (concrète)

La famille $\mathcal{F}$ est (T,ϵ)-**à sens unique** (*One-Way*) si tout adversaire qui s'exécute en temps T n'a qu'une probabilité de succès inférieure) ϵ.

Le sac à dos (subset-sum)

Fonction à sens unique *subset sum*

- *I* Génère n entiers (N_i) aléatoires sur m bits
- *V* Sur une entrée de n bits $(x_1, x_2, \ldots, x_n)$, calcule :

$$\mathcal{F}_{(N_i)}(x) = \sum_{i=1}^{n} x_i N_i \pmod{M, \text{en option}}$$

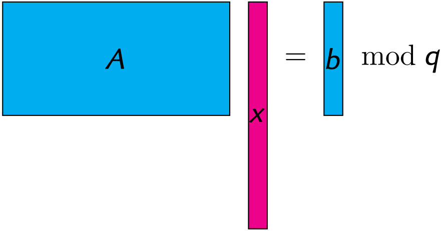
Très tentant !

- Simple, facile à comprendre et à programmer
- Les problèmes suivants sont **NP-durs** :
  - Inversion : à partir des (N_i) et t, trouver x tel que $F(x) = t$
  - Collision : à partir des (N_i), trouver $x \neq y$ tels que $F(x) = F(y)$

Le Produit matrice-vecteur est à sens unique 😲

Fonction très simple (Ajtai, 1996)

Produit matrice-vecteur : ${\mathbb{Z}_q}^n \to {\mathbb{Z}_q}^m$ modulo q

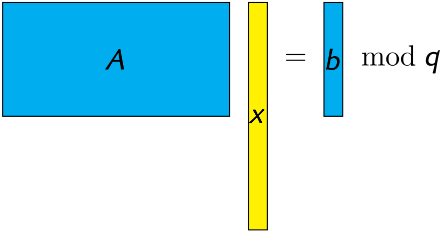


- Avec (A, b), facile de retrouver un x qui satisfait l'équation

Le Produit matrice-vecteur est à sens unique

Fonction très simple (Ajtai, 1996)

Produit matrice-vecteur : $\{0,1\}^n \to \mathbb{Z}_q{}^m$ modulo q

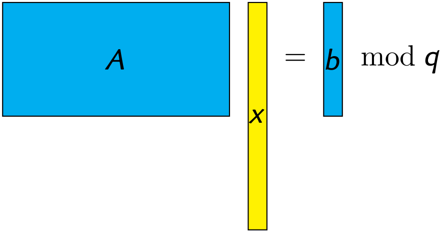


- Si $\mathcal{A}$ trouve x en temps. poly, alors $P = NP$
  - $A =$ matrice aléatoire modulo q
  - x a de petits coefficients ($x_i \in \{0,1\}$)
  - $q = n^\alpha$ (pour un certain α)
  - $m = \beta n \log n$ (pour un certain β)
- A aléatoire = problème aussi dur qu'avec A arbitraire
  - Ceci est vraiment très fort

Le Produit matrice-vecteur est à sens unique

Fonction très simple (Ajtai, 1996)

Produit matrice-vecteur : $\{0,1\}^n \to \mathbb{Z}_q{}^m$ modulo q



- Si A est aléatoire, la fonction :

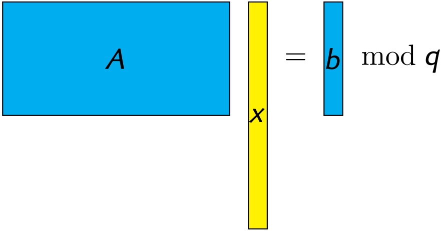
$$\begin{array}{rcl} \{0,1\}^n & \to & (\mathbb{Z}_q)^m \\ x & \mapsto & Ax \end{array}$$

est à sens unique (pour m et q de la bonne taille)

Le Produit matrice-vecteur est à sens unique 😲

Fonction très simple (Ajtai, 1996)

Produit matrice-vecteur : $\{-1, 0, 1\}^n \to \mathbb{Z}_q{}^m$ modulo q



- Si A est aléatoire, la fonction :

$$\begin{array}{rcl} \{0,1\}^n & \to & (\mathbb{Z}_q)^m \\ x & \mapsto & Ax \end{array}$$

  est à sens unique (pour m et q de la bonne taille)
- Compresse m bits en $n \log_2 q$ bits. Résistance aux collisions !

Plan

Fonctions à sens unique

Générateurs Pseudo-Aléatoires

Fonctions pseudo-aléatoires

Norme POSIX : spécification (obligatoire) de mrand48()

```
uint64_t rand48_state;

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

uint32_t mrand48() { /* 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);
}
```

Math.rand() dans JavaScript (Chrome, Firefox, Safari)

```
uint64_t 64 a, b;

uint64_t xorshift128plus()  /* renvoie 64 bits aléatoires */
{
    uint64_t s1 = a
    uint64_t s0 = b;
    a = s0;
    s1 ^= s1 << 23;
    s1 ^= s1 >> 17;
    s1 ^= s0;
    s1 ^= s0 >> 26;
    b = s1;
    return a + b;
}
```

Générateurs **pseudo**-aléatoires

- Pas du « vrai hasard », mais résultat d'un calcul.
- Très utile (génération de clefs cryptographiques...)

