Homomorphic Encryption for Privacy-Preserving Genome Sequences Search

Masato Oguchi
Dept. of Information Sciences
Ochanomizu University
Tokyo, Japan
oguchi@is.ocha.ac.jp

Kurt Rohloff
NJIT Cybersecurity Research Center
New Jersey Institute of Technology
Newark, NJ
rohloff@njit.edu

Yuki Yamada
Dept. of Information Sciences
Ochanomizu University
Tokyo, Japan
yuki@ogl.is.ocha.ac.jp
{Corresponding Author}

***Abstract*—Genome sequence search is useful, for example, in clinical applications where a care provider needs to select a treatment option for a patient based on the exact kind of cancer the patient might have. Homomorphic encryption is a desirable technology to be used for this application because it is non-interactive. However, privacy-preserving genome sequence search using homomorphic encryption has been a practical challenge because of scalability issues driven by the depth of computations that need to be supported for privacy-preserving genome sequence search. In this paper, we build off of earlier privacy-preserving genome sequence search results to design, implement and compare two approaches to a client-server style system for privacy-preserving genome sequence search. There is a myriad of options and design trade-offs associated with the application of homomorphic encryption in this domain driven, for example, by choices in data encoding, scheme selection, and even encryption software library. We particularly focus on the use of the BGV and BFV homomorphic encryption schemes provided by the HElib and PALISADE open-source homomorphic encryption software libraries. Our results show that using the BFV-based approach in PALISADE provides optimal results for this application over our sample data.**

***Index Terms*—Homomorphic Encryption (HE), Genome Sequence, Secure Search, Privacy**

I. Introduction

Ever since the Human Genome Project [1] and the 1000 Genomes Project [2] have begun publishing catalogues of human variation and genotype data, genomic data analytics have found increasingly practical and important use in various fields, not only for medical use but also for a wider range of application. Privacy is critical for analytics on personal genomic data, whether in the area Genome-Wide Association Studies (GWAS), personalized medicine, or more. Privacy challenges associated with analytics on genomic data have been exacerbated by recent innovations that made it much less expensive to sequence genetic information, leading to dramatic increases in the availability of sequenced genomic data. Furthermore, it is difficult to manage genome data in secure on-premise environments at hospitals or research institutes because expensive specialized storage facilities and computers are necessary to manage and analyze genomic data sets. The above advances have led to an interest in the development and application of privacy-preserving analysis techniques for genomic data. However, because genomic data is potentially voluminous, making scalability challenges are important when analyzing genetic data, especially when needed to be done in a privacy-preserving manner.

Of particular interest, genomic search applications look for some specific sub-strings, thus driving the need for a system that can conduct privacy-preserving string searches on vast amounts of genome data. Generic cloud computing environments are not feasible to address this need due to security and privacy concerns engendered by multi-tenancy, and the cloud may be managed by unknown and un-trusted individuals. A simple solution to the cloud-based storage of privacy-sensitive genomic information is to use encryption. If the privacy-preserving genome search system is built with common symmetric- or public-key encryption, the decryption key would be passed to the cloud to enable analytics, thus creating a privacy concern.

Only homomorphic encryption techniques enable non-interactive computation on the data when it is encrypted. Additive homomorphic encryption can also be used as an encryption method, but it is considered that preventing genome data leakage with additive homomorphic encryption is difficult because we cannot conduct complex calculations with it [3]. Previous research into secure search methods on genomic data has leveraged Fully Homomorphic Encryption (FHE) to enable privacy-preserving genomic analysis. FHE supports non-interactive computation on encrypted data. Hence, FHE allows a client to upload a corpus of genomic data to a high-performance off-premise computation environment and then search on that genomic data without leaking its private information to the computation host. However, search operations are considered to be ”deep”, meaning they are not naively efficient when running on homomorphically encrypted data. This has led to recent interest in enabled practical encrypted search operations on genomic data using homomorphic encryption [4].

Prof. Oguchi and Ms. Yamada are partly supported by JST CREST Grant Number JPMJCR1503, Japan. Prof. Rohloff is partly supported by the Defense Advanced Research Projects Agency (DARPA) and the Army Research Laboratory (ARL) under Contract Numbers W911NF-15-C-0226 and W911NF-15-C-0233. The views expressed are those of the authors and do not necessarily reflect the official policy or position of the Department of Defense or the U.S. Government.

