Poster: Privacy-Preserving String Search for Genome Sequences using Fully Homomorphic Encryption

Yu Ishimaki*, Kana Shimizu*†, Koji Nuida†‡, Hayato Yamana*
* Waseda University, Tokyo Japan
† Information Technology Research Institute, National Institute of Advanced Industrial Science and Technology, Tokyo, Japan
‡ Japan Science and Technology Agency (JST) PRESTO Researcher, Tokyo, Japan
Email: yuishi@yama.info.waseda.ac.jp, shimizu.kana@waseda.jp, k.nuida@aist.go.jp, yamana@yama.info.waseda.ac.jp

***Abstract*— Privacy protection for personal genome analyses is one of the emerging issues in the area of medical genomics. We assume the general case where a user would like to compute some sort of score based on a similarity search between a query and a database. In this study, we propose a cryptographic protocol using fully homomorphic encryption (FHE) that enables searching on genome sequences and still allows computation of the score based on the search result. In our method, we use the state-of-the-art technique that combines the recursive oblivious transfer and the efficient discrete data structure that allows linear time searching. In an experiment using a data set created from the 1,000 genome project data, it took 535 seconds for searching a query on 2,184 personal genomes. Though FHE is highly inefficient, our proposed optimization method reduces the runtime, which is only approximately 10 times longer than the previous scheme. This may be considered in some applications as an allowable trade-off with the fully enhanced subsequent functionality.**

I. Introduction

Personal genome analyses is one of the most important topics in current medical genomics and protection of an individual's privacy poses many technical challenges. Here we consider a general case where a user computes statistics based on a personal genome database search. In this case, both the user's query and the database contents should be kept in private except that only the user knows the statistics. In order to facilitate designing a secure protocol for such a case, we propose a secure genome search protocol whose output is an encrypted count of similar genome sequences and the count is easily used as an input to another function for computing some sort of score.

Among previous works, PBWT-sec [1] is an efficient secure protocol for counting prefix match in a large collection of aligned genome sequences. The output of the protocol is encrypted by homomorphic encryption (HE), which can thus basically be reused as an input for subsequent secure computations. However, since the previous work uses additively HE, the subsequent functionality is very limited, e.g., ordinary statistical analysis is not available.

To resolve the restriction, here we propose to use fully HE (FHE) as an alternative building block. The main contribution of this paper is that our proposed method can support both addition and multiplication in encrypted form to realize statistical analysis.

A portion of this work was supported by CREST, JST.

II. Previous Work

PBWT-sec[1] is an efficient two-party protocol for privacy-preserving genome sequences search. The protocol assumes the following model: A server holds a set of aligned genome sequences and a user holds a genome sequence. After the computation, only the user can obtain the count of prefix match larger than a given threshold without knowing any other information about server's sequences.

The key idea of the PBWT-sec is to combine the positional Burrows-Wheeler Transform (PBWT) [2] and the recursive oblivious transfer [1]. The PBWT is a data structure to search substring match for a set of aligned genome sequences where a (k+1)-gram match is found in constant time by computing a function with the previous k-gram match as an input. During the search, the match is reported as an interval $[f,\ g]$ and f and g is updated upon extending the match. It is known that the update task of f and g is replaced by looking up a lookup-vector that stores the values obtained by computing the function in advance. Thus one can obtain the prefix match by computing $v[\dots v[f_0]\dots]$ and $v[\dots v[g_0]\dots]$ where $\boldsymbol{v}$ is denoted as the vector and $[f_0,\ g_0]$ is an initial interval.

The recursive oblivious transfer protocol enables the user to repeat oblivious transfer in such a way where the user inputs the output of the previous search but the user obtains only the last output and all the other intermediate outputs are invisible to the user. In PBWT-sec, the user obtains the elements of the lookup-vector to find the subsequent interval, i.e., extending a match by a letter by using recursive oblivious transfer and obtains count of the match by computing $g - f$.

III. Method

We design the protocol based on the same idea to PBWT-sec by adopting FHE. To reduce both runtime and communication size, we focus on Smart et al.'s packing technique [3] which enables encrypting an integer vector into one ciphertext. Besides that, addition and multiplication can be conducted on the ciphertext via element-wise computation on a plain-text vector (each element is called a slot).

