Étude d'un FHE de troisième génération

Lucas Roux & Eric Sageloli

15 février 2019

Introduction

Un bref historique

- 2009 : un premier plan par Craig Gentry dans sa thèse :
  - idée du bootstrapping ;
- 2011 : premiers FHE de seconde génération : Z.Brakerski, V.Vaikuntanathan, J.Fan, F.Vercauteren :
  - basés sur LWE et ses variantes (comme RLWE) ;
  - une somme simple à définir ;
  - un produit en 2 étapes ;
- 2013 : premiers FHE de troisième génération : GWS par C.Gentry, B.Waters and A.Sahai, en 2013 :
  - basés sur LWE et ses variantes ;
  - produit et somme de même nature ;

Définition de GSW

GSW, premier essai :

Clé secrète : un vecteur $\vec{sk} \in \mathbb{Z}_q^N$

Clé publique : pk

Chiffrement : $\text{Encrypt}(\text{pk}, \mu) = C \in \mathbb{Z}_q^{N \times N}$ telle que

$$C\vec{sk} = \mu\vec{sk}$$

Déchiffrement : évident : recherche de valeur propre

Opérations homomorphes :

Pour $C_i = \text{Encrypt}(\mu_i) \quad (1 \leqslant i \leqslant 2)$ et $\lambda \in \mathbb{Z}_q$,

- **Somme :** $C_1 + C_2$

$$(C_1 + C_2)\,\vec{sk} = (\mu_1 + \mu_2)\,\vec{sk}$$

- **Produit :** $C_1 \times C_2$

$$(C_1 \times C_2)\ \vec{sk} = C_1\left(\mu_2\ \vec{sk}\right) = (\mu_1\mu_2)\,\vec{sk}$$

- **NAND :** $C_1 \times C_2 - \text{Id}$
- **Produit par scalaire :** λC_1

GSW, seconde tentative

Clé secrète : un vecteur $\vec{sk} \in \mathbb{Z}_q^N$

Clé publique : pk

Chiffrement : $\text{Encrypt}(\mu) = C \in \mathbb{Z}_q^{N \times N}$ telle que

$$C\vec{sk} = \vec{sk} + \vec{e} \qquad \text{avec } \vec{e} \text{ petit}$$

Déchiffrement : on prend un i tel que $\vec{sk}_i$ est grand

$$\begin{aligned}\text{Decrypt}(\vec{sk}, C) = \left\lfloor \frac{\left(C\vec{sk}\right)_i}{\vec{sk}_i} \right\rceil &= \left\lfloor \frac{\left(\mu\vec{sk}_i + \vec{e_i}\right)_i}{\vec{sk}_i} \right\rceil \\ &= \left\lfloor \mu + \frac{\vec{e_i}}{\vec{sk}_i} \right\rceil \\ &= \mu\end{aligned}$$

GSW, seconde tentative

Retour sur les opérations homomorphes :

- **Somme :** $C_1 + C_2$

$$(C_1 + C_2)\,\vec{sk} = (\mu_1 + \mu_2)\vec{sk} + \vec{e_1} + \vec{e_2}$$

- **NAND :** $C_1 \times C_2 - \mathrm{Id}$

$$\begin{aligned}(C_1 \times C_2 - \mathrm{Id})\,\vec{sk} &= C_1\left(\mu_2\vec{sk} + \vec{e_2} - \vec{sk}\right)\\ &= (\mu_1\mu_2 - 1)\vec{sk} + \mu_2\vec{e_1} + C_1\vec{e_2}\end{aligned}$$

Problème :
Les coefficients de $C_1\vec{e_2}$ peuvent être gros.