Prior efforts show secure search methods we build on that use Fully Homomorphic Encryption (FHE) to protect privacy [4]- [5]. In these methods FHE encrypted data is uploaded to a cloud environment for privacy-preserving non-interactive computation without decryption, meaning that decryption keys are not uploaded to the cloud with the encrypted data. Despite leveraging discrete data structure Positional-Burrows Wheeler Transform (PBWT) [6] and optimizing the calculation procedure, calculation costs on a server are still an issue in the approaches we leverage because of the large FHE resource requirements. Although there have been attempts to accelerate these systems by introducing decentralized computing such as in [7] as well as calculation optimization as in [4]- [5], the calculation costs on the cloud are still too large to put into practice.

In this paper, we implement the system in two designs that are proposed by the previous work [4]- [5] with two FHE schemes, BGV and BFV provided by HElib and PALISADE, and compare their performance in order to explore design trade-offs. In Section II we introduce the motivating application of privacy-preserving genome sequences search. In Section III, we introduce relevant features of FHE techniques and their prior support in FHE software libraries that we use. In Section IV, we provide a broad overview of relevant prior work. In Section V, we discuss designs trade-offs and approaches that we build on and explore. In Section VI, we discuss our implementation, and experimental settings and show experimental results from deploying our techniques on real-world data. In Section VII, we analyze the result of our experimentation. In Section VIII, we conclude this research and discuss future directions.

II. Genome Sequence Search Application

The goal of an application for the privacy-preserving genome sequences search is for clients to query if there are matches between a query string and the data stored in a genome database stored on an off-site server [3]. Genomic data are composed of sequences of 4 different kinds of nucleotides – A, G, C, and T –, therefore, we can regard this genome sequences search as a 4-kind character search [7]. A representation of this operation is seen in Figure 1.

Fig. 1. Genome Sequences Search

We assume a security model of a privacy-preserving genome search, where the outsourcing system is implemented in the client-server style. A server holds a set of genome sequences data aligned by each sample in a database. A representation of this operation is seen in Figure 2.

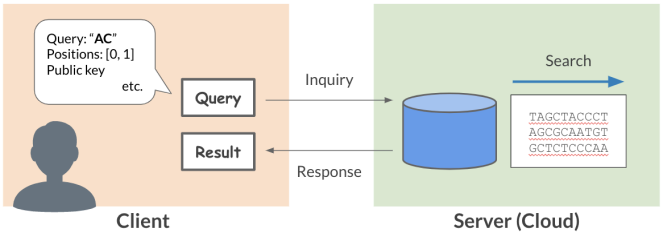

Fig. 2. Application Overview

By arranging the genome sequence of each sample inside the database in a row, it becomes possible to search every sample in a specific position of the genome sequence of each column. Clients send the inquiry to a server in order to calculate the matches with the query and database. This query includes not only the encrypted string that the client wants to search the genome sequence for but also some other parameters such as the multiple numbers of starting points of the search for genome data strings (search positions). By designating multiple numbers of positions including dummy ones, clients can hide the true one from the server. On receiving an inquiry from a client, the server conducts match searching on the data with FHE calculations, and then transmits the result to the client. This result indicates that whether there are any matches between the query string and genome sequences or not.

III. Fully Homomorphic Encryption

As discussed in the introduction, we leverage Fully Homomorphic Encryption (FHE) to provide privacy-preserving genome sequences search. As discussed in prior work [7] the primary property of FHE is that general computations can be performed on encrypted data. Concretely for the modern schemes (such as [8]), this means that we leverage Additive and Multiplicative homomorphisms on encrypted data to translate general computations into circuits that can be evaluated on encrypted data. As seen respectively in Formulas (1) and (2) below, these homomorphisms are called the Additive Homomorphism (which supports addition over encrypted data), and the Multiplicative Homomorphism (which supports multiplication over encrypted data.)

Additive/Multiplicative Homomorphism

$$Encrypt(m) \oplus Encrypt(n) = Encrypt(m+n) \quad (1)$$

$$Encrypt(m) \otimes Encrypt(n) = Encrypt(m \times n) \quad (2)$$

Fully Homomorphic Encryption (FHE) supports both of the Additive and Multiplicative homomorphism properties. By leveraging these properties users can support the evaluation of polynomial circuits over ciphertexts analogous to how they would support similar circuits evaluated on unencrypted plaintext.

