INF8750: Projet de session

Blindsort: le tri en aveugle

Félix Larose-Gervais

April 17, 2024

Contents

1 Introduction

1.1 Contexte

Le chiffrement homomorphe est un outils important dans la construction de Privacy Enhancing Technologies (PETs). Il permet de chiffrer des données de manière à ce qu'elles puissent être utilisées dans un calcul par une tierce partie sans avoir à être déchiffrées au préalable.

Peu après la publication du cryptosystème RSA [Rivest *et al.*, 1978b], ses autheurs ont remarqué que le système était partiellement homomorphe [Rivest *et al.*, 1978a]. Ils se sont ensuite demandé s'il était possible de construire un schéma complètement homomorphe.

Définition 1 *Une fonction $\varphi : \mathbb{G} \to \mathbb{H}$ est appelé un **homomorphisme** si $\forall x, y \in \mathbb{G}$ on a:*

$$\varphi(xy) = \varphi(x)\varphi(y)$$

Définition 2 *On parle de **chiffrement homomorphe** lorsqu'un schéma $\Pi = (Gen, Enc, Dec)$ a pour propriété que son algorithme de chiffrement Enc est un homomorphisme (modulo Dec).*

Définition 3 *Un schéma est dit **partiellement homomorphe** s'il permet l'évaluation de certains calculs sur ses cryptogrammes sans avoir à les déchiffrer.*

Par exemple, le schéma $RSA = (Gen, Enc, Dec)$ est partiellement homomorphe.

$$RSA = \begin{cases} Gen(1^n) & := (N, e, d) \leftarrow GenRSA \\ Enc_{N,e}(m) & := m^e \mod N \\ Dec_d(c) & := c^d \mod N \end{cases}$$

Où $GenRSA$ choisi aléatoirement p, q deux grands nombres premiers et calcule $N = pq$ et $e, d \in \mathbb{Z}_N$ tels que $ed \equiv 1 \pmod N$. On dit que (N, e) forme la clé publique et (N, d) la clé privée.

En effet, soient $m_1, m_2 \in \mathbb{Z}_N$, tels que $c_1 = Enc_{N,e}(m_1)$ et $c_2 = Enc_{N,e}(m_2)$, on a:

$$\begin{aligned} Enc_{N,e}(m_1 m_2) &\equiv (m_1 m_2)^e \pmod N \\ &\equiv m_1^e m_2^e \pmod N \\ &\equiv Enc_{N,e}(m_1) Enc_{N,e}(m_2) \end{aligned}$$

On constate que l'on peut faire faire des multiplications de cryptogrammes RSA à une tierce partie, et en déchiffrer plus tard le résultat correct. D'autres cryptosystème ont été découverts partiellement homomorphes, tels que [ElGamal, 1985], [Benaloh, 1994] et [Paillier, 1999].

Définition 4 *Un schéma est dit **complètement homomorphe** s'il permet l'évaluation de tout calcul sur ses cryptogrammes sans avoir à les déchiffrer.*

L'existence de schéma de chiffrement complètement homomorphe est resté une question ouverte pendant 30 ans. [Gentry, 2009] a montré que c'était possible, en présentant le premier schéma complètement homomorphe, et a présenté une idée nouvelle, le bootstrapping, qui permet d'adapter certains schéma partiellement homomorphes pour les rendre complètement homomorphes.

Suite aux travaux de Gentry, plusieurs schémas de chiffrement complètement homomorphe ont vu le jour, notament TFHE [Chillotti *et al.*, 2020], implémenté dans librairie TFHE-rs [Zama, 2022]. Par dessus cette librairie, le projet RevoLUT [Azogagh, 2024] introduit de nouvelles primitives de rotation et permutation aveugle, celles-ci ont servi à la mise en place de Oblivious Turing Machine [Azogagh *et al.*, 2023].

1.2 Objectif du projet

Bien que le chiffrement homomorphe aie beaucoup évolué depuis sa conception, les schémas proposés aujourd'hui sont encore très inefficaces. Le but de ce projet est d'implémenter quelques algorithmes de tri en aveugle (c'est-à-dire opérant sur des données chiffrées) et de comparer leur performance entre eux et l'état de l'art.

