Privacy-Preserving String Search for Genome Sequences with FHE bootstrapping optimization

Yu Ishimaki*, Hiroki Imabayashi*, Kana Shimizu*, Hayato Yamana*
* *Waseda University, Tokyo Japan*
Email: yuishi@yama.info.waseda.ac.jp, imabayashi@yama.info.waseda.ac.jp, shimizu.kana@waseda.jp, yamana@yama.info.waseda.ac.jp

***Abstract*—Privacy-preserving string search is a crucial task for analyzing genomics-driven big data. In this work, we propose a cryptographic protocol that uses Fully Homomorphic Encryption (FHE) to enable a client to search on a genome sequence database without leaking his/her query to the server. Though FHE supports both addition and multiplication over encrypted data, random noise inside ciphertexts grows with every arithmetic operation especially multiplication, which results in incorrect decryption when the noise amount exceeds its threshold called *level*. There are two approaches to avoid the incorrect decryption: one is setting the sufficient *level* that assures correct decryption within the limited number of operations, and the other is resetting the noise by the method called *bootstrapping*. It is important to find an optimal balance between overhead caused by the *level* and overhead caused by the *bootstrapping*, since using higher *level* deteriorates the performance of all the arithmetic operations, while the more number of *bootstrappings* causes more expensive overhead. In this study, we propose an efficient approach to minimize the number of *bootstrappings* while reducing the *level* as much as possible. Our experimental result shows that it runs at most 10 times faster than a naïve approach.**

***Keywords*- Fully Homomorphic Encryption (FHE); Genome Sequence; PBWT; String Search; Bootstrapping;**

I. Introduction

Privacy-preserving string search for genome sequence is indispensable in medical genomics. Since search results are often used for downstream analyses such as computing statistics, it is more practically efficient to design the protocol by using Fully Homomorphic Encryption (FHE). To keep away from the incorrect decryption due to the noise growth, we have to set the *level* properly (here, we assume L), or both to set the *level* to D where $D < L$ and to execute *bootstrappings* [1] that reset the noise before the noise amount exceeds D. The computational cost of all the arithmetic operations increases in proportion to the *level*, besides the *bootstrapping* requires huge computational cost. Therefore, it is important to find an optimal balance between the overhead caused by using the higher *level* and that caused by using the *bootstrapping*.

In this paper, we propose a one-round string search protocol that minimizes the number of *bootstrappings* while reducing the *level* as much as possible.

II. Related Work

Several works have been done to realize privacy-preserving (sub)string search protocol for genome sequences. PBWT-sec [2] is an efficient two-party prefix match counting protocol that combines both additively homomorphic encryption (AHE) which supports only addition over encrypted data, and an efficient searchable data structure called positional-Burrows Wheeler Transform (PBWT) [3]. PBWT enables a client to search substring match with an aligned genome sequence database on a server. In its searching phase, the client should access to a look-up vector derived from PBWT recursively, i.e., named recursive oblivious transfer (ROT) [2]. When the query string length is ℓ, ROT consists of ℓ times vector-lookups, which needs ℓ rounds communication between the client and the server. Since PBWT-sec is built on AHE, searching with wildcards and computing statistics based on the search result cannot be realized.

In order to support such functionalities, our previous work [4] adopts FHE as a building block of PBWT-sec following the same model. Since it still requires ROT, many communications are indispensable when the length of query string becomes long despite the data size of FHE becomes larger than that of AHE. Since each communication involves large data transfer, ROT is inappropriate for the client with small-computation power. Therefore one-round protocol can be considered when adopting FHE in PBWT-sec.

III. Method

Our method is based on our previous works [2] [4] where a server has a PBWT that is transformed from an original aligned genome sequences database, and a client has a query consisting of an initial searching column parameter (sc) and encrypted search strings whose length is ℓ. In the searching phase, the server creates a Look-up Table(hereafter, LUT) from PBWT in advance, and the client sends the query to the server. Here, sc is not encrypted. Then the server extracts a set of look-up vectors from the columns of LUT started with column sc. After that, the server counts the number of exact match by recursively matching each string of the query by accessing the corresponding element of look-up vectors where the length of each look-up vector is the same n. On the server side, these look-up vectors and LUT are not encrypted. The matched number is computed by $\boldsymbol{v}^{i+\ell}_{c_{\ell-1}}[\ldots\boldsymbol{v}^{i+1}_{c_1}[\boldsymbol{v}^{i}_{c_0}[n-1]]\ldots] - \boldsymbol{v}^{i+\ell}_{c_{\ell-1}}[\ldots\boldsymbol{v}^{i+1}_{c_1}[\boldsymbol{v}^{i}_{c_0}[0]]\ldots]$, where $\boldsymbol{v}^{i}_{c_j}$ $(1 \leq i \leq M, 0 \leq j \leq \ell-1)$ represents the look-up vector for the i-th column of the j-th string of the query, and M represents the total number of columns in the database. Finally the server returns the existence of match

