DNA Similarity Search With Access Control Over Encrypted Cloud Data

Guowen Xu, *Student Member, IEEE*, Hongwei Li, *Senior Member, IEEE*, Hao Ren, *Student Member, IEEE*, Xiaodong Lin, *Fellow, IEEE*, and Xuemin Shen, *Fellow, IEEE*

Abstract—DNA similarity search has been widely applied in human genomic studies including DNA marking, genomic sequencing and genetic disease prediction. Meanwhile, with the explosive growth of data, users are increasingly inclining to store DNA data on the cloud for saving local cost. However, the high sensitivity of DNA data has forced the government to strictly control its acquisition and utilization. One potential solution is to encrypt DNA data before outsourcing them to the cloud. Nevertheless, private DNA similarity query has been an active research issue, state-of-the-art results are still defective in security, functionality, and efficiency. In this article, we propose EFSS, an efficient and fine-grained similarity search scheme over encrypted DNA data. In specific, first, we design an approximation algorithm to efficiently calculate the edit distances between two sequences. Second, we put forward a novel Boolean search strategy to achieve complicated logic queries such as mixed "AND" and "NO" operations on genes. Third, data access control is also supported in our EFSS through a variant of polynomial based design. Moreover, the K-means clustering algorithm is exploited to further improve the efficiency of execution. In the end, security analysis and extensive experiments demonstrate the high performance of EFSS compared with existing schemes.

Index Terms—DNA similarity search, privacy-preserving, fine-grained query, access control

1 Introduction

DNA similarity search has been a rich area of research with promising medical and healthy applications. Examples include similar patient query [1], [2], disease prediction [3], [4] and genome sequencing [5]. For example, genes BRCA1 and BRCA2 have been considered to be closely related to the risk of breast cancer in individuals. Thus, it will be gratifying that this disease can be detected and prevented in advance, if we perform DNA similarity detection between the patient population and the potential disease population. Currently, in response to the explosive growth of data (including DNA data), Public Clouds, such as DNAnexus and Google Genomics,[1] have been increasingly approved and used in outsourced data services due to their powerful superiority in computing and storage. However, large scale data storage and processing over raw DNA data entail security and privacy risks [4], [5], [6]. One real risk is that the cloud server may abuse and reveal users' privacy since it has full data access authority. To address this challenge, an intuitive approach is to encrypt the whole DNA data utilizing the conventional encryption method (e.g., AES [7]) before outsourcing them to the cloud. However, this will significantly reduce data availability and make it difficult to perform DNA similarity queries [8], [9].

To address this challenge, many scholars have thrown themselves into the secure DNA similarity query study and proposed some remarkable results [10], [11]. For example, Wang *et al.* [8] design a system called GENSETS, a genome-wide, privacy-preserving system to actualize private DNA similarity query with high accuracy and efficiency. Based on the secure multi-party computation, Gilad Asharov *et al.* [9] also propose a DNA similarity query model to address the problem of high divergence among individual genomes. Recently, Wang *et al.* [12] design a secure DNA query scheme with only one round of communication throughout the querying process, and provide strong security guarantees in both data privacy and the search pattern. However, these approaches are only devised from the perspective of efficiency, and most of them require all parties remaining online during the entire querying process. Moreover, other common and significant performance indicators, especially data access control and personalized search requirements (i.e., complex logic queries) are rarely taken into account. Obviously, access control and Boolean queries for genetic data have been widely used in various medical scenarios. Consider a common scenario where doctors visit the genetic database of the hospital they work for, doctors in different departments often only have access to their authorized data. For example, the attending doctors of breast cancer probably

1. https://cloud.google.com/genomics/

• *G. Xu, H. Li, and H. Ren are with the School of Computer Science and Engineering, University of Electronic Science and Technology of China, Chengdu 611731, China, and also with the Cyberspace Security Research Center, Peng Cheng Laboratory, Shenzhen 518000, China. E-mail: guowen.xu@foxmail.com, hongweili@uestc.edu.cn, renhao.uestc@gmail.com.*
• *X. Lin is with the School of Computer Science, University of Guelph, Guelph, ON N1G 2W1, Canada. E-mail: xlin@wlu.ca.*
• *X. Shen is with the Department of Electrical and Computer Engineering, University of Waterloo, Waterloo, ON N2L 3G1, Canada. E-mail: sshen@uwaterloo.ca.*