A. Characteristics

FHE was first proposed by Rivest et al. in 1987 [9], but was not known to be feasible until a candidate scheme was discovered by Gentry in 2009 [10]. This first scheme leverages polynomial rings and ideal lattices, and the encrypted text is constructed by encrypted data and random noise in order to guarantee its difficulty to decrypt without the appropriate secret key. This early scheme was computationally inefficient. For example, the ciphertext of Gentry's implementation would be 1GB on encrypting 1bit data. There have been tremendous recent strides in developing increasingly more efficient schemes (such as BFV [8]) and their implementations (for example, [11]). These recent schemes are also more expressive in addition to being more efficient. For example, Lu et al. [12] show a scheme that supports a comparison homomorphism in addition to addition and multiplication homomorphisms.

There are still many large challenges with FHE. For example, noise accumulates in ciphertexts when computations are performed on them. As this noise grows, the ciphertexts eventually can't be decrypted if too many computations are performed. This results in incorrect decryption when the noise amount exceeds its noise threshold. The random noise in ciphertexts grow additively with additive operations and multiplicatively with every multiplication operation. This noise growth would normally limit the size of computations that could be performed with FHE. However, there is a special method called *bootstrapping*, which reduces the noise embedded in a ciphertext, with the drawback that the bootstrapping operations are extremely computationally intensive.

Note that many practical applications of FHE schemes use a limited version of FHE without bootstrapping. The "reduced" version of FHE is called Somewhat Homomorphic Encryption (SHE or SwHE) [13]. This is the ability to conduct some simple calculations that can be derived with one-time multiplication and multiple times addition, such as the inner product of the vector, distribution, and correlation.

B. Software Libraries

With the tremendous advances in the basic FHE schemes, there have been corresponding advances in the implementation of these constructions in open-source libraries available for broad use.

HElib [14] is one of the first and one of the more well known such libraries that support lattice-based Homomorphic Encryption in C++. This library supports Brakerski-Gentry-Vaikuntanathan (BGV) scheme [15], along with many optimizations and a recent version of the library now supports the CKKS scheme [16]. HElib currently supports bootstrapping.

PALISADE [11] is a more recent open-source software library for general lattice cryptography that supports several homomorphic encryption schemes: LTV [17], Stehle-Steinfeld [18], BFV/BFV-rns [8] and BGV [15]. Contrary to HElib, PALISADE does not yet support bootstrapping.

There are several other homomorphic encryption libraries such as SEAL [19] and TFHE [20], but we focus on HElib and PALISADE in this research.

IV. Previous Work

A. PBWT-sec

Several previous attempts have been made to realize practical privacy-preserving string search for genome sequences. PBWT-sec [3] is an efficient two-party prefix much-counting protocol that combines additive homomorphic encryption and an efficient data structure for much searching called Positional-Burrows Wheeler Transform (PBWT) [6]. PBWT is a discrete data structure that sorts genome data by column and is used to search for a substring match for a set of aligned genome sequences rapidly. The server of PBWT-sec has a genome sequences database as PBWT, that is transformed from an original aligned genome sequences database. In its searching phase, the server access to a look-up vector that is derived from PBWT recursively. This is named recursive oblivious transfer (ROT) [6]. When the query string length is l, ROT consists of l times vector-lookups, which needs l rounds of communication between the client and the server. PBWT-sec also devises the idea that the client passes multiple amounts of search positions, which includes dummy ones, to a server in order to preserve the privacy of clients with hiding the positions that the client actually uses.

The C++ implementation of PBWT and PBWT-sec is published on GitHub [21]- [22].

B. Genome Sequences Search with FHE

Although FHE engenders a much longer computation time than that of additive homomorphic encryption, we can extend the PBWT-sec approach to use computation methods that search with wildcards and compute statistics based on the search result by building genome sequences search with FHE.

There are two relevant prior attempts by Ishimaki et al. [4]-[5]. First [4] proposes multi-round privacy-preserving genome sequence searches with FHE on the basis of PBWT-sec [3]. This approach replaces additive homomorphic encryption methods in PBWT-sec with fully homomorphic methods and also introduces the packing technique by Smart et al. [23]. The other [5] propose an efficient approach for one-round search with FHE by introducing bootstrapping and reducing the runtime of the system by optimizing the calculation procedure. These two work use HElib [14] and its BGV implementation as a software library for FHE calculations. The detail of the system design that is proposed by each work is discussed in Sec. V.