Générateurs conguentiels linéaires

Histoire

Inventés par Derrick Lehmer (1905–1991) pour utilisation sur l'ENIAC ! Proposition de 1949 :

$$u_0 = 47\,594\,118,$$
$$u_{i+1} = 23u_i \bmod 10^8 + 1$$

- Utiliser X_i comme source de bits « aléatoires » avec

$$X_{i+1} = aX_i + b \bmod m$$

- X_0 est la graine

Question

Est-il raisonnable d'utiliser ça pour générer des cartes bleues ?

Un **générateur pseudo-aléatoire** G est un algorithme déterministe qui produit une *longue* séquence de bits pseudo-aléatoires, de taille t, à partir d'une *courte* graine.

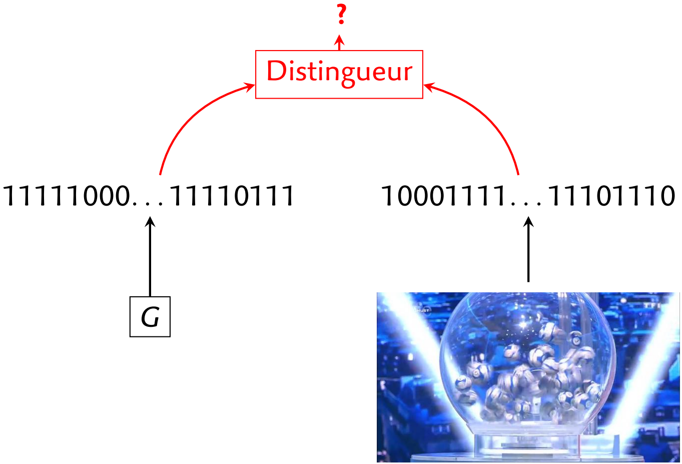
Informellement

séquence de bits « Pseudo-aléatoire » = **indistinguables** de bits vraiment aléatoires par un algorithme **polynomial**.

Distingueur

Algorithme $\mathcal{A}$ polynomial qui tente de faire la différence entre la sortie du PRG et des bits vraiment aléatoires.

- Par ex. $\mathcal{A}(x) = 1$ si x vient du PRG, et 0 sinon


?
Distingueur
11111000...11110111
10001111...11101110
G

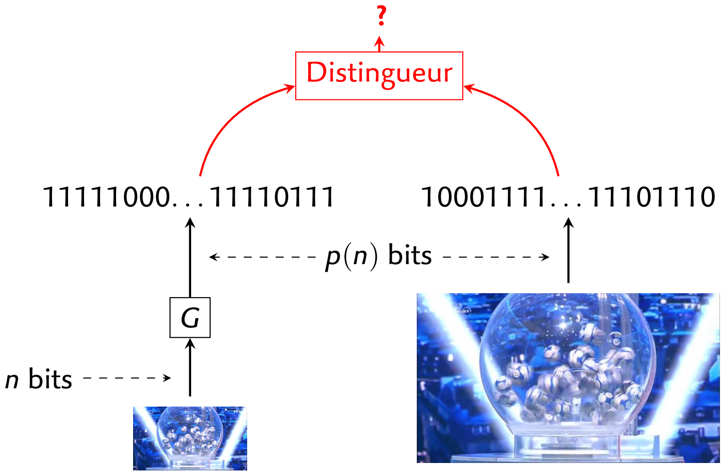
?

Distingueur

$11111000\ldots11110111$ $\qquad$ $10001111\ldots11101110$

$\boxed{G}$

42



En fait le distingueur doit faire la différence entre deux distributions sur les chaines de bits : la distribution uniforme d'un côté et la sortie de G de l'autre (avec graine aléatoire).

Indistinguabilité calculatoire

- Si X_n et Y_n sont deux variables aléatoires dans $\{0,1\}^n$
  (c.a.d. des chaines de bits tirées selon deux distributions différentes),
  L'avantage de l'algorithme $\mathcal{A}$ pour les distinguer est :

$$\mathbf{Adv}(\mathcal{A}) = \left|\Pr\left(\mathcal{A}(X_n) = 1\right) - \Pr\left(\mathcal{A}(Y_n) = 1\right)\right|$$

- Deux suites de variables aléatoires $\{X_n\}_{n\geq 0}$ et $\{Y_n\}_{n\geq 0}$ sont **calculatoirement indistinguables** si tout distingueur en temps polynomial (en n) n'a qu'un avantage négligeable (plus petit que l'inverse de n'importe quel polynôme en n)