Manuscript received 24 Mar. 2019; revised 31 Oct. 2019; accepted 17 Jan. 2020. Date of publication 23 Jan. 2020; date of current version 7 June 2022.
(Corresponding author: Hongwei Li.)
Recommended for acceptance by F. Douglis.
Digital Object Identifier no. 10.1109/TCC.2020.2968893

have not any access rights except breast cancer patient's DNA sequences. Besides, sometimes we do not need to query over entire DNA collections, where specific DNA sequences controlling particular genetic genes (such as genes controlling physical characteristics, mammary cancer, diabetes, etc.) should be the focus of our attention. Therefore, it is meaningful to propose a lightweight similarity search framework over encrypted DNA data, which can efficiently implement string matching between different DNA sequences while supporting the functions of data access control and the Boolean query simultaneously.

However, it is challenging to design a satisfactory solution that meets the above requirements. First, the length of a long gene sequence always ranges from hundreds of thousands to millions of base pairs [8]. This feature makes it possible to generate huge computation overhead if we directly exploit traditional techniques (such as keyword-based and string-matched searchable encryption) to perform DNA sequence matching under ciphertext. To alleviate this, recent works [9], [10] have resorted to the protocol of Secure Multiparty Computing (SMC) to reduce the computation overhead in the matching process. However, in addition to increasing the communication overhead of each participant, SMC based protocol always requires all parties remaining online during the entire querying process, which greatly limits the robustness of these solutions to sporadic events such as network delays and device damage. Second, although there have been some research results on data access control [13], [14], [15] and Boolean query [16], [17], [18], the heterogeneity of the application scenarios makes these solutions difficult to be directly adopted in DNA queries. Moreover, expressiveness functionality often leads to lower efficiency since the strong functional guarantees are never free. Therefore, it is also an intractable challenge to design a light-weight DNA similarity search framework that is highly supportive of data access control and Boolean query.

In this paper, we design EFSS, the first efficient and non-interactive DNA similarity search framework with data access control over encrypted cloud data. Specifically, we first propose an approximation algorithm to compute the edit distances between two sequences privately. Then, we present an efficient data access control strategy to manage the permissions of different users. In addition, by executing the custom query requests, our EFSS is supportive of mixed Boolean queries. In summary, our contributions can be summarized as follows:

- We introduce a private approximation algorithm to convert the edit distance computation problem to the symmetric set difference size approximation problem. Through this conversion, each DNA sequence will be compressed into a set of hash values, which can significantly reduce the number of elements that need to be matched under ciphertext.
- We design a novel Boolean search method to achieve the complicated logic query such as mixed "AND" and "NO" operations on genes. Moreover, since the search strategy can avoid unnecessary DNA sequences matched during the query process, users can also gain the benefit of receiving accurate results with a short period.

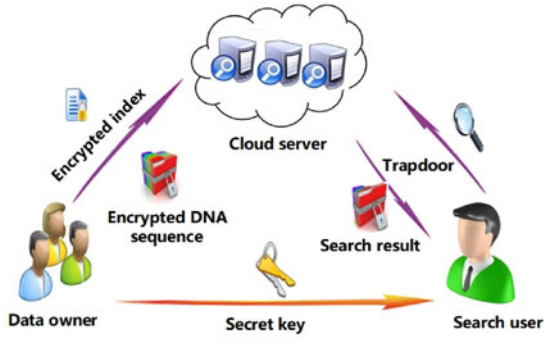


Fig. 1. System model.

- By utilizing a variant of polynomial based design, EFSS is able to support data access control during the query process, where each DNA sequence is accessible to users who are authorized. Besides, we adopt the K-means clustering algorithm to further improve the efficiency of our proposed scheme.
- We give a formal security analysis of our proposed model and stress that our EFSS is secure under the same-closeness-pattern chosen-plaintext attacks. Besides, extensive experiments conducted on real-world data also demonstrate the high efficiency of our proposals compared with existing schemes.

Different from the conference version [19], we propose two secure DNA similarity search models (i.e., EFSS_I and EFSS_II) to elaborate and achieve our purpose in a gradual way. Then, we put more effort into the analysis of security, functionality and performance evaluation. Note that EFSS leaks the "closeness" of queried messages (i.e., *Size Pattern* and *Access Pattern*) to better support efficient DNA similarity query. An optimal security notion for EFSS would be a natural relaxation of the standard IND-CPA security definition prohibiting queries that trivially exploit this leakage of closeness. We call this security as *indistinguishability under same-closeness-pattern chosen-plaintext attacks* (IND-CLS-CPA), which is adopted in [6], [10], [20]. In the end, we also polish the conference version to enhance the presentation and readability.

