Ring Ring! Who's There? A Privacy Preserving Mobile Number Search

Akshit Aggarwal[1]

Private set intersection (PSI) allows any two parties (say client and server) to jointly compute the intersection of their sets without revealing anything else. Fully homomorphic encryption (FHE)-based PSI is a cryptographic solution to implement PSI-based protocols. Most FHE-based PSI protocols implement hash function approach and oblivious transfer approach. The main limitations of their protocols are 1) high communication complexity, that is, $O(xlogy)$ (where x is total number of elements on client side, and y is total number of elements on server side), and 2) high memory usage due to SIMD packing for encrypting large digit numbers. In this work, we design a novel tree-based approach to store the large digit numbers that achieves less communication complexity, that is, $O(|d|^2)$ (where d is digits of a mobile number). Later we implement our protocol using Tenseal library. Our designed protocol opens the door to find the common elements with less communication complexity and less memory usage.

***Index Terms*—Digit, FHE, Mobile number, PSI, Tree.**

I. Introduction

PRIVATE set intersection (PSI) is a cryptographic protocol that allows two parties, say P_1 and P_2, having sets x and y, respectively, to compute the intersection $x \cap y$ without revealing the privacy of any other elements. PSI is widely used in various fields such as smart contact discovery [1], healthcare industry [2], supply chain and inventory management [3], [4], and many more [5], [6], [7]. Over the last few decades, PSIs have become ubiquitous due to their practical implementations. A long line of work [8], [9], [10], [11] implements PSI-based protocol and achieves high communication overhead. Thus, there is a strong need to reduce communication overhead between the parties.

Fully homomorphic encryption (FHE)-based PSI protocols are widely used to reduce communication overhead.[1] FHE allows to perform computation using single-instruction-multiple-data (SIMD) that allows to encrypt multiple data simultaneously using a single vector. FHE increases the communication complexity, that is, linear in the size of both sets. Over the last decade, there has been a long line of publications that discusses the reduction in communication complexity by considering either hash-based approach or oblivious transfer [1], [12], [13], [14]. A few of them are discuss as follows.

To the best of our knowledge, Meadows et al. [15] first introduce the secure PSI protocol, which was later describe by Huberman et al. [16]. The approach propose in [16] leverages the multiplicative homomorphic property of Diffie-Hellman key exchange algorithm [17] but increases the running time due to many exponential operations. Thus, to reduce the running time, several other paradigms (say, hash-based approach, and oblivious transfer [18], [19], [20]) are evolved.

First, Chen et al. [12] hash their data in cuckoo hash table, and homomorphic evaluation is performed over the server. The main drawback of their protocol is high communication overhead when all the elements are store in a single slot of cuckoo hash. To its improvement, Pinkas et al. [14] add k auxiliary positions in cuckoo hash and map these positions to the single slot of cuckoo hash. The main drawback is server communication overhead due to k auxiliary positions. Thereafter, Ghosh et al. [21] propose polynomial-based approach where all the data are interpolated into polynomials, and oblivious operations (say polynomial-related operations) are applied. The main drawback of this approach is slow running time. Later on, Ghosh et al. [13] improve their own polynomial approach by considering the minimum number of common elements between client and server (say t). The authors try to find the root of this polynomial to obtain the common elements. The main drawback of this work is being unable to find the common elements when number of common elements is less than t. Chakraborti et al. [22] extend the work of [13] when the common elements between client and server are less than t. The authors compute hamming distance between the sets, whenever the distance is 0, the element is common. Thereafter, Hu et al. [23] encode their data in Bloom filter and evaluate a polynomial over it. The main drawback of their work is high communication complexity. Later on, Ghosh et al. [24] recursively divide the datasets into smaller chunks and apply the symmetric set difference operations.[2] The propose work in [24] is again not applicable whenever the intersection size between the two sets are less than t. Kußmaul et al. [25] recursively store the elements into cuckoo hash table and later apply the homomorphic subtraction operations. The main drawback of their work is high communication complexity whenever the entries of dataset are consider as large.

Thus, in order to reduce communication overhead, FHE-based PSI protocols face several challenges, mainly high communication complexity whenever the entries of dataset are consider as larger and pre-defined minimum number of common elements t that leads to leakage about dataset.