We assume that the server has an n-length vector $\boldsymbol{v}$, and prepares the two same-length vectors $\boldsymbol{v_0}$ and $\boldsymbol{v_1}$. Here, $\boldsymbol{v_0} = \boldsymbol{v}/\ell$ and $\boldsymbol{v_1} = \boldsymbol{v} \bmod \ell$, where ℓ is the number of slots ($\ell \geq \sqrt{n}$). Moreover, the server divides them into ℓ sub vectors, all having a length of ℓ. Here, the j-th sub vectors of $\boldsymbol{v_0}$ and $\boldsymbol{v_1}$ are denoted by $\boldsymbol{v_{0j}}$ and $\boldsymbol{v_{1j}}$, respectively.

Our proposed algorithm is described in the following 3 steps. This is repeated by the length of the user's query string, e.g., 4 times when the query string is "ATTG".

Step.1 User's query construction

The user constructs two query bit-vectors $\boldsymbol{q_{t_0}}$ and $\boldsymbol{q_{t_1}}$. If he wants to know the t-th element of the server's vector $\boldsymbol{v}$, he sets $t_0 = t/\ell$ and $t_1 = t \bmod \ell$ such that $t = t_0\ell + t_1$.

$$q_{t_0}[i] = \begin{cases} 1 & (i = (t_0 + t_1)_{\bmod \ell}) \\ 0 & (i \neq (t_0 + t_1)_{\bmod \ell}) \end{cases} \quad (0 \leq i \leq \ell - 1) \tag{1}$$

$$q_{t_1}[i] = \begin{cases} 1 & (i = t_1) \\ 0 & (i \neq t_1) \end{cases} \quad (0 \leq i \leq \ell - 1) \tag{2}$$

Then, the user encrypts them using Smart et al.'s packing technique, and sends $\mathsf{Enc}(\boldsymbol{q_{t_0}})$ and $\mathsf{Enc}(\boldsymbol{q_{t_1}})$ to the server, where they are used as the ciphertexts of $\boldsymbol{q_{t_0}}$ and $\boldsymbol{q_{t_1}}$, respectively.

Step 2. Server's calculation

The server obtains $\mathsf{Enc}(\hat{\boldsymbol{q}}_{\boldsymbol{t_0}}) = \mathsf{Perm}(\mathsf{Enc}(\boldsymbol{q_{t_0}}), (r'_0 + r'_1)_{\bmod \ell})$ and $\mathsf{Enc}(\hat{\boldsymbol{q}}_{\boldsymbol{t_1}}) = \mathsf{Perm}(\mathsf{Enc}(\boldsymbol{q_{t_1}}), r'_1)$, where $\mathsf{Perm}(\mathsf{Enc}(\boldsymbol{q}), r)$ denotes the permutation of $\mathsf{Enc}(\boldsymbol{q})$ whose j-th slot moves to $(j - r)_{\bmod \ell}$. Here, r'_0 and r'_1 are initialized by 0 only at the first iteration. The server then generates random values r_0 and r_1, and calculates c_0 and c_1.

$$\begin{aligned} c_0 &= \textstyle\bigoplus_{i=0}^{\ell-1} \mathsf{Enc}(\hat{\boldsymbol{q}}_{\boldsymbol{t_1}}) \otimes (\boldsymbol{v_{0i}}[i] + r_0)_{\bmod \ell} \otimes \mathsf{Perm}(\mathsf{Enc}(\hat{\boldsymbol{q}}_{\boldsymbol{t_0}}), i) \\ c_1 &= \textstyle\bigoplus_{i=0}^{\ell-1} \mathsf{Enc}(\hat{\boldsymbol{q}}_{\boldsymbol{t_1}}) \otimes (\boldsymbol{v_{1i}}[i] + r_1)_{\bmod \ell} \otimes \mathsf{Perm}(\mathsf{Enc}(\hat{\boldsymbol{q}}_{\boldsymbol{t_0}}), i) \end{aligned} \tag{3}$$

The server sends $\hat{c_0} = \mathsf{Perm}(c_0, -r'_1)$ and $\hat{c_1} = \mathsf{Perm}(c_1, -r'_1)$ to the user. Subsequently, the server sets $r'_0 = r_0$ and $r'_1 = r_1$ for the next iteration.