Pseudo-aléatoire

- La suite $\{X_n\}_{n\geq 0}$ est **pseudo-aléatoire** si elle est calculatoirement indistinguable de la suite uniforme $\{U_n\}_{n\geq 0}$
  U_n est uniformément distribuée sur $\{0,1\}^n$

Diffie-Hellman Décisionnel

- $\left(g, h, g^{X_n}, h^{X_n}\right) \overset{?}{\longleftrightarrow} \left(g, h, g^{X_n}, h^{Y_n}\right)$

Subset Sum (Impagliazzo & Naor, 1996)

- $\left(A_1, \ldots, A_n, \sum_{i=1}^{n} x_i A_i \bmod 2^\ell\right) \overset{?}{\longleftrightarrow} (A_1, \ldots, A_n, y)$
  - $\ell \leq 1.06n$ (pour être sûr que le subset-sum est one-way)
  - x_i : uniformément aléatoire dans $\{0, 1\}$
  - A_i, y : uniformément aléatoires dans $\mathbb{Z}_{2^\ell}$

N'importe quelle fonction OW (Goldreich-Levin)

- f une fonction à sens unique, b un prédicat *hard-core* pour f
- $(f(x), b(x)) \overset{?}{\longleftrightarrow} (f(x), y)$
  - x : uniformément aléatoire dans $\{0, 1\}^n$
  - y : uniformément aléatoire dans $\{0, 1\}$.

Définition

Un **générateur pseudo-aléatoire** est un algorithme déterministe G qui satisfait les deux conditions :

1. *Expansion* : il y a une fonction $\ell : \mathbb{N} \to \mathbb{N}$ avec $\ell(n) > n$ et $|G(s)| = \ell(|s|)$.
2. *Pseudo-aléa* : la suite $\{G(U_n)\}_{n \geq 0}$ est pseudo-aléatoire.

$\ell(n)$ est le « facteur d'étirement » (*stretch*)

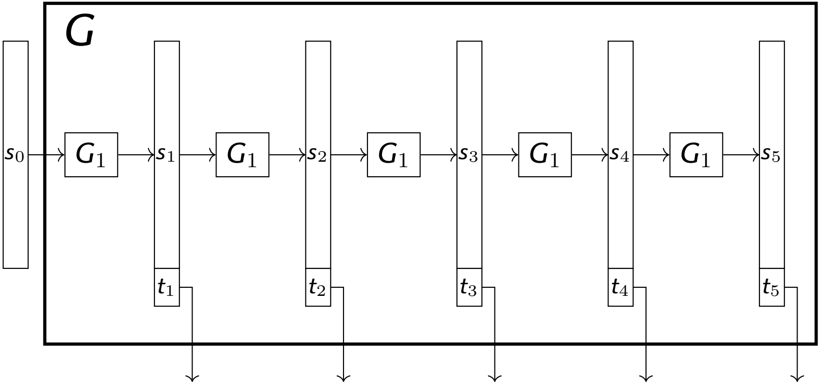
Exemples

- Si g engendre un groupe cyclique et $h \in \langle g \rangle$, alors

$$G(x) = (g^x, h^x) \quad \text{est un PRG} \quad \textbf{si DDH est dur}$$

- Si f est une **bijection** à sens unique et b un prédicat *hard-core*, alors $G(x) = (f(x), b(x))$ est un PRG.

Comment augmenter le facteur d'étirement ?



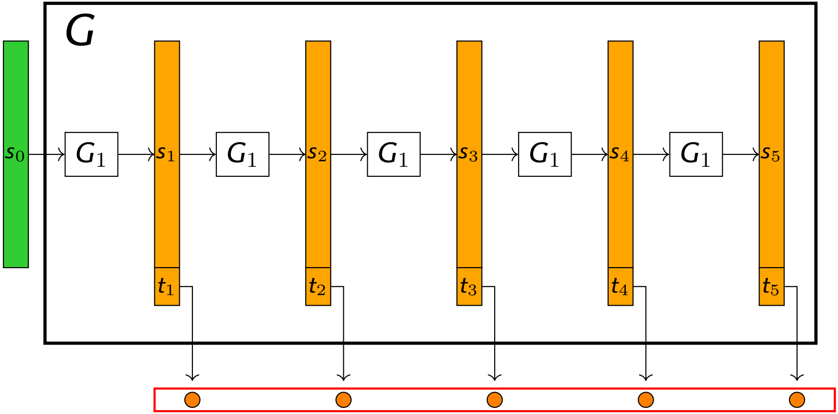
$$G_1(s_i) = t_{i+1} \parallel s_{i+1}$$

Theorem

Si G_1 est un PRG (avec étirement $n+1$), alors G est un PRG.