In this work, we first store all 10-digit mobile numbers on the server. The numbers are store digit by digit in the form of tree.

Corresponding author: Akshit Aggarwal, Department of CSE, Indian Institute of Information Technology Guwahati, Assam (India) (email: akshit.aggarwal@iiitg.ac.in

[1]as FHE allows to perform operations over encrypted data without decrypting it.

[2]symmetric set difference operation is $x - y \cup y - x$

The first digit of all mobile numbers is stored at Level 1 of the tree. The second digit is stored at Level 2, and this process continues such that the 10^{th} digit of each mobile number is stored at Level 10. Each digit at a given level is connected to its corresponding digit at the next level through a link, thereby maintaining the hierarchical relationship between the digits across levels (as discuss in Section III-A). Later on, a client homomorphically searches the number by encrypting it digit by digit and returns the common mobile numbers.

A. *Problem Formulation*

Consider a server that stores 10-digit mobile numbers. There is a client who has a 10-digit mobile number and wants to check whether this number is registered on the server or not without learning anything else.

B. *Our Contribution*

We propose a 10-digit mobile number search protocol that determines if the client-entered number is on the server.

- First, on the server side, a novel tree-based approach is propose to store all the 10-digit mobile numbers (say, a total of n numbers) that helps in reducing the communication complexity by $O(|d|^2)$ (where d is total digit of a mobile number, that is, 10 in our case, and $d << n$).
- We implement our protocol using Tenseal library to check whether the number is present on the server or not.

C. *Organization of our paper*

In Section II, we discuss the preliminaries used in this work. The propose methodology of this work is discuss in Section III. Section IV discusses the implementation setup and results. Later, Section V discusses the limitations of our work. Finally, we present the conclusion of our work in Section VI.

II. Preliminary

In this section, we discuss the encryption scheme that is used in this work, that is, BFV homomorphic encryption.

A. *BFV Homomorphic Encryption*

The BFV encryption scheme facilitates simultaneous operations on encrypted data vectors and utilizes the Single Instruction Multiple Data (SIMD) model. During encryption, BFV introduces noise whenever plaintext is transformed into ciphertext (where the plaintext is an element of $\mathbb{Z}_p$, with p being a prime number). This noise grows progressively as homomorphic operations are applied to the ciphertexts [26]. BFV encryption supports the following operations for processing plaintexts and ciphertexts while ensuring that computations are performed modulo p.

- **Add**(c_1, c_2): Takes two ciphertexts c_1 and c_2, which encrypt the plaintexts m_1 and m_2, and returns a ciphertext encrypting $(m_1 + m_2) \mod p$ (element-wise addition).
- **AddPlain**(c, m): Takes a ciphertext c (encrypting m_1) and a plaintext m_2, and returns a ciphertext encrypting $(m_1 + m_2) \mod p$ (element-wise addition).
- **Sub**(c_1, c_2): Takes two ciphertexts c_1 and c_2, which encrypt the plaintexts m_1 and m_2, and returns a ciphertext encrypting $(m_1 - m_2) \mod p$ (element-wise subtraction).
- **SubPlain**(c, m): Takes a ciphertext c (encrypting m_1) and a plaintext m_2, and returns a ciphertext encrypting $(m_1 - m_2) \mod p$ (element-wise subtraction).
- **Mul**(c_1, c_2): Takes two ciphertexts c_1 and c_2, which encrypt the plaintexts m_1 and m_2, and returns a ciphertext encrypting $(m_1 \cdot m_2) \mod p$ (element-wise multiplication).
- **MulPlain**(c, m): Takes a ciphertext c (encrypting m_1) and a plaintext m_2, and returns a ciphertext encrypting $(m_1 \cdot m_2) \mod p$ (element-wise multiplication).

III. Proposed Methodology

In this section, we discuss the methodology of our work. We divide our work into two phases. In first phase, we discuss the design of server (where all 10-digit mobile numbers are stored in tree-like manner (as shown in Fig. 1). Later, in second phase, we discuss the existence of searched query. Finally, we discuss the complexity of our propose protocol.

A. *Design of Server*

In this phase, all the 10-digit mobile numbers are stored on the server using tree-based approach. The first digit of all mobile numbers are stored at Level 1 of the tree. Similarly, the second digit is stored at Level 2, and this process continues until the 10^{th} digit, which is stored at Level 10. Each digit at a given level is linked to its corresponding digit at the next level, preserving the hierarchical structure and relationships across the levels (as shown in Algorithm 1). The above observation is illustrate with following toy example.