Les deux premiers algorithmes implémentés sont basés sur le Direct Sort de [Çetin *et al.*, 2015], l'un utilisant les primitives de Blind Matrix Access et de Blind Permutation offertes par RevoLUT, puis un autre utilisant directement les primitives de TFHE-rs. Ensuite, on présente une technique nouvelle basée sur deux passes de Blind Permutation de RevoLUT.

1.3 Schéma TFHE

Le schéma de chiffrement homomorphe TFHE est basé sur le problème de l'apprentissage avec erreur (Learning With Errors, LWE). La sécurité du schéma est basée sur la difficulté du problème de recherche de plus court vecteur dans les réseaux Euclidiens [Regev, 2009].

$$LWE = \begin{cases} Gen(1^n) & := s \xleftarrow{R} \{0,1\}^n \\ Enc_s(m) & := (a, \sum a_i s_i + \Delta m + e),\ a \xleftarrow{R} \mathbb{Z}_q^n, e \xleftarrow{R} \chi_\sigma \\ Dec_s(a,b) & := (b - \sum a_i s_i)/\Delta \end{cases}$$

Où p, q des puissances de 2 sont les tailles de l'espace des messages et des cryptogrammes respectivement, tel que $\Delta = q/p$, avec Δ la taille réservée au bruit. Le bruit e est tiré d'une distribution Gaussienne centrée de variance σ. On a donc $Enc_s : \mathbb{Z}_p \to \mathbb{Z}_q^n \times \mathbb{Z}_q$. On peut vérifier que le schéma LWE est additivement homomorphe.

$$\begin{aligned} Enc_s(m_1) + Enc_s(m_2) &= (a_1, a_1 \cdot s + \Delta m_1 + e_1) + (a_2, a_2 \cdot s + \Delta m_2 + e_2) \\ &= (a_1 + a_2, (a_1 + a_2) \cdot s + \Delta(m_1 + m_2) + (e_1 + e_2)) \\ &= Enc_s(m_1 + m_2) \end{aligned}$$

Similairement, on peut chiffrer des tableaux de taille N (une puissance de 2) en les encodant comme des éléments de $\mathcal{R}_q = \mathbb{Z}_q[X]/(X^N + 1)$, c'est-à-dire des polynômes à coefficients entiers (modulo q) de degré au plus $N - 1$.

$$RLWE = \begin{cases} Gen(1^n) & := S \xleftarrow{R} \mathcal{R}_2 \\ Enc_s(M) & := (A, AS + \Delta M + E),\ A \xleftarrow{R} \mathcal{R}_q, E \xleftarrow{R} \chi_\sigma \\ Dec_s(A,B) & := (B - AS)/\Delta \end{cases}$$

Il est important de mentionner que les opérations homomorphes sur les cryptogrammes (R)-LWE accumulent le bruit. Par conséquent, après un certains nombre d'opérations, le bruit déborde dans les bits de poids fort réservés au message, et rend le cryptogramme indéchiffrable. Afin de régler ce problème, [Gentry, 2009] a proposé l'idée de bootstrapping; c'est-à-dire d'évaluer la procédure de déchiffrement en aveugle, à partir d'un chiffré de la clé et d'un chiffré du cryptogramme dont on veut réduire le bruit.

[Chillotti *et al.*, 2020] ont proposé une amélioration à la procédure coûteuse de Gentry, basée sur la rotation aveugle de polynômes. Leur approche, en plus d'être plus rapide, permet l'évaluation

de n'importe quelle fonction discrétisée f, en encodant les images de la fonction dans les coefficients d'un polynôme $P = \sum f(i)X^i$. L'évaluation aveugle de f consiste donc en une rotation aveugle de P suivie d'une extraction du premier coefficient.

1.4 RevoLUT

Le projet RevoLUT, basé sur TFHE-rs, expose une structure appelée une Look-Up Table (LUT). Cette structure est un tableau à taille fixe de messages chiffrés, sur lequel a été implémenté plusieurs nouvelles primitives tel que la rotation aveugle, la permutation aveugle et l'accès dans un tableau ou une matrice en aveugle.