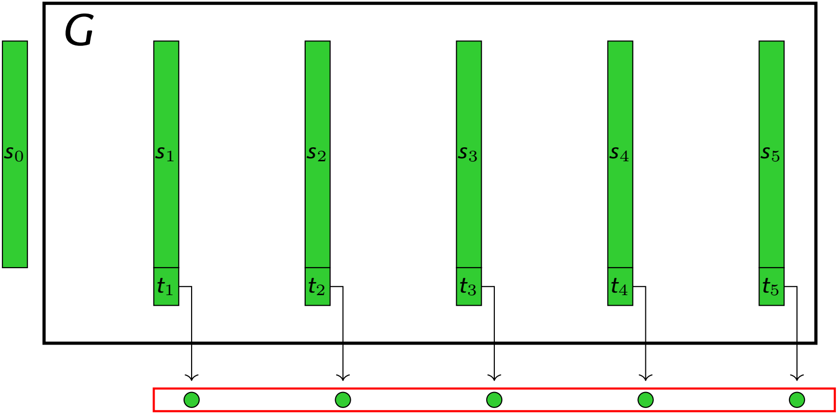
- On prouve : G n'est pas un PRG $\Longrightarrow$ G_1 n'est pas un PRG

Preuve : l'argument hybride



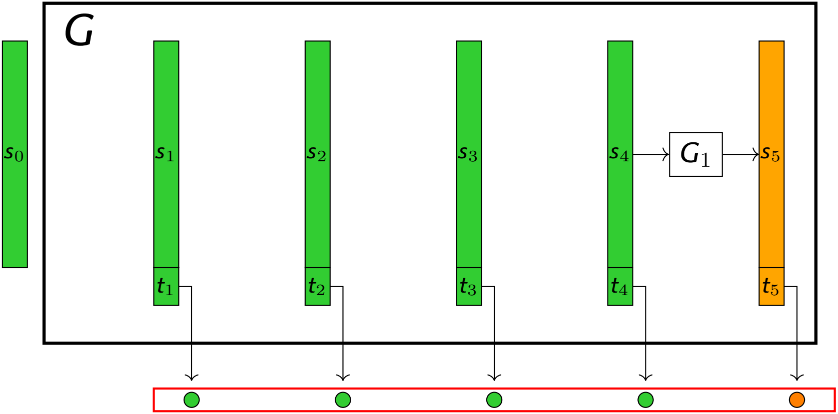
avantage non-negligeable

Preuve : l'argument hybride



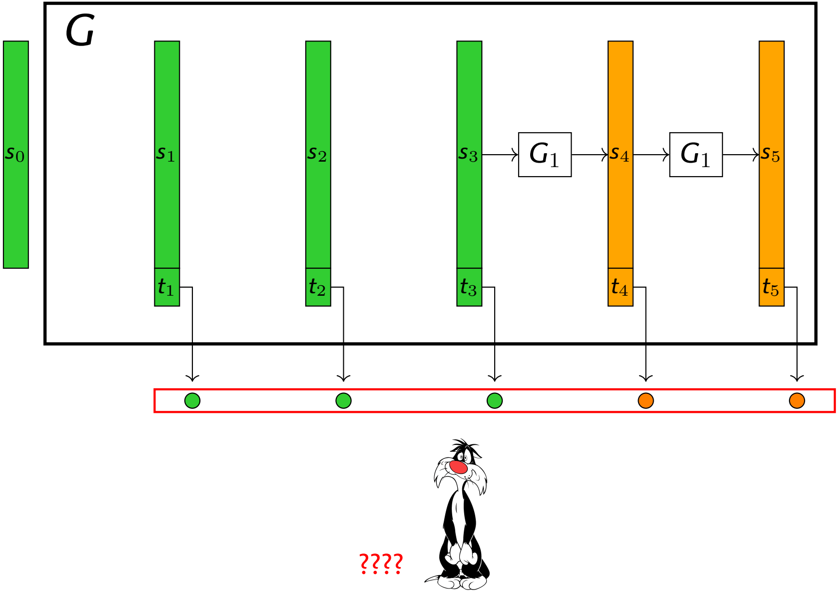
avantage nul

Preuve : l'argument hybride

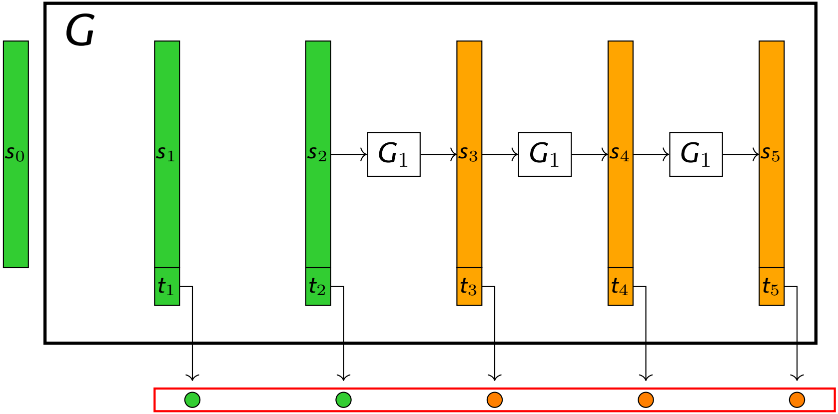


????