Example 1

1) The server contains five 3-digit numbers: $212, 221, 231, 312, 321$.
2) At Level 1 of the tree:
   - Digits 2 and 3 are stored.
3) At Level 2 of the tree:
   - Digits $1, 2$, and 3 are stored and are connected to digit 2 from Level 1.
   - Digits 1 and 2 are stored and are connected to digit 3 from Level 1.
4) At Level 3 of the tree:
   - Digits $2, 1, 1, 2$, and 1 are stored.
   - These digits are connected to the corresponding digits at Level 2.
5) The hierarchical structure and connections across levels are shown in Fig. 2.

B. *Retrieving common elements*

This phase is divided into three phases, that is, encryption, evaluation of server, and decryption. In encryption phase, client first encrypts the mobile number digit by digit (say $Enc(d_i)$) and sends it to the server. Server homomorphically

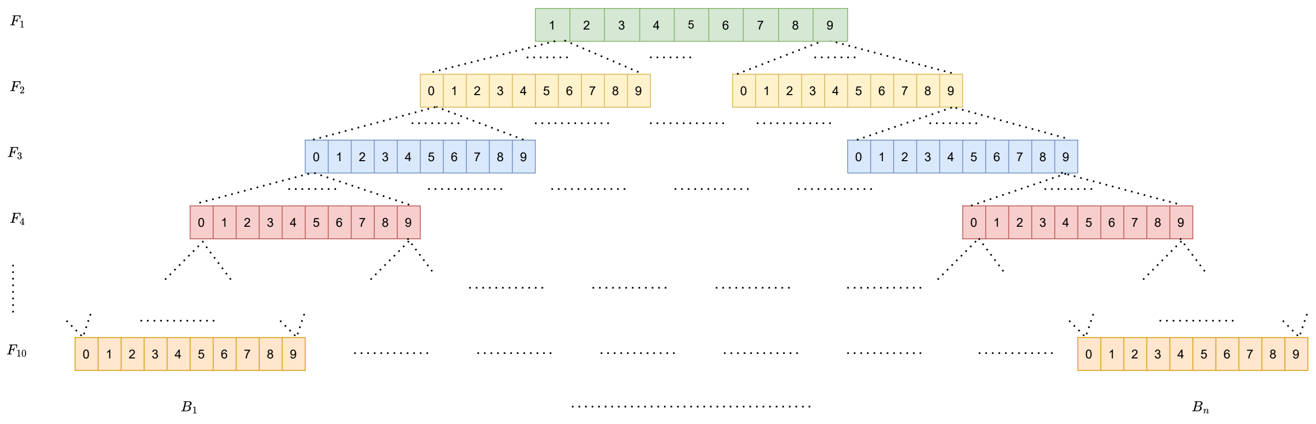

Fig. 1. Here F_i (where $1 \leq i \leq 10$) represents the Level i of tree or digits of mobile numbers (meaning F_1 is first digit of all mobile numbers whereas F_{10} is the 10^{th}-digit of mobile number). B_x (where $1 \leq x \leq n$, $n \geq 1$) represents the numbers of child nodes present at Level i. Dotted 'Black' line represents the connection between the levels or digits.

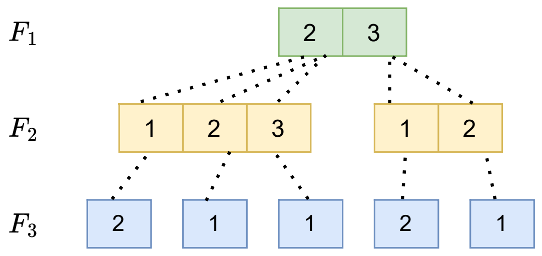

Fig. 2. Here F_i (where $1 \leq i \leq 3$) represents the Level i of tree. Also, F_i represents the digit number (meaning F_1 is first digit, F_2 is second digit, and F_3 is third digit). Dotted 'Black' line represents the connection between the levels or digits.