The remainder of this paper is organized as follows. In Section 2, we outline the system model, threat model and security requirements. Then, we describe the preliminaries of the proposed schemes in Section 3, and describe our EFSS in Section 4. Later, we carry out security analysis and performance evaluation in Sections 5 and 6, respectively. The related works are presented in Section 7. Finally, in Section 8, we conclude this paper.

2 System Model, Threat Model, and Security Requirements

2.1 System Model

As shown in Fig. 1, we consider three entities that are not colluding with each other in our system.

- *Data owner*: The data owner is usually considered to be organizations such as hospitals and Biopharmaceutical companies. Its primary task is to encrypt the raw DNA sequences before outsourcing them. Besides, to improve the search efficiency, each ciphertext is linked with an index. Then, the data owner submits both the encrypted DNA data and encrypted indexes to the cloud, as well as shares the secret key (used to generate the secure index) to the authorized users.

- *The cloud server*: The main responsibility of the cloud server is to store encrypted data and execute users' search requests in the ciphertext environment. Once the execution is completed, the encrypted results will be returned to authorized users.
- *Search user*: After receiving the secret key, the search user generates the trapdoors (i.e., the encrypted search request) and uploads them to the cloud server for obtaining the query results.

Note that in our scenario, the data owner only sends the key (independent of the key used to encrypt the raw DNA sequences) used to generate the encrypted index to the search user, since this is sufficient for achieving the function of secure DNA similarity query. Therefore, the data owner's raw DNA sequences will not be disclosed to any unauthorized entity in our model. On the other hand, due to the one-way property of encrypted index/trapdoor (analyzed in Section 5), the cloud server is impossible to recover the contents of indexes/trapdoors of other entities, even if it can impersonate a legitimate search user.

2.2 Threat Model and Security Requirements

The cloud server is considered to be "honest-but-curious" [10], [21], [22], which means that the cloud server strictly follows the pre-defined protocols such as querying, storage, computation, and other requests [23], [24]. However, it will also use the existing knowledge to infer the behavior of search user, and try to compromise the privacy of both the data owner and search users. Concretely, in our threat model, the cloud server can access the encrypted DNA data, indexes, and trapdoors, as well as require search users to generate new encrypted indexes/trapdoors to get extra information. However, it cannot do anything that violates the established protocols.

Besides, same as schemes [10], [21], [22], to concentrate on the topic of similarity search, we do not consider the secure key distribution between the data owner and search user, which can be addressed by utilizing traditional cryptography primitives such as secure multi-party computation [25], [26] and authentication technology [27], [28]. Interested readers can refer to literatures [29], [30], [31] for more details.

Based on the threat model, we define the security requirements as follows.

- *Confidentiality of DNA sequence*: Data owner's DNA data are usually outsourced to the cloud server before being utilized. Taking privacy issues into account, these data should be encrypted ahead and not leaked to anyone except the data owner and authorized search users.
- *Privacy protection of index and trapdoor*: Index and trapdoor are generated for querying conveniently. However, the leakage of index/trapdoor privacy has been shown to bring risks for adversaries to compromise user data privacy [32], [33]. Therefore, the privacy of the trapdoor and index should be protected well.