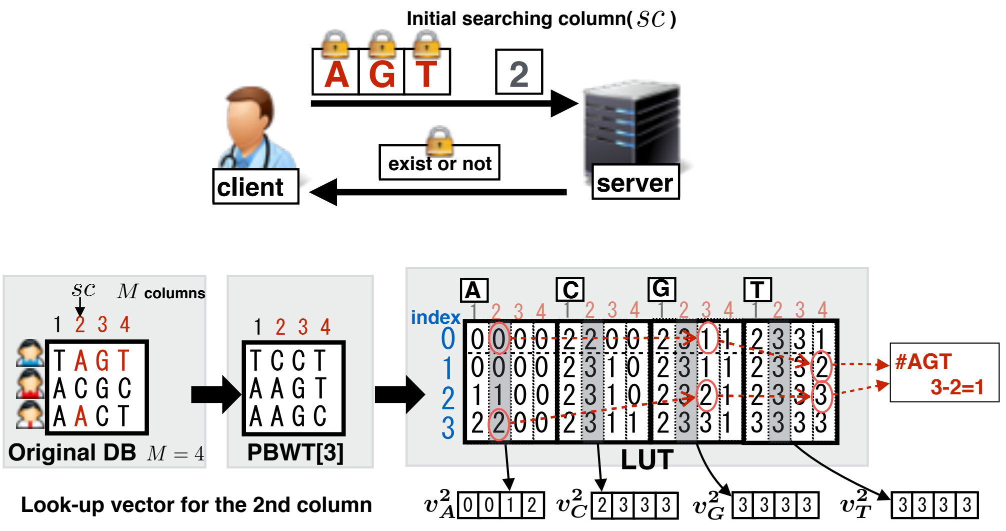

Figure 1. Overview of our protocol

between the query string and the database back to the client in encrypted form. We describe the overview of our protocol in Fig.1.

In order to obtain a specific element of the vector in a privacy preserving manner, we have to conduct equality checks [5] that include multiplications, which causes a large growth of noise. As shown in Fig.2(a), a naïve implementation is to place *bootstrapping* after every multiplicative operation, which results in increasing the number of *bootstrappings*. Thus, the way to place *bootstrapping* is crucial to speed-up. In our optimization, *bootstrapping* is placed at a merging node of data-flow-graph so that its number becomes small as shown in Fig.2(b).

Let us compare the proposed method to two naïve methods: 1) the method that conducts *bootstrapping* after every multiplicative operation with setting the *level* low, 2) the method that adopts the sufficient *level* so that no *bootstrapping* is needed. The time complexity of 1) is $O(D'g+B\ell n)$ where D' is the *level*, g is the total number of operations except the *bootstrapping* and B is the overhead caused by one *bootstrapping*, and the time complexity of 2) is $O(Lg)$ where L is the *level* enough to require no *bootstrappings* which is proportion to ℓ. Meanwhile, the time complexity of the proposed method is $O(Dg+B\ell)$ where $D << L$. Among the three methods, the proposed method is able to have the smallest time complexity when ℓ becomes large, since $D' < D << L$ and $n \propto$ #sample in the database whereas $g \propto n$.

IV. Experimental Result

We implemented our proposed algorithm over HElib [1]. In the experiment, we used an artificial dataset whose vector length is 200. The experiment was run on the server equipped with Intel Xeon E5-2643 v3 @ 3.40 GHz CPU with 4 threads and 512 GB RAM on the client side, and was run on the server equipped with Intel Xeon E7-8880 v3 @ 2.30 GHz CPU with 72 threads and 1TB RAM on the server side. The server and the client are connected under the same subnet via 10Gpbs Ethernet.

We compared execution time among the three methods by varying input query string length where the *level* used for each method is summarized in Table I. As shown in Fig. 3, the experimental result shows that our proposed method achieves at most 10 times faster than the method "naïve 2" whose *level* is defined so that no *bootstrapping* is required.

Table I
Level used for each method varying query string length

query string length ℓ / Method	1	5	10
naïve 1	15	15	15
naïve 2	9	37	72
proposed	23	23	23

V. Conclusion and Discussion

In this paper, we proposed a one-round privacy-preserving genome sequences search protocol by using FHE with *bootstrapping* optimization, i.e., minimizing the number of *bootstrappings* and reducing the *level* as much as possible. The experimental result showed that our proposed method achieved the fastest performance. The difference of execution time between the naïve methods and our proposed method becomes large when we need longer query string length ,i.e, >10. Our future work includes extending search functionality such as wildcard search.

Acknowledgment

This work was supported by the CREST, JST.

[1]https://github.com/shaiH/HElib, Accessed on 2016-10-1.