Algorithm 1: Tree-Based Storage of 10-Digit Mobile Numbers

```
Input: List of 10-digit mobile numbers:
       $\mathcal{N} = \{n_1, n_2, \ldots, n_k\}$
Output: Tree-like structure where each level stores
        digits and maintains links to the next level
1  Initialize: Create a root structure $\mathcal{T}$ to represent Level
     1 of the tree.
2  Each node in the tree represents a digit and contains
     pointers to its child nodes.
3  foreach $n \in \mathcal{N}$ do
4      Extract Digits: Let $n = d_1 d_2 \cdots d_{10}$, where $d_i$ is
         the $i^{th}$ digit.
5      Insert into Tree:
6      Set CurrentNode ← $\mathcal{T}$ ;     // Start from
         the root
7      for $i = 1$ to 10 do
8          if $d_i$ is not a child of CurrentNode then
9              Add $d_i$ as a child of CurrentNode
10         CurrentNode ← Child node corresponding
             to $d_i$ ;   // Move to the next level
11 Return: $\mathcal{T}$ ;          // The constructed tree
     structure
```

subtracts the number from digits (say d) at each level (say $Enc((d_i - d).r)$, where $r > 0$ is random number to maintain the privacy of other digits) and returns the result to the client. Client decrypts it, and if the decryption result is non-zero, then digit d_i is not present and searches for the next number; otherwise, digit d_i is present and checks for next digit d_j. For checking d_j, apply homomorphic operation only for the connecting child node. If decryption result is non-zero in any of the levels, it signifies that the number is not present, and check for the next number (as discuss in Algorithm 2). The below toy example illustrates the above observations.

Example 2

1) The client needs to check whether the number 231 is present on the server (by considering Fig. 2).
2) The client encrypts the first digit, 2, using $Enc(2)$ and sends it to the server.
3) The server homomorphically subtracts the first digit from all entries in Level F_1:

$$Enc((2-2).r),\ Enc((3-2).r),$$

and returns the encrypted results to the client.
4) The client decrypts the results (e.g., $0, r$). Since the result contains 0, it indicates that the first digit 2 is present.
5) The client encrypts the second digit, 3, using $Enc(3)$ and sends it to the server.
6) The server homomorphically subtracts the second digit from all entries in Level F_2, that is, connected to digit 3 of Level F_2:

$$Enc((1-3).r),\ Enc((2-3).r),\ Enc((3-3).r),$$

and sends the encrypted results back to the client.
7) The client decrypts the results (e.g., $-2r, -r, 0$). Since the result contains 0, it indicates that the second digit 3 is present.
8) The client encrypts the third digit, 1, using $Enc(1)$ and sends it to the server.

9) The server homomorphically subtracts the third digit from all entries in Level F_3:
$$Enc((1-1).r),$$
and returns the encrypted result to the client.
10) The client decrypts the result (e.g., 0). Since the result contains 0, it indicates that the third digit 1 is present.
11) Since all digits have been verified successfully, the number 231 is confirmed to be present on the server.

Algorithm 2: Homomorphic Search for Mobile Number Presence

```
Input: 10-digit mobile number $M = \{d_1, d_2, \ldots, d_{10}\}$,
       Encrypted server database $Enc(F)$
Output: Presence or Absence of $M$ in the server
1  Initialization:
2  Client encrypts each digit $d_i$ of $M$ using $Enc(d_i)$
3  Send $Enc(d_1)$ to the server for verification at Level 1
     ($F_1$)
4  for each level $F_k$ $(k = 1, \ldots, 10)$ do
5      Server performs homomorphic subtraction for all
         digits at $F_k$ that connected with $F_{k-1}$:
6              $Enc((x - d_k).r)$ for each $x \in F_k$, where
         $r > 0$ is a random multiplier
7      Server sends the encrypted results back to the
         client.
8      Client decrypts the result:
9              if Decrypted result contains 0 then
10         Digit $d_k$ is present. Proceed to next digit $d_{k+1}$
             in $M$.
11         if $k < 10$ then
12             Send $Enc(d_{k+1})$ to the server for
                 verification at Level $F_{k+1}$ (child nodes).
13     else
14         Output: Number $M$ is not present. Stop
             search.
15 Output: Number $M$ is present.
```

C. *Complexity Discussion*

In this work, we perform a worst-case analysis of our protocol. Consider a d-digit number, where each digit is homomorphically search within a block at every level of the tree. Since the search is performed sequentially for d levels, and each level involves searching through d elements in the corresponding block, the overall complexity of the protocol is $O(d^2)$.