TABLE 1
Notations

Symbol	Description
Ref	**Ref** = $R = (r_1 r_2 \cdots r_{\mathcal{T}})$, denotes the public reference genome, where $r_i (i = 1, 2 \cdots \mathcal{T})$ represents a element of **Ref** (i.e., "A", "T", "C" or "G").
$\mathbf{A_i}$	A raw DNA sequence $(i = 1, 2 \cdots N)$.
$\tilde{\mathbf{A}}_i$	$\tilde{\mathbf{A}}_{\mathbf{i}} = (A_{i1}, A_{i2}, \ldots, A_{is})$, denotes the blocked $\mathbf{A_i}$, where $A_{im} (m = 1, 2, \ldots s)$ represents the mth subsequence of $\mathbf{A_i}$.
$\mathbf{A'_i}$	$\mathbf{A'_i} = (A'_{i1}, A'_{i2}, \ldots A'_{im}, \ldots A'_{is})$, denotes the set of editing operations of $\mathbf{A_i}$, where A'_{im} represents the set of editing operations of the mth subsequence A_{im}.
Q	a query request, i.e., a DNA sequence generated from the search user.
$H_{hr}(\mu)$	$H_{hr}(\mu) = \{-1, 1\}$, where H is a hash function and hr denotes the secret key. μ is a element belonged to a given set of editing operations.
$d_{\mathbf{A'_i}}$	The sum of all $H_{hr}(\mu), \mu \in \mathbf{A'_i}$.
$d_{A'_{im}}$	The sum of all $H_{hr}(\mu), \mu \in A'_{im}$.
$I^j_{\mathbf{A'_i}}$	$I^j_{\mathbf{A'_i}} = (d_{A'^j_{i1}}, d_{A'^j_{i2}}, d_{A'^j_{im}}, \ldots, d_{A'^j_{is}})$, denotes the encrypted index of $\mathbf{A_i}$ in our EFSS_I, where j represents the jth iteration.
$T^j_{\mathbf{Q'}}$	$T^j_{\mathbf{Q'}} = (d_{Q'^j_1}, d_{Q'^j_2}, d_{Q'^j_m}, \ldots, d_{Q'^j_s})$, denotes the trapdoor of **Q** in our EFSS_I, where j represents the jth iteration.
$I^j_{\mathbf{A_i}}$	The encrypted index of $\mathbf{A_i}$ in our EFSS_II, where j represents the jth iteration.
$T^j_{\mathbf{Q}}$	The trapdoor of **Q** in our EFSS_II, where j represents the jth iteration.

- *Unlinkability of trapdoors*: Unlinkability of trapdoors, means that if a DNA sequence is queried frequently, whose trapdoors should be indistinguishable since the cloud server can utilize the relationship between trapdoors to further infer the contents of user's DNA data. Thus, given a query request, a security-critical model should guarantee the indistinguishability of all generated trapdoors.

3 Preliminaries

As we all know, to determine the similarity between two DNA sequences, the most common metric is to calculate the edit distances between the two sequences. In this section, we design an approximation algorithm that can calculate the edit distances between two sequences privately. It is also the building block of our proposed scheme.

3.1 Notations

For easy reference, we first list some notations which will be used frequently in this paper. As shown in Table 1.

3.2 Approximation Algorithm for Edit Distance

It is well known that the distribution of DNA sequences is subject to a special way [34]. Specifically, for any two

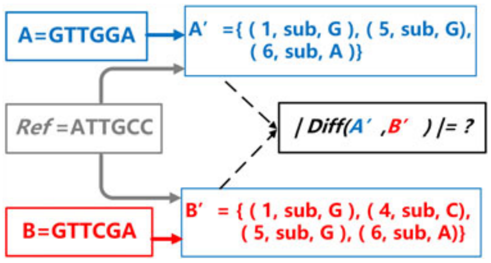

A=GTTGGA
A′ ={ (1, sub, G), (5, sub, G), (6, sub, A)}
Ref =ATTGCC
| Diff(A′ ,B′) |= ?
B=GTTCGA
B′ = { (1, sub, G), (4, sub, C), (5, sub, G), (6, sub, A)}

Fig. 2. Symmetric set difference between sets $\mathbf{A}'$ and $\mathbf{B}'$.

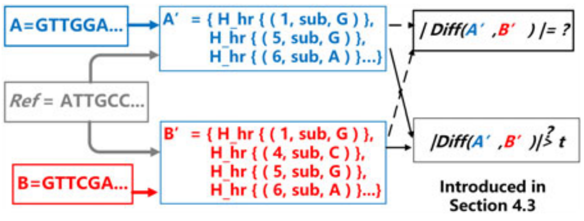

A=GTTGGA...
A′ = { H_hr { (1, sub, G) }, H_hr { (5, sub, G) }, H_hr { (6, sub, A) }...}
| Diff(A′ ,B′) |= ?
Ref = ATTGCC...
B′ = { H_hr { (1, sub, G) }, H_hr { (4, sub, C) }, H_hr { (5, sub, G) }, H_hr { (6, sub, A) }...}
B=GTTCGA...
|Diff(A′ ,B′)| ≶? t
Introduced in Section 4.3

Fig. 3. Compute symmetric set difference privately.

irrelevant people, (1) more than 99.5 percent of their DNA sequences are identical. (2) much ($\geq 95\%$) of their edits take place in the non-contiguous positions. (3) most (about 80–90 percent) edit between two DNA sequences are substitutions. Based on this, we introduce an approximation algorithm (called *Approx*) to calculate the edit distances between two sequences approximately. In general, *Approx* consists of two algorithms, i.e., *Approx_Init* and *Approx_Ed*.