2 Première implémentation

2.1 Direct Sort

L'idée principale de l'algorithme est de construire une permutation à partir des comparaisons entre les paires d'éléments de la liste, puis de l'appliquer.

Algorithm 1 Direct Sort dans RevoLUT

```
function BlindSort(T: LUT)
    σ ← [0; n]
    for i = 0 to n do
        for j = 0 to i do
            b ← BMA(L, A[i], A[j])
            σ[i] ← σ[i] + b
            σ[j] ← σ[j] + 1 − b
        end for
    end for
    return BlindPermutation(A, σ)
end function
```

La comparaison aveugle (LT) est implémentée grâce à la primitive Blind Matrix Access de RevoLUT. On cherche dans la matrice binaire triangulaire inférieure stricte (C telle que les entrées $C_{ij} = [i < j]$) en utilisant les deux valeurs à comparer comme indices.

Par exemple, pour la LUT $A = [5, 4, 6, 3]$, on obtient:

$$C = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 \end{pmatrix} \qquad L = \begin{pmatrix} 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 \end{pmatrix}$$

On observe que la somme des colonnes de L forment la permutation $\sigma = (2, 1, 3, 0)$ qui, lorsque appliquée à A donne la liste triée $\sigma(A) = [3, 4, 5, 6]$.

2.2 Complexité

Cette approche requiert $\frac{n^2-n}{2}$ comparaisons et une permutation, ce qui domine le temps d'exécution puisque chaque comparaison prends plus d'une seconde.

3 Deuxième implémentation

Cette deuxième méthode évite les comparaisons aveugles, au cout d'une permutation additionnelle.

3.1 Algorithme

L'idée est d'interpréter la liste donnée comme une permutation, que l'on applique à elle-même, ce qui produit une liste clairsemée mais relativement ordonnée. Ensuite, on construit et applique une seconde permutation compactant les entrées non-nulles.

Algorithm 2 Permutation Sort

```
function BlindSort(A: LUT)
    B ← BlindPermutation(A, A)
    offset ← 0
    for i = 0 to n do
        offset ← offset + NULL(B[i])
        σ[i] ← B[i] − offset
    end for
    return BlindPermutation(B, σ)
end function
```

La comparaison à zéro (NULL) est essentiellement un accès aveugle dans une LUT statique de la forme $[1, 0, \ldots, 0]$.

Par exemple, pour $A = [5, 2, 7, 3, 0, 0, 0, 0]$, on peut la lire comme une permutation et l'appliquer à elle-même pour obtenir $B = [0, 0, 2, 3, 0, 5, 0, 7]$. Ensuite, on veut construire $\sigma = (4, 5, 0, 1, 6, 2, 7, 3)$ telle que $\sigma(B) = [2, 3, 5, 7, 0, 0, 0, 0]$.

3.2 Complexité

Cette deuxième approche ne requiert qu'un nombre linéaire en n d'additions et de comparaisons scalaires, mais 2 permutations aveugles.

4 Méthodologie

4.1 Property-based Testing

À l'aide de la librairie QuickCheck, on peut comparer le comportement des nouvelles primitives aveugles avec leur contrepartie en clair sur des entrées arbitraires. Cela permet de s'assurer de leur exactitude pendant le développement.

4.2 Micro-benchmarking

Afin de comparer les temps d'exécution des différentes implémentations et de l'état de l'art, on utilisera la librairie Criterion pour établir des benchmarks avec statistiques et graphiques. En plus des travaux déjà mentionnés, on peut retrouver des résultats sur la performance du tri aveugle dans les articles de [Baldimtsi et Ohrimenko, 2014], [Jönsson *et al.*, 2011], [Zuber et Sirdey, 2021] et [Choffrut *et al.*, 2023].

References

[Azogagh, 2024] Azogagh, S. (2024). Revolut. https://github.com/sofianeazogagh/revoLUT.