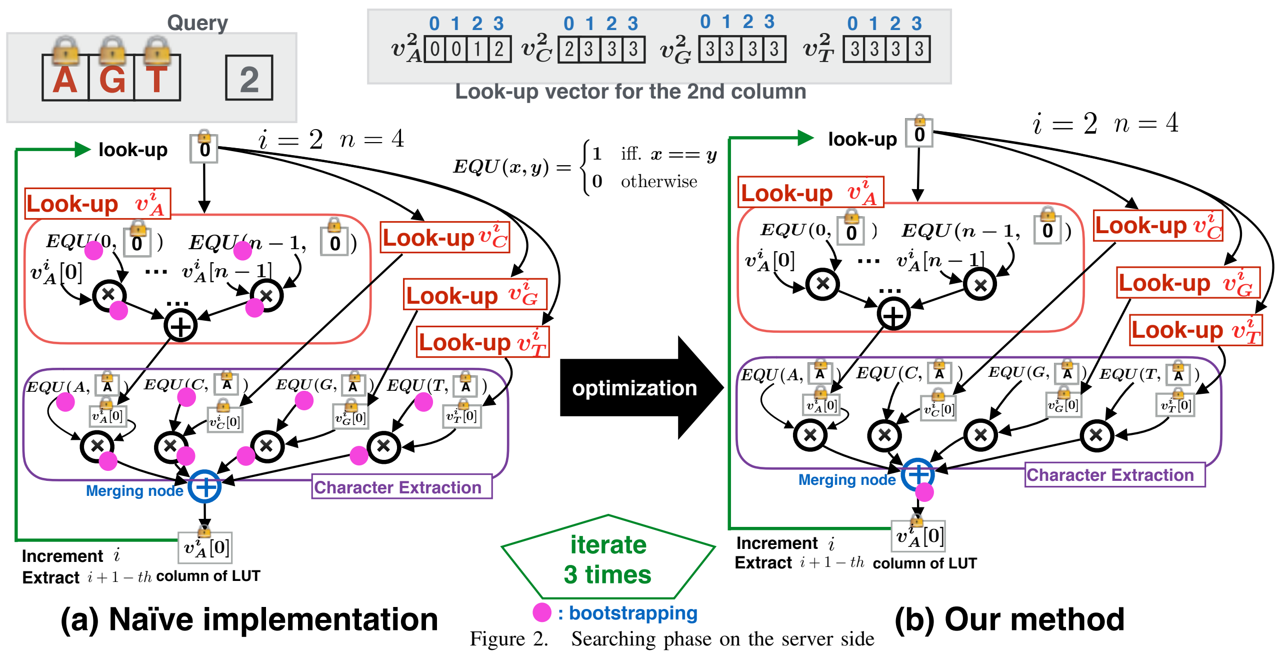

Figure 2. Searching phase on the server side

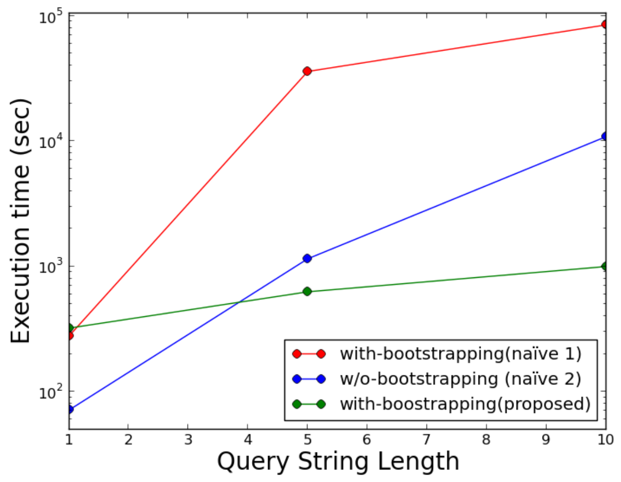

Figure 3. Execution time comparison varying query string length

References

[1] C. Gentry, "Fully Homomorphic Encryption Using Ideal Lattices," In Proc. of STOC 2009, pp. 169-178, 2009.

[2] K. Shimizu, K. Nuida, and S. Rätsch, "Efficient Privacy-Preserving String Search and an Application in Genomics," Bioinformatics, vol. 32, no. 11, pp. 1652-1661, 2016.

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

[4] Y. Ishimaki, K. Shimizu, K. Nuida, and H. Yamana, "Poster: Privacy-Preserving String Search for Genome Sequences using Fully Homomorphic Encryption," the 37th IEEE Symposium on Security and Privacy, 2016.

[5] J. H. Cheon, M. Kim, and M. Kim, "Optimized search-and-compute circuits and their application to query evaluation on encrypted data," IEEE Transactions on Information Forensics and Security, vol. 11, no. 1, pp. 188-199, 2016.