INF8750: BlindSort Design Doc

Félix Larose-Gervais

Mars 2024

Contents

1 Introduction

1.1 Motivation

Le chiffrement homomorphe est un excellent outils pour préserver la confidentialité. Malheureusement, son cout en efficacité est encore trop grand pour justifier sa mise en place à l'échelle. Pour ce projet, on tente d'améliorer la performance aveugle d'un algorithme fondamental; le tri.

1.2 Objectif

Le but du projet est d'implémenter un algorithme de tri en aveugle dans RevoLUT [Azogagh, 2024], un projet rust basé sur TFHE-rs [Zama, 2022], une librairie libre de chiffrement complètement homomorphe [Marcolla *et al.*, 2022]. La structure principale offerte par RevoLUT est une LookUp Table (LUT), un conteneur ordonné de chiffrés muni de plusieurs opérations aveugles comme la rotation et la permutation.

2 Première implémentation

Cette première méthode est similaire à celle de [Iliashenko et Zucca, 2021], basée sur l'algorithme Direct Sort proposé par [Çetin *et al.*, 2015].

2.1 Algorithme

L'idée est de calculer, pour une LUT donnée $A = [a_0, a_1, \ldots, a_{n-1}]$, une matrice L définie par:

$$L_{ij} = \begin{cases} LT(a_i, a_j) & \text{if } i < j \\ 0 & \text{if } i = j \\ 1 - LT(a_j, a_i) & \text{if } i > j \end{cases}$$

Ensuite, on construit une permutation de tri à partir des poids de Hamming de ses colonnes.

Algorithm 1 Direct Sort

```
function BlindSort(A)
    σ ← [0; n]
    for i = 0 to n do
        for j = 0 to i do
            b ← LT(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)
  $B \leftarrow BlindPermutation(A, A)$
  $offset \leftarrow 0$
  for $i = 0$ to n **do**
    $offset \leftarrow offset + NULL(B[i])$
    $\sigma[i] \leftarrow B[i] - offset$
  end for
  return $BlindPermutation(B, \sigma)$
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.

[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

[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

[Iliashenko et Zucca, 2021] Iliashenko, I. et Zucca, V. (2021). Faster homomorphic comparison operations for BGV and BFV. *Proceedings on Privacy Enhancing Technologies*. Récupéré le 2024-01-25 de https://petsymposium.org/popets/2021/popets-2021-0046.php

[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

[Marcolla *et al.*, 2022] Marcolla, C., Sucasas, V., Manzano, M., Bassoli, R., Fitzek, F. H. P. et Aaraj, N. (2022). Survey on Fully Homomorphic Encryption, Theory, and Applications. *Proceedings of the IEEE*, *110*(10). http://dx.doi.org/10.1109/JPROC.2022.3205665

[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