GSW, troisième tentative

On utilise une fonction Flatten qui a notamment les propriétés suivantes :

$$C \in \mathbb{Z}_q^{n \times n} \Rightarrow \text{Flatten}(C) \in \{0,1\}^{N \times N}$$

$$\langle \mathit{Flatten}(C), \vec{sk} \rangle = \langle C, \vec{sk} \rangle \quad \text{pour un secret } \vec{sk} \text{ bien choisi}$$

Clé secrète : un vecteur $\vec{sk} \in \mathbb{Z}_q^N$ bien choisi

Clé publique : pk

Chiffrement : $\text{Encrypt}(\text{pk}, \mu) = \text{Flatten}(C) \in \mathbb{Z}_q^{N \times N}$ pour C telle que

$$C\vec{sk} = \vec{sk} + \vec{e} \qquad \text{avec } \vec{e} \text{ petit}$$

Déchiffrement : on prend un i tel que $\vec{sk}_i$ est grand et :

$$\text{Decrypt}(\vec{sk}, C) = \left\lfloor \frac{\left(C\vec{sk}\right)_i}{\vec{sk}_i} \right\rceil$$

Opérations homomorphes : on applique Flatten aux précédentes opérations homomorphes.

clé publique et sécurité IND-CPA

$pk = A \in \mathbb{Z}_q^{N*m}$ une matrice telle que

$$A\,\vec{sk} = \vec{e}$$

où les coordonnées de $\vec{e}$ sont petites.

Cette matrice est choisie en se basant sur le problème DLWE de façon à ce que, pour de bons choix de paramètres, A est indistinguable d'une matrice choisie uniformément.

Pour chiffrer $\mu \in \mathbb{Z}_q$, on peut poser :

$$C = \mu\,\mathrm{Id} + AR$$

pour $R \in \{0,1\}^{m\times N}$ tirée uniformément.

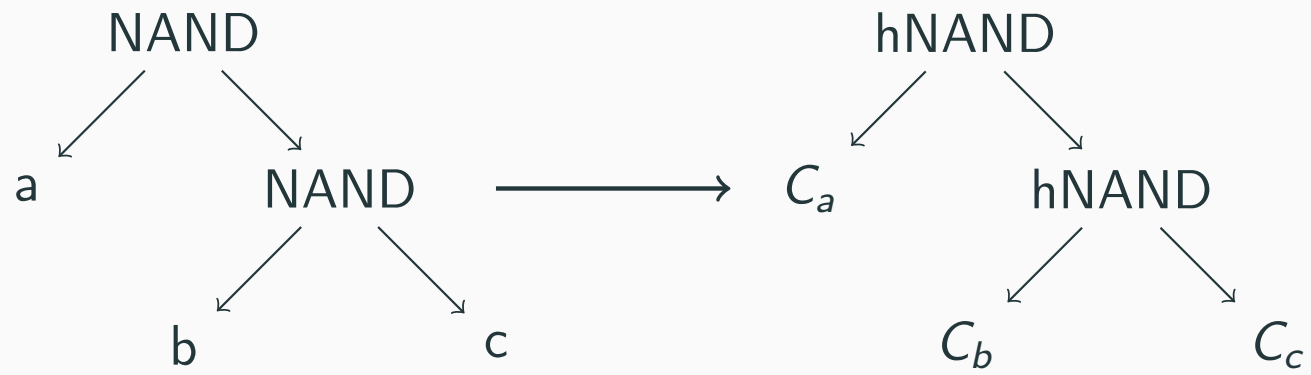
Alors : (A, AR) est indistinguable d'un choix uniforme de matrices (U, V) par un attaquant $\mathcal{A}$ polynomial probabiliste.

C peut donc se voir comme un one-time pad.