IV. Performance Evaluation

In this section, we evaluate the performance of our protocol. First, we discuss the details of implementation setup. Thereafter, we discuss the implementation results. Our proof of code is written in Tenseal library and publicly available at https://github.com/akshit-aggarwal/Ring-Ring-Who-s-There-A-Privacy-Preserving-Mobile-Number-Search/tree/main.

A. *Implementation Setup*

The proof-of-concept code for our observation is written in Jupyter notebook (python version 3.9.10) using TenSeal library. Jupyter notebook is installed over Intel(R) Core(TM) i5-10500 CPU @ 3.10GHz 3.10 GHz. The installed memory is 8GB.

B. *Implementation Results*

1) *Data collection and preprocessing*

In this phase, 10-digit mobile numbers are collected from running python 10-digit random number (where first digit is either $6, 7, 8$ or 9) generation program. Thus, total of 15000 mobile numbers are collected. The total execution time for generating 10-digit random numbers are 0.03 seconds. Thereafter, the implementation of server is performed, where all the 10-digits mobile numbers are stored in tree-like manner. The total average execution time taken by our server is 0.55 seconds.

2) *Implementation of our protocol*

In this phase, we discuss the encryption, decryption, and evaluation of client's query. For our implementation, we use Tenseal library by considering parameter settings for plain-text modulus, which is approximately 2^{20}, and polynomial modulus degree, which is 2^{13} (that is used to control security and computation performance). The random value r is taken from private subset having range of $(1, 10^2)$. We execute our program for several iterations where, at each iteration, the client contains various mobile numbers. At each iteration, client can find the common elements. Table I indicates the execution time for client to privately find the common element. The implementation results show that the execution time is high whenever the client needs to search for more elements (as it searches the number using linear scan).

# Mobile numbers clients wants to search	Execution time (in sec(s))
1	1.00
10	3.83
100	32.55
1000	310.86
10000	2897.54

TABLE I
It represents the total time taken by our designed protocol when client wants to search large amount of mobile numbers.

V. Limitation

The propose work utilizes the linear scan for homomorphic searching of the phone numbers. The proposed work leaks the block numbers (as it will return the block number where the digit is present).

VI. Conclusion

In this work, we design a privacy-preserving protocol for retrieving the common 10-digit mobile numbers. Our design protocol searches large-digit mobile numbers without any SIMD packing. The achieved communication complexity is less as compared to all other existing works.

References

[1] N. Angelou, A. Benaissa, B. Cebere, W. Clark, A. J. Hall, M. A. Hoeh, D. Liu, P. Papadopoulos, R. Roehm, R. Sandmann, P. Schoppmann, and T. Titcombe, "Asymmetric Private Set Intersection with Applications to Contact Tracing and Private Vertical Federated Machine Learning," *arXiv preprint arXiv:2011.09350*, 2020. [Online]. Available: https://arxiv.org/abs/2011.09350

[2] Y. Qian, J. Shen, P. Vijayakumar, and P. K. Sharma, "Profile Matching for IoMT: A Verifiable Private Set Intersection Scheme," *IEEE journal of biomedical and health informatics*, vol. 25, no. 10, pp. 3794–3803, 2021.

[3] B. Liu, X. Zhang, R. Shi, M. Zhang, and G. Zhang, "SEPSI: A Secure and Efficient Privacy-Preserving Set Intersection with Identity Authentication in IoT," *Mathematics*, vol. 10, no. 12, p. 2120, 2022.

[4] D. P. Hellwig and A. Huchzermeier, "Distributed Ledger Technology and Fully Homomorphic Encryption: Next-Generation Information-Sharing for Supply Chain Efficiency," in *Innovative Technology at the Interface of Finance and Operations: Volume II*. Springer, 2022, pp. 31–49.

[5] L. Shen, X. Chen, D. Wang, B. Fang, and Y. Dong, "Efficient and Private Set Intersection of Human Genomes," in *2018 IEEE International Conference on Bioinformatics and Biomedicine (BIBM)*. IEEE, 2018, pp. 761–764.