Step 3. User's decryption

The user obtains $a_0 = \mathsf{Dec}(\hat{c}_0, t_1)$ and $a_1 = \mathsf{Dec}(\hat{c}_1, t_1)$, where $\mathsf{Dec}(\hat{c}, t')$ denotes the t'-th element of the vector obtained by decrypting $\hat{c}$. He then sets $t_0 = a_0$ and $t_1 = a_1$ for the next iteration.

IV. Experimental Result and Discussion

We have implemented our proposed algorithm by adopting HElib[4] and compared it to PBWT-sec by using PBWT-sec open-source library[5]. In the experiment, we used 2,184 haploid genome sequences from Phase 1 of the 1,000 Genomes Project [6]. The experiment was run on the server equipped with Intel Core-i7 3.40 GHz CPU with four threads and 16 GB RAM on the user side, and on the server equipped with Intel Xeon E7-8880 v3 2.30 GHz CPU with 8 threads and 1TB RAM on the server side. The user's query string had a length of five. We set plain text space as 2^{20}, the depth of evaluation as 10, the lattice dimension as 12,000, the number of slots as 600, and security parameter as 128 in HElib.

The experimental results of our proposed algorithm and PBWT-sec are shown in Fig.1 and Fig.2 whose x-axis represents the length of the server's vector $\boldsymbol{v}$. The results are averaged by five times experiments. Fig.1 shows the runtime consisting of key generation time on the user. It takes approximately 20 seconds for our proposed algorithm and 0.012 seconds for PBWT-sec. The runtime of our proposed algorithm

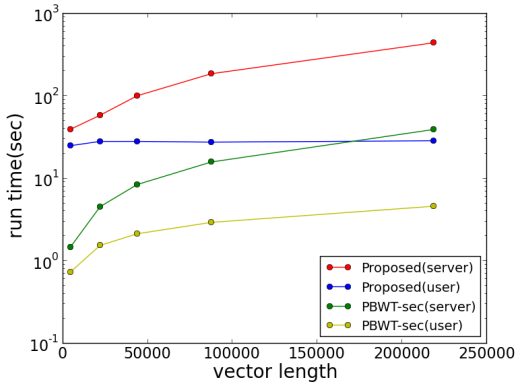

Fig. 1. Runtime of our proposed algorithm and PBWT-sec.

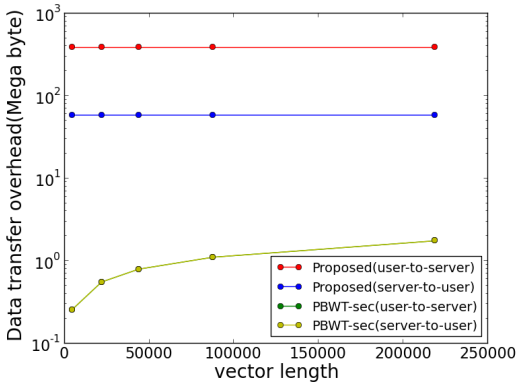

Fig. 2. Data transfer overhead of our proposed algorithm and PBWT-sec.

on the server is approximately 10 times longer than that of PBWT-sec. Fig.2 shows the data transfer overhead. The data transfer overhead of our proposed algorithm remains constant, because the number of ciphertexts sent to both the user and the server is fixed.

V. Conclusion

In this paper, we designed a privacy-preserving genome sequences search method that is build on fully homomorphic encryption, based on the same design principle used for PBWT-sec which is build on additively homomorphic encryption. Though the performance of the proposed method is rather lower, it is still acceptable level for some applications. Our method can be extended to the computation method of a complex statistics such as disease risks. We plan to extend our method to support wild card searches and statistical analysis.

References

[1] K. Shimizu, K. Nuida and S. Rätsch: *Efficient Privacy-Preserving String Search and an Application in Genomics*, Bioinformatics, doi:10.1093/bioinformatics/btw050, 2016.

[2] R. Durbin: *Efficient haplotype matching and storage using the Positional Burrows-Wheeler Transform (PBWT)*, Bioinformatics, Vol. 30, No. 9, pp. 1266-1272, 2014.

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

[4] HELib. http://shaih.github.io/HElib/, accessed on 2016-4-1.

[5] PBWT-sec. https://github.com/iskana/PBWT-sec/, accessed on 2016-4-1.

[6] The 1000 Genome Project Consortium:*An integrated map of genetic variation from 1,092 human genomes*, Nature, Vol. 491, pp. 56-65, 2012.