Approx_Init $(\mathbf{Ref}, \mathbf{A}) \rightarrow (\mathbf{A}')$. Assume that every party agrees on a public reference genome (called **Ref**$= R = (r_1 r_2 \cdots r_T)$). Then, every raw DNA sequence will be converted to a set of editing operations (i.e., the records of minimum edits to convert the sequence **Ref** into this sequence). For example, assume $\mathbf{Ref} = ATTGCC$, given a DNA sequence $\mathbf{A} = GTTGGA$, the records of minimum edits from **Ref** to **A** can be denoted as $\mathbf{A}' = \{(1, sub, G), (5, sub, G), (6, sub, A)\}$, where sub denotes the substitution operation. Based on the distribution of DNA sequences, it is clear that this conversion is fast since the DNA sequences of any two people are very similar. In this paper, we exploit the well-known Wagner-Fischer Algorithm [35] to fulfil above conversion (Please see the Appendix for more technical details, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TCC.2020.2968893).

Approx_Ed $(\mathbf{A}', \mathbf{B}') \rightarrow (Diff(\mathbf{A}', \mathbf{B}'))$. Given two editing operations set $\mathbf{A}'$ and $\mathbf{B}'$, the algorithm *Approx_Ed*$(\mathbf{A}', \mathbf{B}')$ outputs the symmetric set difference between sets $\mathbf{A}'$ and $\mathbf{B}'$, i.e., $Diff(\mathbf{A}', \mathbf{B}')$. As shown in Fig. 2, there are three sequences $\mathbf{Ref} = ATTGCC$, $\mathbf{A} = GTTGGA$, and $\mathbf{B} = GTTCGA$. By involving the Wagner-Fischer algorithm [35], **A** and **B** will be converted into the set of editing operations denoted as $\mathbf{A}' = \{(1, sub, G), (5, sub, G), (6, sub, A)\}$, $\mathbf{B}' = \{(1, sub, G), (4, sub, C), (5, sub, G), (6, sub, A)\}$, respectively. Then, taking the symbol $Diff(\mathbf{A}', \mathbf{B}') = (\mathbf{A}' - \mathbf{B}') \cup (\mathbf{B}' - \mathbf{A}')$ to denote the symmetric set difference between sets $\mathbf{A}'$ and $\mathbf{B}'$, we have $|Diff(\mathbf{A}', \mathbf{B}')| = |\mathbf{A}'| + |\mathbf{B}'| - 2|\mathbf{B}' \cap \mathbf{A}'| = 1$. It is exactly the same as the edit distances between **A** and **B**. The main reason for implementing this approximation is that genes are subject to the above distribution, so that each DNA sequences can be easily converted into the set of editing operations. However, this approximation algorithm also brings about small errors. For example, assume $\mathbf{Ref} = ATTGCCCGA$, $\mathbf{A} = GTTGGATAA$, and $\mathbf{B} = GTTCGATGA$. Then, **A** and **B** will be transformed as $\mathbf{A}' = \{(1, sub, G), (4, ins, C), (5, del, 1), (6, sub, A), (7, sub, T)\}$, $\mathbf{B}' = \{(1, sub, G), (5, sub, G), (6, sub, A), (7, sub, T), (8, sub, A)\}$ respectively. We have $|Diff(\mathbf{A}', \mathbf{B}')| = 4$, which is different from the actual edit distances (the actual value is 2). To understand the accuracy of the above approximation algorithm more intuitively, we calculate the edit distance between two gene sequences randomly selected from the 1,000 Genomes project [34], where we use the standard dynamic programming algorithm [35] to complete 9,000 tests of sequence matching. As shown in Table 2, *Approx* shows that at least 95.46 percent test sequences exhibit an error rate of less than 5 percent. Besides, since gene edits between long sequences tend to occur more at non-adjacent locations, *Approx* shows superior accuracy for longer DNA sequences. In the Section 4.1, we will give a formal accuracy analysis to discuss how to bound the error rate of this estimation.