Preuve : l'argument hybride

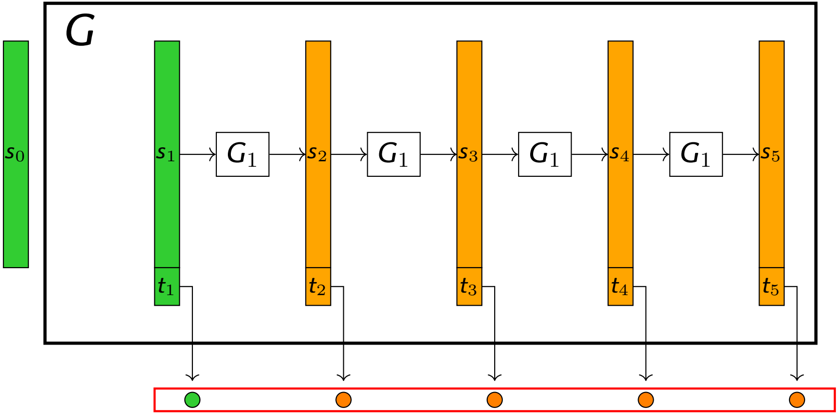


Preuve : l'argument hybride



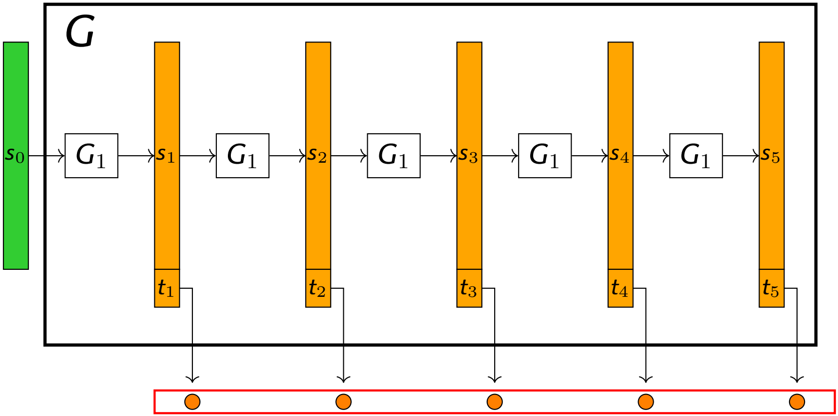
????

Preuve : l'argument hybride



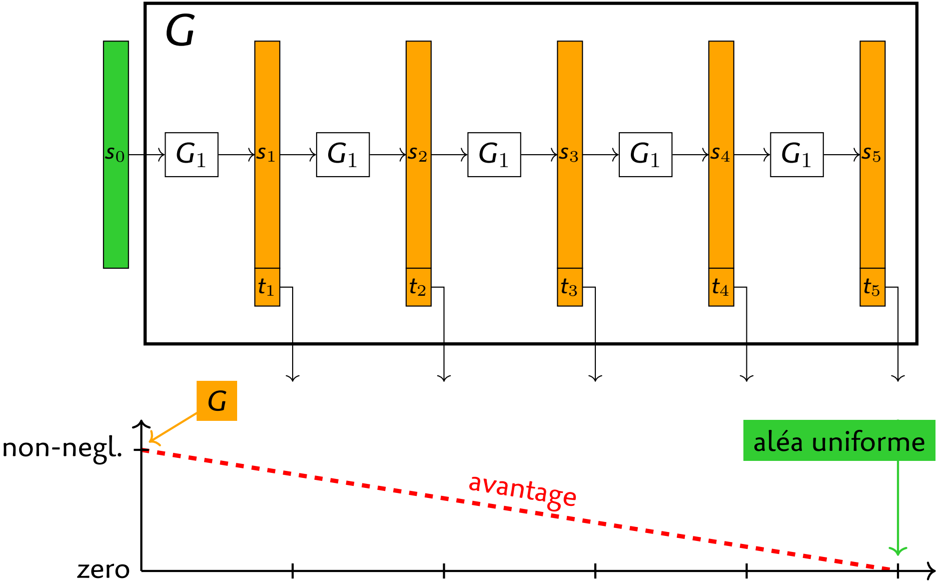
????