Mise en place d'un bootstrapping

Evaluation d'un circuit booléen, profondeur de NAND

Soit $a, b, c \in \{0, 1\}$ et C_a, C_b, C_c leurs chiffrés pour des clés $(\vec{sk}, \text{pk})$



$\Pi(a, b, c)$ $\qquad$ $\text{Eval}(\Pi, C_a, C_b, C_c)$

Si le circuit Π a une profondeur de NAND assez faible :

$$\Pi(a, c, b) = \text{Decrypt}(\vec{sk}, \text{Eval}\,(\Pi, C_a, C_b, C_b))$$

On note alors :

$$\text{Encrypt}\,(\text{pk}, \Pi(a, c, b)) \approx \text{Eval}\,(\Pi, C_a, C_b, C_b)$$

FHE avec bootstrapping

$$C = D + \text{erreur}$$

$$C = D + \text{erreur} \xrightarrow{\text{Decrypt}(\vec{sk}, C)} \mu \xrightarrow{\text{Encrypt}(\vec{sk}, \mu)} C_{\text{new}} = D + \text{erreur}$$

FHE avec bootstrapping

$$C = D + \text{erreur}$$

$$C = D + \text{erreur} \xrightarrow{\text{Decrypt}(\vec{sk}, C)} \mu \xrightarrow{\text{Encrypt}(\vec{sk}, \mu)} C_{\text{new}} = D + \text{erreur}$$

Soit Π le circuit booléen tel que

$$\Pi(\overrightarrow{binsk}) = \text{Decrypt}\left(\vec{sk}, C\right)$$

FHE avec bootstrapping

$$C = D + \text{erreur}$$

$$C = D + \text{erreur} \xrightarrow{\text{Decrypt}(\vec{sk}, C)} \mu \xrightarrow{\text{Encrypt}(\vec{sk}, \mu)} C_{\text{new}} = D + \text{erreur}$$

Soit Π le circuit booléen tel que

$$\Pi(\overrightarrow{binsk}) = \text{Decrypt}\left(\vec{sk}, C\right)$$

Alors :

$$\begin{aligned}\text{Encrypt}\left(\vec{sk}, \text{Decrypt}\left(\vec{sk}, C\right)\right) &= \text{Encrypt}\left(\vec{sk}, \Pi(\overrightarrow{binsk})\right) \\ &\approx \text{Eval}\left(\Pi, \text{Encrypt}(\vec{sk}, \overrightarrow{binsk})\right)\end{aligned}$$

- Si Π contient assez peu de NAND, on peut avoir un FHE.

Découpage de Decrypt

L'algorithme de déchiffrement est le suivant :

1. trouver $1 \leqslant i \leqslant l$ tel que $q/4 \leqslant 2^i < q/2$
2. calculer $a = C_i \cdot \vec{sk}$
3. retourner $|\frac{a}{\vec{sk}_i}|$

- On peut ramener le calcul du produit scalaire à une somme de nombres binaire ;
- Diviser par une puissance de 2 est un shift à gauche sur l'écriture binaire ;
- Calculer la valeur valeur absolue implique essentiellement de faire un complément à 2.

Voyons comment sommer deux nombres binaires.

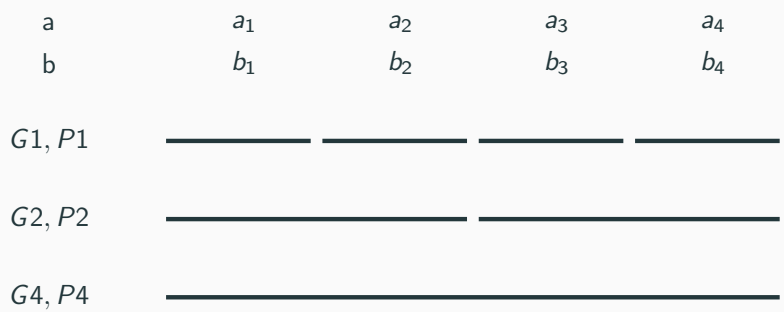
Sommer deux listes :

Somme classique entre deux nombres binaires :