TABLE 2
Accuracy of Approximation Algorithm

Proportions	Le_sequences [1]	Nu_tests [2]	Relative error
80.65%	80,000	1,000	0.25%
99.46%	80,000	1,000	0.5%
100%	80,000	1,000	1%
78.65%	60,000	1,000	0.25%
96.46%	60,000	1,000	0.5%
100%	60,000	1,000	1%
77.65%	40,000	1,000	0.25%
95.46%	40,000	1,000	0.5%
100%	40,000	1,000	1%

[1]*The length of DNA sequences.*
[2]*The number of tests.*

3.3 Computing Symmetric Set Difference Privately

In this section, we design a secure algorithm to calculate the symmetry difference between two sequences privately. As shown in Fig. 3, unlike Section 3.2, we first block sequences **A**, **B** and **Ref**, each of which represents a DNA fragment that controls a particular gene. Specifically, assume **A**, **B**, and **Ref** are blocked as $\tilde{\mathbf{A}} = (A_1, A_2, A_i \ldots, A_s)$, $\tilde{\mathbf{B}} = (B_1, B_2, B_i \ldots, B_s)$ and $\tilde{\mathbf{R}} = (R_1, R_2, R_i \ldots, R_s)$ respectively, where A_i, B_i and R_i represents corresponding subsequence. For example, given a DNA sequence $\mathbf{A} = GTTGGATAA$, it can be blocked into $\tilde{\mathbf{A}} = (GTT, GGA, TAA) = (A_1, A_2, A_3)$. Then, similar to Section 3.2, each A_i and B_i will be converted into a subset of editing operations denoted as $A_i^{'}$ and $B_i^{'}$ separately. Thus, **A** and **B** can ultimately be expressed as $\mathbf{A}' = (A_1^{'}, A_2^{'}, A_i^{'} \cdots A_s^{'})$ and $\mathbf{B}' = (B_1^{'}, B_2^{'}, B_i^{'} \cdots B_s^{'})$.

Next, we take $\mathbf{A}'$ as an example to introduce the details of our secure approximation algorithm. Specifically, for each element μ in $\mathbf{A}'$, it will be compressed into $\{-1, 1\}$ by a binary hash function $H_{hr} : \mathbf{U} \longrightarrow \{-1, 1\}$, where hr is the secret key selected by the data owner, and **U** denotes the universe of all set elements. Then, due to the randomness of hash function, for any element μ, μ_1 and μ_2 belong to set $\mathbf{A}'$, we have $E[H_{hr}(\mu)] = 0$, $E[H_{hr}^2(\mu)] = 1$, and $E_{\mu_1 \neq \mu_2}$

$[H_{hr}(\mu_1)H_{hr}(\mu_2)] = E[H_{hr}(\mu_1)]\cdot E[H_{hr}(\mu_2)] = 0$. Based on these, we use $d_{\mathbf{A}'}$ to denote the sum of all $H_{hr}(\mu), \mu \in \mathbf{A}'$, and the following equation is established.

$$
\begin{aligned}
E[d^2_{\mathbf{A}'}] &= E\left[\left(\sum_{i=1}^{s}\sum_{\mu\in A'_i} H_{hr}(\mu)\right)^2\right]\\
&= E\left[2\sum_{i=1}^{s}\sum_{\mu\in A'_i \& \mu'\in A'_i} H_{hr}(\mu)H_{hr}(\mu')\right] + E\left[\sum_{i=1}^{s}\sum_{\mu\in A'_i} H^2_{hr}(\mu)\right]\\
&\quad + E\left[\sum_{\mu\in A'_k \& \mu'\in A'_j \& k\neq j} H_{hr}(\mu)H_{hr}(\mu')\right] (Square\ decomposition)\\
&= \sum_{i=1}^{s} E\left[\sum_{\mu\in A'_i} H^2_{hr}(\mu)\right] = \sum_{i=1}^{s} |A'_i|,
\end{aligned}
\tag{1}
$$

where $|A'_i|$ denotes the number of the elements in the subset A'_i. Similarly, we have

$$
E[d^2_{\mathbf{B}'}] = \sum_{i=1}^{s} |B'_i|. \tag{2}
$$

Known

$$
\begin{aligned}
d_{\mathbf{A}'} - d_{\mathbf{B}'} &= \sum_{i=1}^{s}\sum_{\mu\in A'_i} H_{hr}(\mu) - \sum_{i=1}^{s}\sum_{\mu\in B'_i} H_{hr}(\mu)\\
&= \sum_{i=1}^{s}\left(\sum_{\mu\in A'_i - B'_i} H_{hr}(\mu) - \sum_{\mu\in B'_i - A'_i} H_{hr}(\mu)\right).
\end{aligned}
\tag{3}
$$