Preuve : l'argument hybride

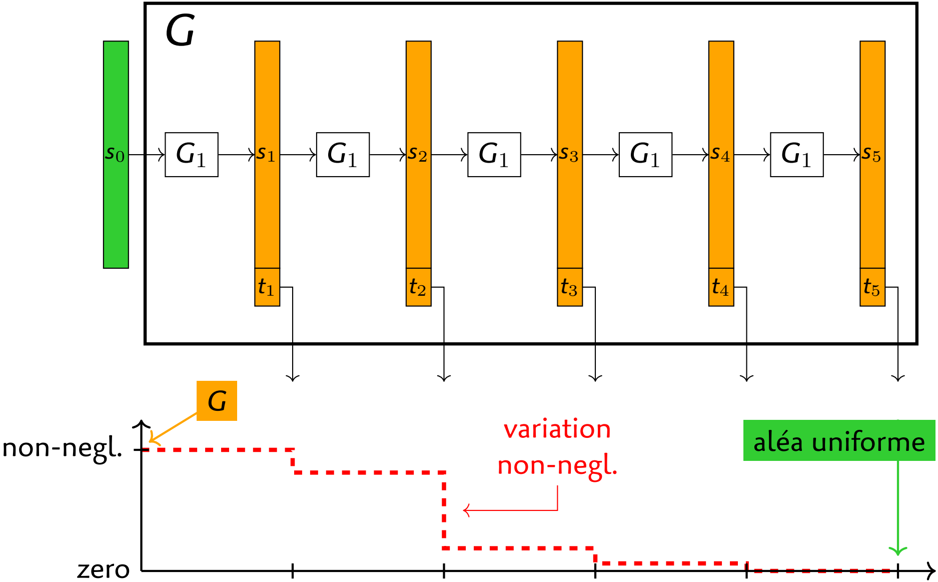


avantage non-negligeable

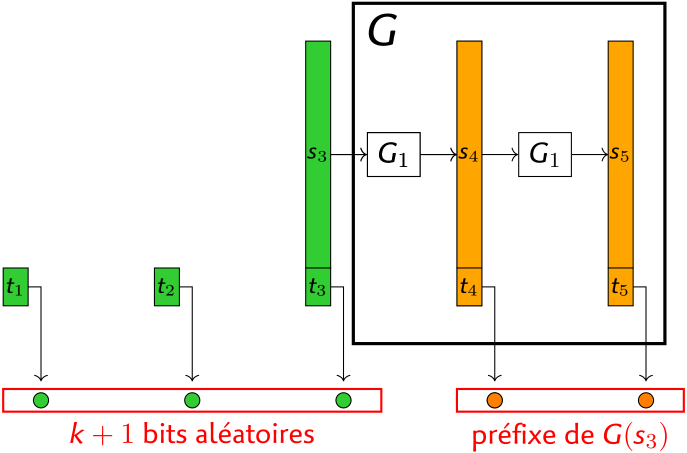
Preuve : l'argument hybride



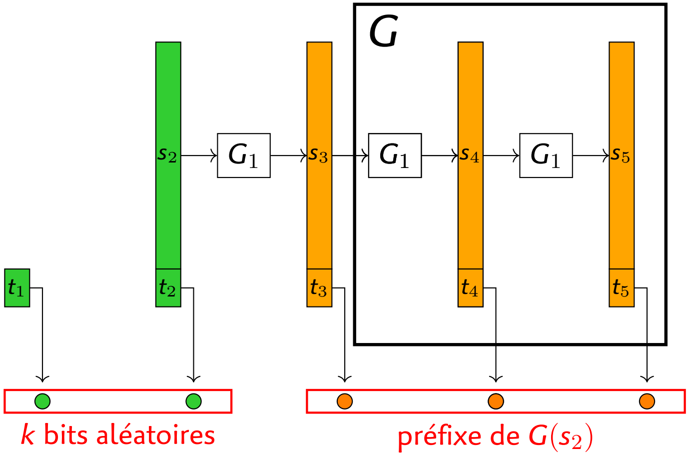
Preuve : l'argument hybride



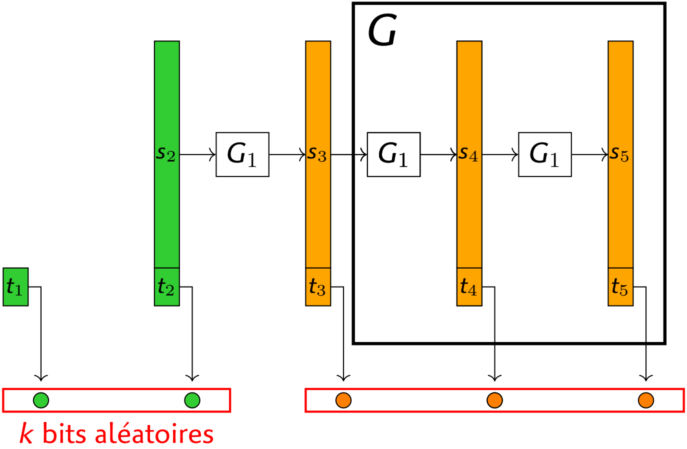
Preuve : l'argument hybride (suite)



Preuve : l'argument hybride (suite)



Preuve : l'argument hybride (suite)



Le distingueur a un comportement différent selon que (s_3, t_3) est aléatoire ou issu de G_1

Preuve : l'argument hybride (fin)

- On part d'un distingueur $\mathcal{D}$ pour G avec avantage non-negl.