There has also been prior work to accelerate these FHE-based PBWT systems with parallel processing [7]. It is possible to realize the following three approaches as the method for parallelizing the FHE-based PBWT computation: decentralizing each individual data point, decentralizing each independent calculation, or decentralizing the independent algorithm. The first one, where each worker has an independent database, is the simplest to implement and also leverages techniques that use the results of previous calculations repeatedly. Thus, this prior work adopts the division of the database on the server side as a distribution method.

V. Design and Approach

A. Design and Tradeoffs

We build our approach to support the needed depth of computation and the depth limitations of FHE to avoid bootstrapping for a practical approach to private genome sequence search and thus limit the depth of computations to keep the noise in ciphertexts less than noise threshold for decryption. We thus build from the two systems proposed in [4] and [5]. We start from the design proposed by [4] shown in Fig. 3.

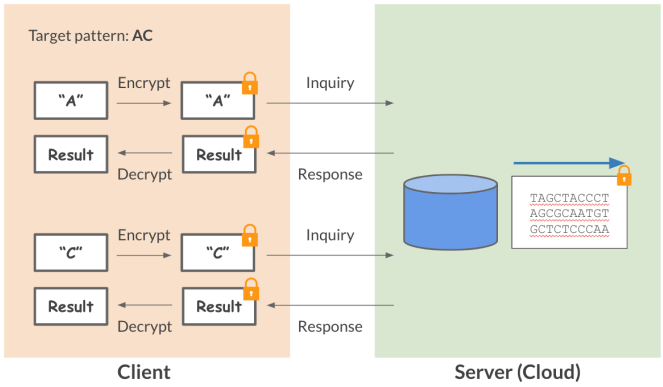

Fig. 3. Application Design 1

(1) The client encrypts one character of the query string and then passes the resulting ciphertext to the server with other parameters.
(2) The server then performs FHE computations and then returns the result to the client.
(3) The client decrypts the intermediate result.
(4) The client encrypts the next character of the query using the result and sends it to the server.
(5) (2)-(4) are repeated as many times as the length of the query.

In this design, the server operates over a single character at a time. This reduces the depth of computation on the server and enables reduced noise to remove the need for bootstrapping. However, computation costs on the clients and communication costs increase as the length of the query increases. Since each communication involves large data transfer, this design is inappropriate for the clients with limited communication resources and requires that the clients both be available and have appropriate computation resources for repeated encryption and decryption.

The other design built off of from [5] is shown in Fig. 4.

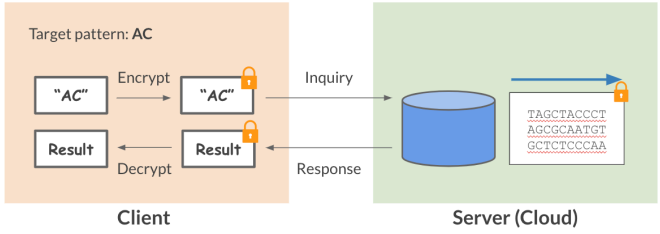

Fig. 4. Application Design 2

(1) The client encrypts the whole query string and then passes the encrypted query string with supporting parameters to the server.
(2) The server performs FHE computations and then transmits the encrypted result to the client.
(3) The client gains a result by decrypting the received data.

The server in Design 2 supports the whole string search. This Design 2 is more appropriate for the client with limited computation power, as compared to Design 1. However, The data size of ciphertexts and also FHE calculation costs on the server of Design 2 are much greater than that of Design 1.

There are two general approaches to support the large computation depth needed in Design 2. First, we can set sufficiently large parameters to ensure correct decryption within a limited (but large) number of operations. Alternatively, we can reduce the noise by bootstrapping. Using the parameters for the larger number of operations deteriorates the performance of all the arithmetic operations, while each bootstrapping operation causes expensive overhead.

B. Implementation

Prior experimentation focused on the use of the BGV implementation in HElib, each in isolation. We implemented these designs in PALISADE and HElib, respectively, and perform experimental apples-to-apples comparisons on them for variations in configuration. As such, we implement these two designs for the privacy-preserving genome sequences search system with two FHE schemes, BGV provided by HElib [14] and BFV provided by PALISADE [11]. The Design 2 implementation with BFV is implemented without bootstrapping since PALISADE does not support it, while that with BGV is implemented with bootstrapping as proposed in the previous work [5].