[Azogagh *et al.*, 2023] Azogagh, S., Deflour, V. et Killijian, M.-O. (2023). Oblivious Turing Machine. Cryptology ePrint Archive, Paper 2023/1643. https://eprint.iacr.org/2023/1643. Récupéré le 2024-02-05 de https://eprint.iacr.org/2023/1643

[Baldimtsi et Ohrimenko, 2014] Baldimtsi, F. et Ohrimenko, O. (2014). Sorting and Searching Behind the Curtain: Private Outsourced Sort and Frequency-Based Ranking of Search Results Over Encrypted Data. Publication info: Published elsewhere. Major revision. Financial Cryptography and Data Security (FC) 2015. Récupéré le 2024-02-29 de https://eprint.iacr.org/2014/1017

[Benaloh, 1994] Benaloh, J. (1994). Dense probabilistic encryption. Dans *Proceedings of the workshop on selected areas of cryptography*, 120–128.

[Chillotti *et al.*, 2020] Chillotti, I., Gama, N., Georgieva, M. et Izabachène, M. (2020). TFHE: Fast Fully Homomorphic Encryption Over the Torus. *Journal of Cryptology*, *33*(1), 34–91. http://dx.doi.org/10.1007/s00145-019-09319-x. Récupéré le 2024-02-05 de http://link.springer.com/10.1007/s00145-019-09319-x

[Choffrut *et al.*, 2023] Choffrut, A., Guerraoui, R., Pinot, R., Sirdey, R., Stephan, J. et Zuber, M. (2023). SABLE: Secure And Byzantine robust LEarning. arXiv:2309.05395 [cs] version: 4. Récupéré le 2024-03-01 de http://arxiv.org/abs/2309.05395

[ElGamal, 1985] ElGamal, T. (1985). A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE transactions on information theory*, *31*(4), 469–472.

[Gentry, 2009] Gentry, C. (2009). Fully homomorphic encryption using ideal lattices. Dans *Proceedings of the forty-first annual ACM symposium on Theory of computing*, STOC '09, 169–178., New York, NY, USA. Association for Computing Machinery. http://dx.doi.org/10.1145/1536414.1536440. Récupéré le 2024-03-19 de https://doi.org/10.1145/1536414.1536440

[Jönsson *et al.*, 2011] Jönsson, K. V., Kreitz, G. et Uddin, M. (2011). Secure multi-party sorting and applications. Cryptology ePrint Archive, Paper 2011/122. https://eprint.iacr.org/2011/122. Récupéré de https://eprint.iacr.org/2011/122

[Paillier, 1999] Paillier, P. (1999). Public-key cryptosystems based on composite degree residuosity classes. Dans J. Stern (dir.). *Advances in Cryptology — EUROCRYPT '99*, 223–238., Berlin, Heidelberg. Springer Berlin Heidelberg.

[Regev, 2009] Regev, O. (2009). On Lattices, Learning with Errors, Random Linear Codes, and Cryptography. arXiv:2401.03703 [quant-ph], http://dx.doi.org/10.48550/arXiv.2401.03703. Récupéré le 2024-03-20 de http://arxiv.org/abs/2401.03703

[Rivest *et al.*, 1978a] Rivest, R. L., Adleman, L. et Dertouzos, M. L. (1978a). On data banks and privacy homomorphisms. *Foundations of Secure Computation, Academia Press*, 169–179.

[Rivest *et al.*, 1978b] Rivest, R. L., Shamir, A. et Adleman, L. (1978b). A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, *21*(2), 120–126. http://dx.doi.org/10.1145/359340.359342. Récupéré le 2024-01-26 de https://dl.acm.org/doi/10.1145/359340.359342

[Zama, 2022] Zama (2022). TFHE-rs: A Pure Rust Implementation of the TFHE Scheme for Boolean and Integer Arithmetics Over Encrypted Data. https://github.com/zama-ai/tfhe-rs.

[Zuber et Sirdey, 2021] Zuber, M. et Sirdey, R. (2021). Efficient homomorphic evaluation of k-nn classifiers. *Proc. Priv. Enhancing Technol.*, *2021*(2), 111–129.

[Çetin *et al.*, 2015] Çetin, G. S., Doröz, Y., Sunar, B. et Savaş, E. (2015). Depth Optimized Efficient Homomorphic Sorting. In K. Lauter et F. Rodríguez-Henríquez (dir.), *Progress in Cryptology – LATINCRYPT 2015*, volume 9230 61–80. Cham: Springer International Publishing. Series Title: Lecture Notes in Computer Science