We have

$$
\begin{aligned}
E[(d_{\mathbf{A}'} - d_{\mathbf{B}'})^2] &= E\left[\sum_{i=1}^{s}\left(\sum_{\mu\in A'_i - B'_i} H^2_{hr}(\mu)\right.\right.\\
&\quad \left.\left. + \sum_{\mu\in B'_i - A'_i} H^2_{hr}(\mu) - 2\sum_{\mu_1\in A'_i - B'_i} H_{hr}(\mu_1)\sum_{\mu_2\in B'_i - A'_i} H_{hr}(\mu_2)\right)\right]\\
&= E\left[\sum_{i=1}^{s}\sum_{\mu\in A'_i - B'_i} H^2_{hr}(\mu)\right] + E\left[\sum_{i=1}^{s}\sum_{\mu\in B'_i - A'_i} H^2_{hr}(\mu)\right] - 2\cdot 0\\
&= \sum_{i=1}^{s}\left(|A'_i - B'_i| + |B'_i - A'_i| - 2\cdot 0\right)\\
&= \sum_{i=1}^{s} |Diff(A'_i, B'_i)|.
\end{aligned}
\tag{4}
$$

Hence, according to the Eqn. (4), for any two DNA sequences, we can approximate their edit distances by privately calculating the symmetry difference of their set of editing operations.

4 Proposed Schemes

In this section, we describe the technical detail of EFSS. In order to illustrate our scheme in a gradual way, we first give a basic model called EFSS_I, which can perform secure similarity queries on outsourced DNA data. Then, we propose EFSS_II to implement the functions of hybrid Boolean queries and data access control. Finally, by using the K-means clustering algorithm, we update our EFSS_II to further improve the efficiency.

4.1 EFSS_I

The basic model is shown in Fig. 4. It consists of four parts: **Initialization**, **Index generation**, **Trapdoor generation** and **Query**. The data owner first needs to pre-process all DNA sequences to be outsourced. This operation converts each DNA sequence into its corresponding set of editing operations. Then, the data owner will generate an encrypted index for each DNA sequence, and send the index along with the encrypted raw data to the cloud server. During the query phase, the search user also needs to raise secure query requests (i.e., trapdoors) to the cloud server. Finally, the cloud server matches all the indexes associated with the trapdoor in the ciphertext environment, and returns the matching results to the user.

Correctness. We take $\widehat{D_i}, i = \{1, 2\}$ as an example to analyze the correctness of our EFSS_I. Specifically, based on the Chebyshev's Large Number Theorem (CLN Theorem), we have

$$
\begin{aligned}
\widehat{D_1} - \widehat{D_2} &= \frac{\sum_{j=1}^{k} D_1^j}{k} - \frac{\sum_{j=1}^{k} D_2^j}{k}\\
&= \frac{1}{k}\sum_{j=1}^{k}\left(I^j_{\tilde{\mathbf{A}'_1}}\cdot T^j_{\tilde{\mathbf{Q}'}} - I^j_{\tilde{\mathbf{A}'_1}}\cdot T^j_{\tilde{\mathbf{Q}'}}\right)\\
&= \frac{1}{k}\sum_{j=1}^{k}\left(I^j_{\mathbf{A}'_1} - I^j_{\mathbf{A}'_2}, -\frac{1}{2}\left(||I^j_{\mathbf{A}'_1}||_2\right)^2 + \frac{1}{2}\left(||I^j_{\mathbf{A}'_2}||_2\right)^2\right)\cdot T^j_{\tilde{\mathbf{Q}'}}\\
&= \frac{1}{2k}\sum_{j=1}^{k}\left[\left(I^j_{\mathbf{A}'_2} - T^j_{\mathbf{Q}'}\right)^2 - \left(I^j_{\mathbf{A}'_1} - T^j_{\mathbf{Q}'}\right)^2\right]\\
&= \frac{1}{2}E\left[\left(d_{\mathbf{A}'_2} - d_{\mathbf{Q}'}\right)^2 - \left(d_{\mathbf{A}'_1} - d_{\mathbf{Q}'}\right)^2\right]\left(CLN\ Theorem\right)\\
&= \frac{1}{2}\sum_{i=1}^{i=s}\left[Diff(A'_{2i}, Q'_i) - Diff(A'_{1i}, Q'_i)\right](Based\ on\ Eqn.(4)).
\end{aligned}
$$