VI. Experimentation

A. Problem settings

The genomic data used for this experiment are single-nucleotide polymorphism (SNP) [24] sequences from the 1,000 Genomes Project [2]. This data provide the representation of where variations from a reference genome are likely to appear, without showing entire genome sequences. In our experimental setting, the number of genomic data samples is 512 and the number of characters per sample is 10,000.

We experimented with query strings from 5 to 25. Also, clients designate 5 search positions which include dummy ones in order to improve security as mentioned in Sec. II.

B. System overview

We implemented the privacy-preserving genome sequences search system in two designs and two schemes discussed in Sec. V in C++. Both systems adopt the Chinese Reminder Theorem (CRT) packing technique by Smart et al. [23]. Experiments were conducted on the machine that has the specification shown in Table I, and temporary parameters used for these experiments are summarized in Table II.

TABLE I
EXPERIMENTAL ENVIRONMENT

Server	OS	CentOS 6.9
	CPU	Intel®Xeon®Processor E5-2643 v3 (3.4GHz) 6 Cores × 2 Sockets
	Main Memory	512GB
	SSD	80GB
	HDD	2TB

TABLE II
PARAMETERS FOR THESE EXPERIMENTS

Scheme	Design	Parameter
BGV	Design 1	*level L = 23*
	Design 2	*level L = 8*
BFV	Design 1	*numAdds A = 10*
	Design 2	*numAdds A = 50*

C. Experimentation Results

We ran each experiment three times and calculated the average of each execution time.

Fig. 5 and Fig. 6 show each graph of the average execution time of the main calculation on the server of Design 1 and Design 2 shown in Fig. 3 and Fig. 4, based on the length of the query. Fig. 7 shows the average execution time of Design 2 with BGV provided by HElib in detail, classified by each execution time for bootstrapping, other calculations, and initialization. Table III shows the calculation time of bootstrapping as the percentage of the whole main calculation time.

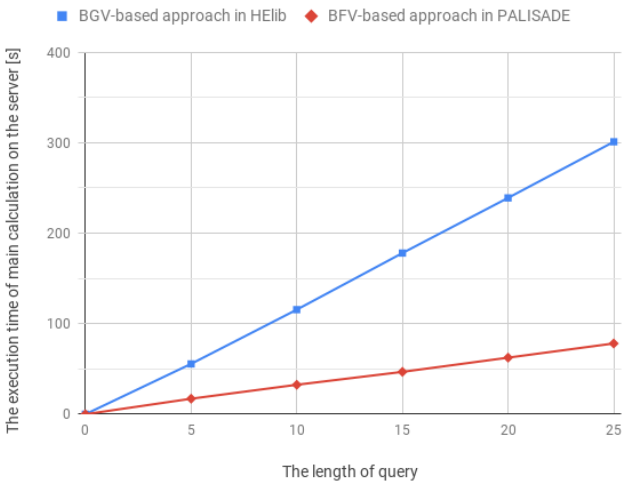

Fig. 5. Average execution time of the main calculation on the server of design 1 by the length of query

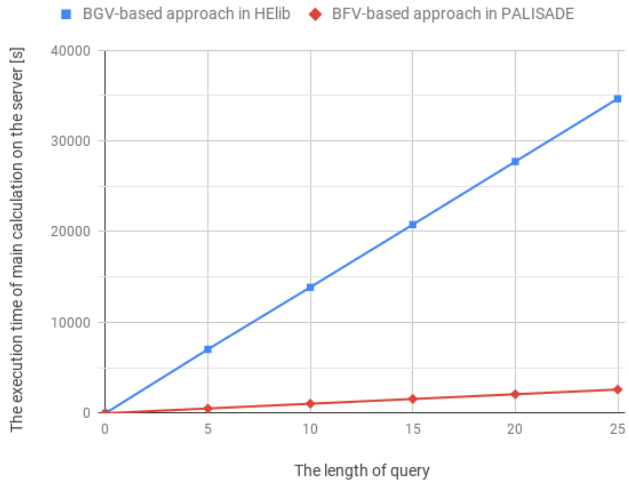

Fig. 6. Average execution time of the main calculation on the server of design 2 by the length of query