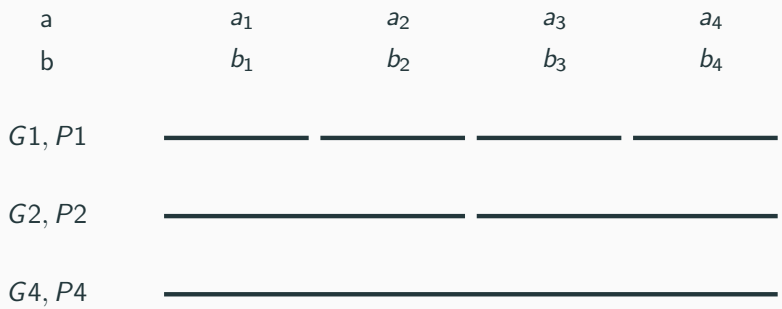
$$
\begin{array}{cccccc}
 & & \overset{c_1}{\curvearrowright} & \overset{c_2}{\curvearrowright} & & \overset{c_{s-1}}{\curvearrowright} \\
 & a_1 & a_2 & \cdots & a_s \\
+ & b_1 & b_2 & \cdots & b_s \\
\hline
 & r_1 & r_2 & \cdots & r_s
\end{array}
$$

- a_1 et b_1 présents dans la formule booléenne de r_s.
- profondeur de NAND en $O(s)$

Sommer deux listes : carry lookahead adder



Sommer deux listes : carry lookahead adder



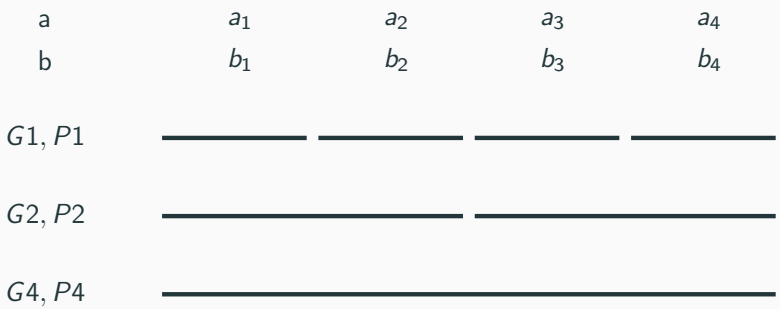
- G pour génération

$$\begin{array}{cccccc} & a_5 & a_6 & a_7 & a_8 & \\ + & b_5 & b_6 & b_7 & b_8 & \\ \hline & X & X & X & X & 1 \end{array}$$

- P pour propagation

$$\begin{array}{cccccc} & a_5 & a_6 & a_7 & a_8 & \\ + & b_5 & b_6 & b_7 & b_8 & \\ + & 1 & & & & \\ \hline & X & X & X & X & 1 \end{array}$$

Sommer deux listes : carry lookahead adder



- G pour génération

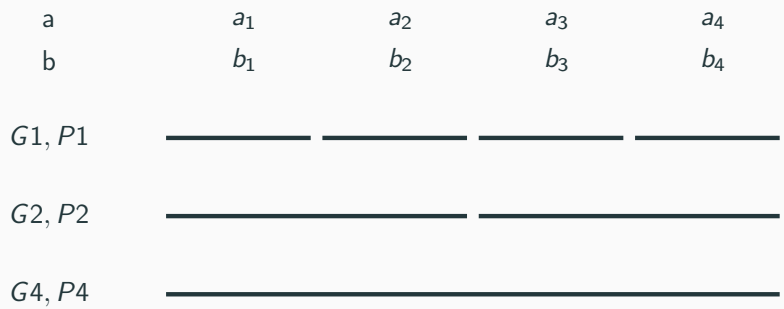
$$\begin{array}{cccccc} & a_5 & a_6 & a_7 & a_8 & \\ + & b_5 & b_6 & b_7 & b_8 & \\ \hline & X & X & X & X & 1 \end{array}$$