[6] Y. Zheng, R. Lu, H. Zhu, S. Zhang, Y. Guan, J. Shao, F. Wang, and H. Li, "SetRkNN: Efficient and Privacy-Preserving Set Reverse kNN Query in Cloud," *IEEE Transactions on Information Forensics and Security*, vol. 18, pp. 888–903, 2022.

[7] J. Hu, Y. Zhao, B. H. M. Tan, K. M. M. Aung, and H. Wang, "Enabling Threshold Functionality for Private Set Intersection Protocols in Cloud Computing," *IEEE Transactions on Information Forensics and Security*, 2024.

[8] P. Benny, S. Thomas, and Z. Michael, "Faster Private Set Intersection Based on OT Extension," in *Usenix Security*, 2014, pp. 797–812.

[9] B. Pinkas, T. Schneider, G. Segev, and M. Zohner, "Phasing: Private Set Intersection Using Permutation-based Hashing," in *24th USENIX Security Symposium (USENIX Security 15)*, 2015, pp. 515–530.

[10] C. Dong, L. Chen, and Z. Wen, "When Private Set Intersection Meets Big Data: An Efficient and Scalable Protocol," in *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, 2013, pp. 789–800.

[11] M. Lambæk, "Breaking and Fixing Private Set Intersection Protocols," *Cryptology ePrint Archive*, 2016. [Online]. Available: https://eprint.iacr.org/2016/665

[12] H. Chen, K. Laine, and P. Rindal, "Fast Private Set Intersection from Homomorphic Encryption," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, 2017, pp. 1243–1255.

[13] S. Ghosh and M. Simkin, "The Communication Complexity of Threshold Private Set Intersection," in *Annual International Cryptology Conference*. Springer, 2019, pp. 3–29.

[14] B. Pinkas, M. Rosulek, N. Trieu, and A. Yanai, "PSI from PaXoS: Fast, Malicious Private Set Intersection," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2020, pp. 739–767.

[15] C. Meadows, "A More Efficient Cryptographic Matchmaking Protocol for Use in the Absence of a Continuously Available Third Party," in *1986 IEEE Symposium on Security and Privacy*. IEEE, 1986, pp. 134–134.

[16] B. A. Huberman, M. Franklin, and T. Hogg, "Enhancing Privacy and Trust in Electronic Communities," in *Proceedings of the 1st ACM conference on Electronic commerce*, 1999, pp. 78–86.

[17] D.-H. K. Exchange, "Diffie-Hellman Key Exchange," *Diffie% E2*, vol. 80, 1976.

[18] M. Naor and B. Pinkas, "Efficient Oblivious Transfer Protocols." in *SODA*, vol. 1, 2001, pp. 448–457.

[19] R. Pagh and F. F. Rodler, "Cuckoo Hashing," *Journal of Algorithms*, vol. 51, no. 2, pp. 122–144, 2004.

[20] S. Tarkoma, C. E. Rothenberg, and E. Lagerspetz, "Theory and Practice of Bloom Filters for Distributed Systems," *IEEE Communications Surveys & Tutorials*, vol. 14, no. 1, pp. 131–155, 2011.

[21] S. Ghosh and T. Nilges, "An Algebraic Approach to Maliciously Secure Private Set Intersection," in *Annual international conference on the theory and applications of cryptographic techniques*. Springer, 2019, pp. 154–185.

[22] A. Chakraborti, G. Fanti, and M. K. Reiter, "Distance-Aware Private Set Intersection," in *32nd USENIX Security Symposium (USENIX Security 23)*, 2023, pp. 319–336.

[23] J. Hu, J. Chen, W. Dai, and H. Wang, "Fully Homomorphic Encryption-based Protocols for Enhanced Private Set Intersection Functionalities," *Cryptology ePrint Archive*, 2023.

[24] S. Ghosh and M. Simkin, "Threshold Private Set Intersection with Better Communication Complexity," in *IACR International Conference on Public-Key Cryptography*. Springer, 2023, pp. 251–272.

[25] J. Kußmaul, M. Akram, and A. Tueno, "Unbalanced Private Set Intersection from Homomorphic Encryption and Nested Cuckoo Hashing," *Cryptology ePrint Archive*, 2023.

[26] J. Fan and F. Vercauteren, "Somewhat Practical Fully Homomorphic Encryption," *Cryptology ePrint Archive*, 2012. [Online]. Available: https://ia.cr/2012/144