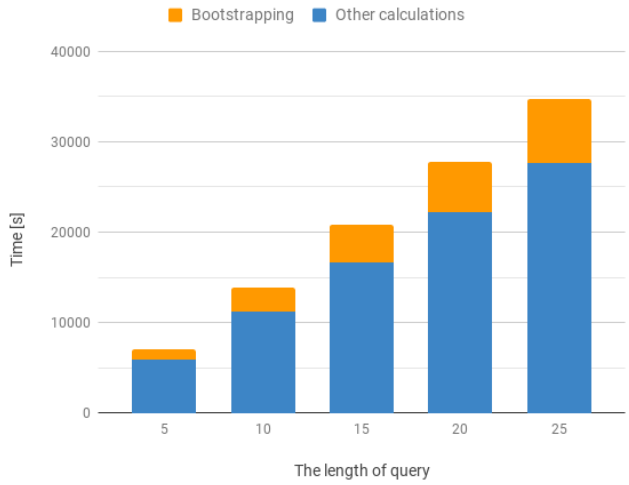

Fig. 7. Average execution time on the server of design 2 with BGV by the length of query

VII. ANALYSIS OF EXPERIMENTAL RESULTS

A. Scheme

First, we compare two scheme implementations: BGV by HElib and BFV by PALISADE. Fig. 5 shows two things; BFV by PALISADE is faster than BGV by HElib for Design 1, and the gap between the execution time with PALISADE and that of HElib widens as the length of query increases for both design. This means BFV by PALISADE supports improved performance for each arithmetic operations included in every loop of processing than BGV by HElib does. Fig. 6 shows similar results as in Fig. 5, but with a larger difference in execution time between that of BFV in PALISADE and that of BGV in HElib for Design 2 than that in Design 1. We believe this is the case because the execution time of the main calculation on the server of Design 2 with BGV increases rapidly.

TABLE III
THE RATIO OF THE CALCULATION TIME OF BOOTSTRAPPING TO THE EXECUTION TIME OF WHOLE MAIN CALCULATION

The length of query	The ratio of the calculation time of bootstrapping
5	0.16698
10	0.18987
15	0.19698
20	0.20099
25	0.20312

B. Bootstrap

Second, we look at the result shown in Fig. 6 into more detail. Fig. 7 and Table III focus on the details of the execution time of the design 2 with BGV. They show that the calculation time of bootstrapping accounts for a certain percentage of the whole main calculation time, and it grows as the length of query increases.

As mentioned in Sec. V-A, it is required to set the sufficient parameter that allows the larger number of operations without bootstrapping or to reset the noise in ciphertexts with bootstrapping in order to avoid the incorrect decryption because of the random noise in ciphertexts and its threshold. While using the parameters for the larger number of operations makes the performance of all the arithmetic operations worse, we can consider that the overhead caused by bootstrapping is more expensive than that caused by using the parameters for the larger number of operations as shown in Fig. 6.

C. Practicality

The execution time on the server of Design 1 with PALISADE is around 80 seconds when the length of the query is 25. Given that string search is generally a batch operation, this runtime would be considered acceptable time for practical use. Contrary to the execution time on the server of Design 1 shown in Fig. 5, that of Design 2 with BFV provided by PALISADE is over 500 seconds even when the query length is 5 according to Fig. 6. Design 2 runtime increase to around 2,600 seconds when the length of the query is 25. To put this into practical use, further acceleration would be needed.

VIII. CONCLUSION AND DISCUSSION

In this paper, we implemented and compared two designs for the client-server style system for privacy-preserving genome sequences search with two homomorphic encryption schemes, BGV provided by HELib and BFV provided by PALISADE, on the basis of prior work. Our results show that using the BFV-based approach in PALISADE provides optimal results for both designs of this application over our sample data. Though BFV-based approach shows better performance in this research, the parameters used for this experiment are not well-tuned. We have to consider which kind of parameters we should use and what value is appropriate for the system, and adjust the parameters in accord with the length of the query. It is necessary to compare BGV-based approach and BFV-based approach without bootstrapping as regards design 2 as well. As future work, we also plan to compare transfer size as well as execution time.