Distingueur $\mathcal{E}$ pour G_1

- Entrée : $y \in \{0,1\}^{n+1}$
  - Ou bien $y \xleftarrow{\$} \{0,1\}^{n+1}$
  - Ou bien $y \leftarrow G_1(s)$ avec $s \xleftarrow{\$} \{0,1\}^n$
- Deviner k $(0 \leq k \leq |G(s)|)$
- $z_k \leftarrow [k \text{ bits aléatoires }] \parallel y[0] \parallel G(y[1:n+1])$
- $\hat{b} \leftarrow \mathcal{D}(z_k)$
- Renvoyer $\hat{b}$

Performance

[avantage de $\mathcal{E}$] $\geq$ [avantage de $\mathcal{D}$]/[taille de $G(s)$]

Instantiations

Avec $f : x \mapsto g^x \bmod p$

- g racine primitive $\Longrightarrow$ f bijection sur $\mathbb{Z}_p^\times$
- $\mathsf{MSB}(x) = 0$ si $x < (p-1)/2$ et 1 si $(p-1)/2 \leq x$
- MSB est *hard-core* pour f (cf. TD)

$\rightsquigarrow$ $G(x) := (g^x \bmod p, \mathsf{MSB}(x))$ est un PRG avec étirement $n+1$

$\rightsquigarrow$ La séquence suivante est pseudo-aléatoire

$$\mathsf{MSB}(x), \mathsf{MSB}(g^x \bmod p), \mathsf{MSB}(g^{g^x \bmod p} \bmod p), \ldots$$

Ce PRG est sûr sous l'hypothèse que DLOG est difficile

Instantiations

Avec $f : x \mapsto x^2 \bmod N$

- N entier de Blum ($N = pq$ et $p, q \equiv 3 \bmod 4$)
- f permutation des résidus quadratiques modulo N
- $\mathsf{LSB}(x) = x \bmod 2$
- LSB est *hard-core* pour f

$\rightsquigarrow$ $G(x) := (x^2 \bmod N, \mathsf{LSB}(x))$ est un PRG avec étirement $n+1$

$\rightsquigarrow$ La séquence suivante est pseudo-aléatoire

$$\mathsf{LSB}(x), \mathsf{LSB}(x^2 \bmod N), \mathsf{LSB}(x^4 \bmod N), \mathsf{LSB}(x^8 \bmod N) \ldots$$

Ce PRG est sûr sous l'hypothèse que la factorisation est difficile

Plan

Fonctions à sens unique

Générateurs Pseudo-Aléatoires

Fonctions pseudo-aléatoires

Fonctions aléatoires

- Une fonction $\{0,1\}^n \to \{0,1\}^m$...
- ... qui renvoie des chaînes de bits aléatoires

$x \longrightarrow$

$\longrightarrow f(x) = \$\$\$$

Fonctions aléatoires

Fonction aléatoire $\{0,1\}^3 \to \{0,1\}^{32}$

x	$f(x)$
000	0110 1001 0000 1111 1101 1110 1110 0001
001	0011 1101 0001 0101 1111 1100 1010 0111
010	0000 1000 1000 0000 0000 1100 1001 1110
011	0010 0000 0011 1010 0010 1011 0011 1011
100	1010 0110 0100 0100 0100 1010 1100 1110
101	0000 1001 1111 0110 1111 0111 0011 1100
110	1010 0011 1110 1100 1011 1110 1010 1001
111	1011 0010 0111 0111 0010 0110 1110 1000

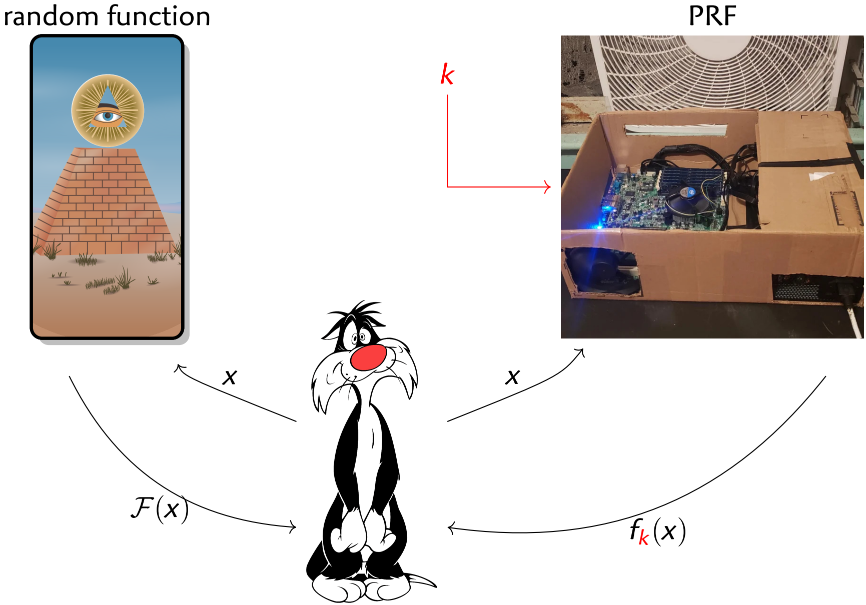
- Obtenue en tirant $2^3 \times 32$ bits aléatoires
  - Chaque sortie est tirée au hasard
  - $\{0,1\}^n \to \{0,1\}^m \rightsquigarrow 2^n$ sorties de m bits
  - 2^{m2^n} fonctions différentes possibles