- P pour propagation

$$\begin{array}{cccccc} & a_5 & a_6 & a_7 & a_8 & \\ + & b_5 & b_6 & b_7 & b_8 & \\ + & 1 & & & & \\ \hline & X & X & X & X & 1 \end{array}$$

$$(G1)_i = a_i \wedge b_i \quad (P1)_i = a_i \vee b_i$$

Sommer deux listes : carry lookahead adder



- G pour génération

$$\begin{array}{cccccc} & a_5 & a_6 & a_7 & a_8 & \\ + & b_5 & b_6 & b_7 & b_8 & \\ \hline & X & X & X & X & 1 \end{array}$$

- P pour propagation

$$\begin{array}{cccccc} & a_5 & a_6 & a_7 & a_8 & \\ + & b_5 & b_6 & b_7 & b_8 & \\ + & 1 & & & & \\ \hline & X & X & X & X & 1 \end{array}$$

$$(G1)_i = a_i \wedge b_i \quad (P1)_i = a_i \vee b_i$$

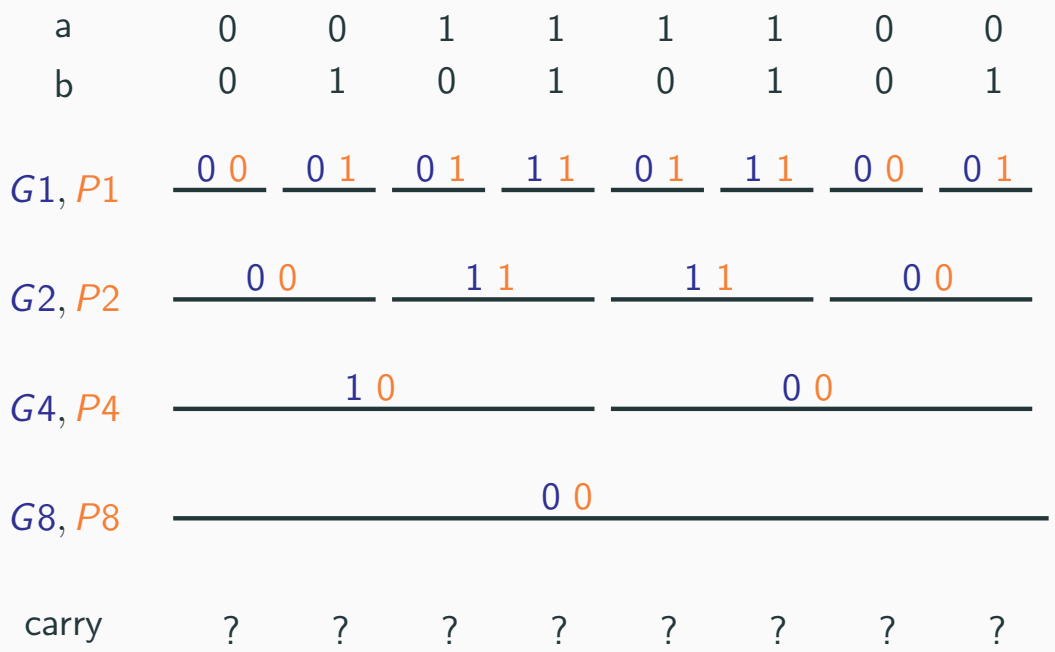
$G2^i, P2^i$ — 1

$G2^{i+1}, P2^{i+1}$ — 1 — 2

$$(G2^{i+1})_1 = (G2^i)_2 \vee ((G2^i)_1 \wedge (P2^i)_2)$$

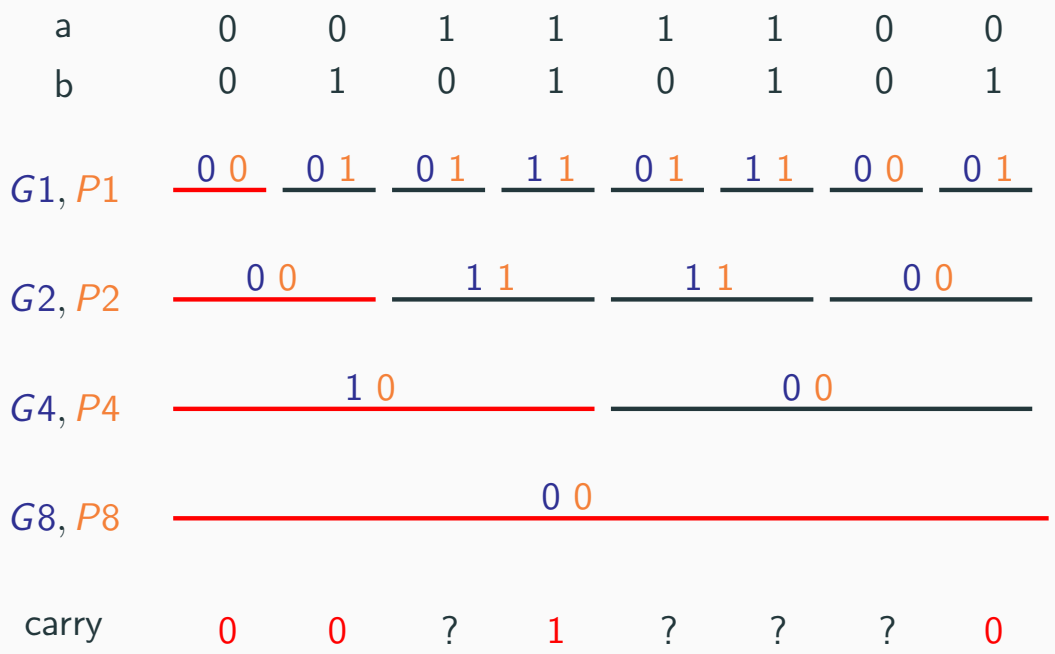
$$(P2^{i+1})_1 = (G2^i)_1 \wedge (P2^i)_2$$

Sommer deux listes : carry lookahead added



Les variables de générations de blocs commençants par 0 calculent des retenues.

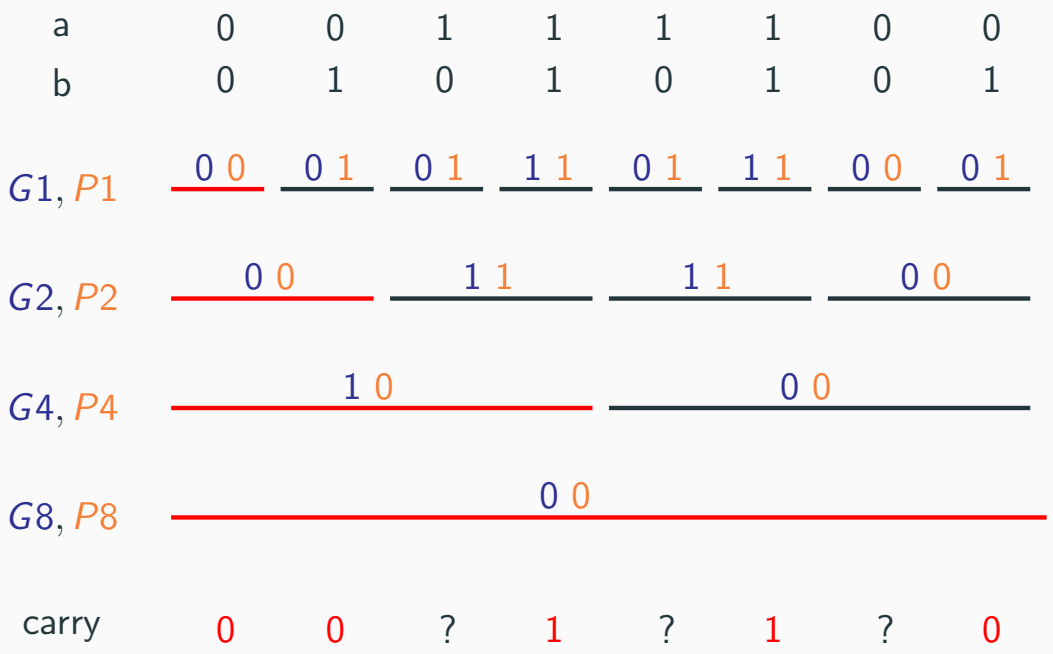
Sommer deux listes : carry lookahead added