REFERENCES

[1] National Human Genome Research Institute - All About The Human Genome Project : https://www.genome.gov/10001772/all-about-the--human-genome-project-hgp/, Accessed in 2019-02

[2] IGSR - The International Genome Sample Resource : http://www.internationalgenome.org/, Accessed in 2019-02

[3] Kana Shimizu, Koji Nuida, Gunnar Rtsch "Efficient Privacy-Preserving String Search and an Application in Genomics," bioRxiv 018267; doi: urlhttps://doi.org/10.1101/018267

[4] Yu Ishimaki, Kana Shimizu, Koji Nuida and Hayato Yamana : "Poster: Privacy-Preserving String Search for Genome Sequences using Fully Homomorphic Encryption," Proc. of the 37th IEEE Symp. on Security and Privacy, 2016.5

[5] Yu Ishimaki, Hiroki Imabayashi, Kana Shimizu and Hanato Yamana : "Privacy-preserving string search for genome sequences with FHE bootstrap optimization," 2016 IEEE International Conference on Big Data (Big Data), Washington, DC, 2016, pp. 3989-3991.

[6] Richard Durbin : " Efficient haplotype matching and storage using the Positional Burrows-Wheeler Transform (PBWT)," Bioinformatics, vol. 30, no. 9, pp. 1266-1272, 2014.

[7] Yuri Yamamoto and Masato Oguchi : "A Decentralized System of Genome Secret Search Implemented with Fully Homomorphic Encryption," 2017 IEEE International Workshop on BigData and IoT Security in Smart Computing (BITS 2017) in conjunction with IEEE International Conference on Smart Computing (SMARTCOMP), Hong Kong, 2017, pp. 1-6.

[8] Junfeng Fan and Frederik Vercauteren : "Somewhat Practical Fully Homomorphic Encryption," Cryptology ePrint Archive, Report 2012/144: https://eprint.iacr.org/2012/144

[9] Ronald K. Rivest, Len Adleman, Michael L. Dertouzos : "On Data Banks and Privacy Homomorphisms," Foundations of Secure Computation, Academia Press, 1978

[10] Craig Gentry : "Fully Homomorphic Encryption Using Ideal Lattices," Proceedings of the forty-first annual ACM symposium on Theory of computing, 2009

[11] PALISADE : https://git.njit.edu/palisade/PALISADE, Accessed in 2019-02

[12] Wen-Jie Lu, Shohei Kawasaki, Jun Sakuma : "Using Fully Homomorphic Encryption for Statistical Analysis of Categorical, Ordinal and Numerical Data," Proceedings of The Network and Distributed System Security Symposium, 2017

[13] Michael Naehrig, Kristin Lauter, and Vinod Vaikuntanathan : "Can homomorphic encryption be practical?," Cloud computing security workshop (CCSW), ACM, New York, 2011, pp. 113-124

[14] HElib : http://shaih.github.io/HElib/index.html, Accessed in 2019-02

[15] Zvika Brakerski and Craig Gentry and Vinod Vaikuntanathan : "Fully Homomorphic Encryption without Bootstrapping," Cryptology ePrint Archive, Report 2011/277: https://eprint.iacr.org/2011/277

[16] Cheon, J. H., Kim, A., Kim, M., Song, Y : "Homomorphic encryption for arithmetic of approximate numbers," International Conference on the Theory and Application of Cryptology and Information Security, Springer, Cham, 2017, pp. 409437

[17] Adriana Lopez-Alt and Eran Tromer and Vinod Vaikuntanathan : "On-the-Fly Multiparty Computation on the Cloud via Multikey Fully Homomorphic Encryption," Cryptology ePrint Archive, Report 2013/094: https://eprint.iacr.org/2013/094

[18] Damien Stehle and Ron Steinfeld : "Faster Fully Homomorphic Encryption," Cryptology ePrint Archive, Report 2010/299: https://eprint.iacr.org/2010/299

[19] Microsoft SEAL : https://www.microsoft.com/en-us/research/project/microsoft-seal/, Accessed in 2019-02

[20] TFHE : https://tfhe.github.io/tfhe/, Accessed in 2019-02

[21] PBWT : https://github.com/richarddurbin/pbwt, Accessed in 2019-02

[22] PBWT-sec : https://github.com/iskana/PBWT-sec, Accessed in 2019-02

[23] N. P. Smart and F. Vercauteren : "Fully homomorphic SIMD operations," Designs, Codes and Cryptography, Vol. 71, No. 1, pp. 57-81, 2014.

[24] Genetics Home Reference - What are single nucleotide polymorphisms (SNPs)? : https://ghr.nlm.nih.gov/primer/genomicresearch/snp