Fonctions pseudo-aléatoires

- On ne peut pas vraiment manipuler des fonctions aléatoires
  - Description de taille exponentielle 🙄

Fonctions pseudo-aléatoires

- Clef secrète
- Algorithme d'évaluation efficace
- Calculatoirement indistinguable d'une « vraie » fonction aléatoire


random function
PRF
k
x
x
$\mathcal{F}(x)$
$f_k(x)$

Fonction (potentiellement) pseudo-aléatoires

- $f_k(x) := \mathrm{SHA256}(k \parallel x)$
- $f_k(x) := \mathrm{AES}_k(x)$
- $f_k(x) := x \oplus \mathrm{AES}_k(x)$ (Davies-Meyer)
- GOLD (DLOG à l'envers)

$$f_k(x) := (x+k)^g, \quad \text{avec } p-1 = gq \text{ et } q \text{ premier} \approx 2^{256}$$

- PRF de Legendre :

$$f_k(x) := \left(\frac{k+x}{p}\right), \left(\frac{k+x+1}{p}\right), \ldots, \left(\frac{k+x+n-1}{p}\right)$$

Usages des fonction pseudo-aléatoires

Chiffrement symétrique IND-CPA

- Appliquer un bourrage au message m
- Découper le message à chiffrer en blocs $m_0, \ldots, m_\ell$
- Tirer un IV aléatoirement
- Générer un masque jetable qui dépend de k et de l'IV :

$$\mathcal{E}_k(m) := IV, m_0 \oplus f_k(IV), m_1 \oplus f_k(IV+1), \ldots, m_\ell \oplus f_k(IV+\ell)$$

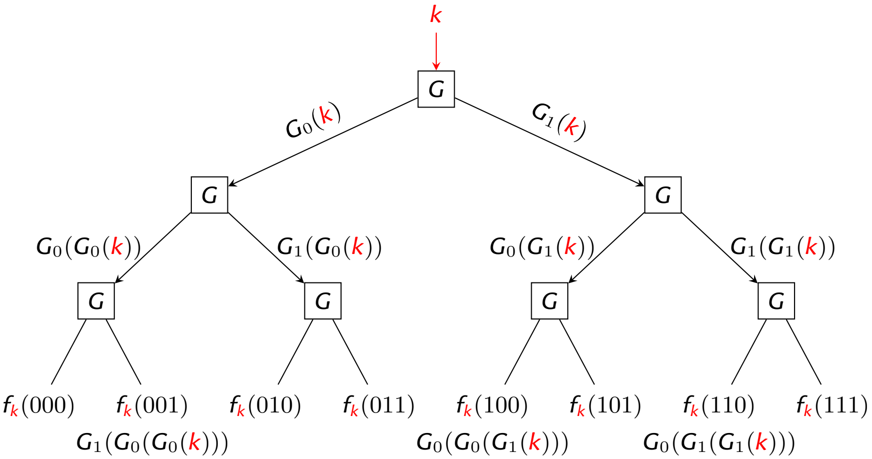
- f PRF $\Longrightarrow$ IND-CPA

Code d'authentification de message

- $\mathrm{MAC}_k(m) = f_k(H(m))$

Construction de Goldreich, Goldwasser et Micali (GGM — 1984)

- $G : \{0,1\}^n \to \{0,1\}^{2n}$ est un PRG
  - $G_0(x)$ est la première moitié de $G(x)$
  - $G_1(x)$ est le deuxième moitié de $G(x)$



Construction de Goldreich, Goldwasser et Micali (GGM — 1984)

- $G : \{0,1\}^n \to \{0,1\}^{2n}$ est un PRG
  - $G_0(x)$ est la première moitié de $G(x)$
  - $G_1(x)$ est le deuxième moitié de $G(x)$

Avec $G(x) = (g^x, h^x)$: la PRF de Naor-Reingold

- $f_{a_0,\ldots,x_n}(x) = g^{a_0 \prod_{i=1}^{n} {a_i}^{x_i}}$
- Indistinguable d'une fonction aléatoire si DDH est dur