Les variables de générations de blocs commençants par 0 calculent des retenues.

$$c_6 = G2_3 \vee (c_4 \wedge P2_3) = 1 \vee (1 \wedge 1) = 1$$

sommer deux listes : carry lookahead added



Les variables de générations de blocs commençants par 0 calculent des retenues.

$$c_6 = G2_3 \vee (c_4 \wedge P2_3) = 1 \vee (1 \wedge 1) = 1$$

$$c_7 = G1_7 \vee (c_6 \wedge P1_7) = 0 \vee (1 \wedge 0) = 0$$

Effectuer un bootstrapping

Profondeur de NAND totale en :

$$\mathcal{O}(\log(\log(q)) + \log(n))$$

avec n un paramètre du système.

Certains choix de paramètres permettent d'effectuer un bootstrapping en garantissant que le cryptosystème est IND-CPA.

Effectuer un bootstrapping

Profondeur de NAND totale en :

$$\mathcal{O}(\log(\log(q)) + \log(n))$$

avec n un paramètre du système.

Certains choix de paramètres permettent d'effectuer un bootstrapping en garantissant que le cryptosystème est IND-CPA.

sécurité	taille du secret	taille de la clé	taille d'un chiffré
$\lambda = 8$	3 Ko	113 Mo	135 Go
$\lambda = 16$	12 Ko	2 Go	6 To
$\lambda = 32$	45 Ko	32 Go	176 To

Un œil sur le monde réel

The Gate Bootstrapping API

- librairie open source utilisable en C et C++
- utilise une version modifiée du cryptosysteme GSW, avec une variante de LWE nommée TFHE.
- s'appuie notamment sur des travaux de I. Chillotti, N. Gama, M. Georgieve et M. Izabachène

Performances : pour un ordinateur 64-bit simple coeur (i7-4930MX) cadencé à 3.00GHz, le bootstrapping se fait en un temps moyen de 52ms et la clé de bootstrapping fait environ 24 Mo.

The Gate Bootstrapping API

- Alice génère des clés, chiffre deux nombres de 16 bits et les inscrits dans un fichier ;
- le cloud récupère les données, applique homomorphiquement la fonction minimum aux deux nombres et inscrit le résultat dans un fichier ;
- Alice récupère et déchiffre le résultat.

Performances : pour un paramètre de sécurité $\lambda = 110$ et sur un ordinateur 64-bit quadri-coeur (i5-7200U CPU) cadencé à 2.50GHz, on obtient un temps de 2.10s.

On a alors des données de tailles :

Données	Taille
Clé secrète	79 Mo
Clé publique et clé de bootstrapping	79 Mo
Chiffrement d'un bit	2 Ko
Chiffrement des deux nombres	64 Ko

Ce dont nous n'avons pas parlé

Concernant la définition de GSW :

- flatten, dont nous avons caché la définition sous le tapis ;
- deux autres algorithmes de déchiffrement.

Concernant les choix paramètres et la sécurité :

- le problème DLWE ;
- les gaussiennes discrètes ;
- la librairie sagemath lwe_estimator pour estimer la sécurité de paramètres LWE ;
- La notion de leveled FHE et des choix des paramètres pour un leveled FHE.

Concernant le bootstrapping :

- des optimisations supplémentaires pour sommer plusieurs nombres binaires ;
- contraintes de sécurité supplémentaires (sécurité circulaire).