Therefore, if $\widehat{D_1} > \widehat{D_2} \Rightarrow \sum_{i=1}^{i=s}|Diff(A'_{2i}, Q'_i)| > \sum_{i=1}^{i=s}|Diff(A'_{1i}, Q'_i)|$, it shows that the DNA sequence $\mathbf{A_1}$ is closer to $\mathbf{Q}$.

Accuracy. As shown in Fig. 4, we use the CLN Theorem to estimate the expectation of $(d_{\mathbf{A}'_\mathbf{i}} - d_{\mathbf{Q}'})^2, i \in \{1, 2\}$, which will incur some error in the accuracy of the final match. We present a formal analysis to show how to bound the error rate of this estimation.

For ease of presentation, we take $\mathbf{A}'_\mathbf{i}$ and $\mathbf{Q}'$ as an example. Specifically, for $1 \le m \le s$, we define $d_{im} = E[(d_{A'_{im}} - d_{Q'_m})^2] = |Diff(A'_{im}, Q'_m)|$. Then, we have

$$
\begin{aligned}
E(X_i) &= E\left[(d_{\mathbf{A}'_\mathbf{i}} - d_{\mathbf{Q}'})^2\right] = E\left[\sum_{m=1}^{m=s}\left(d_{A'_{im}} - d_{Q'_m}\right)^2\right]\\
&= \sum_{m=1}^{m=s} d_{im}\\
Var(X_i) &= Var\left[\left(d_{\mathbf{A}'_\mathbf{i}} - d_{\mathbf{Q}'}\right)^2\right] = \sum_{m=1}^{m=s} Var\left(d_{A'_{im}} - d'_{Q_m}\right)^2.
\end{aligned}
$$

Implementation process of EFSS_I

- **Initialization** :
  - Data owner: pre-process all DNA data before being outsourced. That is, convert each DNA sequence $\mathbf{A_i}$ into its corresponding set of editing operations $\mathbf{A_i'}$, where $\mathbf{A_i'}$ is represented as $\mathbf{A_i'} = (A'_{i1}, A'_{i2}, \cdots A'_{im}, \cdots A'_{is}), (i = 1, 2 \cdots N, m = 1, 2 \cdots s)$. For the i-th DNA sequence, A'_{im} denotes the set of editing operations of the m-th subsequence.
- **Index generation** :

  For each set $\mathbf{A_i'}$, the data owner repeats the following operations k times:
  - At j-th time, randomly select a hash function $H_{r_j} : \mathbf{U} \longrightarrow \{-1, 1\}$, where r_j is the secret key, and $\mathbf{U}$ denotes the universe of all set elements.
  - Convert each element $\mu \in \mathbf{A_i'}$ into a single integer $H_{r_j}(\mu) \in \{-1, 1\}$.
  - For $1 \leq m \leq s$, calculate $d_{A'^j_{im}} = \sum_{\mu \in A'_{im}} H_{r_j}(\mu)$.
  - Send the encrypted index of $\mathbf{A_i}$ as $I^j_{\mathbf{A_i'}} = (d_{A'^j_{i1}}, d_{A'^j_{i2}}, d_{A'^j_{im}}, \cdots, d_{A'^j_{is}})$ to the cloud server, and all the secret keys (used to generate the secure index) to the authorized search user.
- **Trapdoor generation** :

  Search user:
  - Preprocessing: For a query request, i.e., a DNA sequence $\mathbf{Q}$ that needs to be matched, generate the set of its editing operations $\mathbf{Q'}$, where $\mathbf{Q'}$ is represented as $\mathbf{Q'} = (Q'_1, Q'_2, \cdots, Q'_m, \cdots, Q'_s)$, $(m = 1, 2 \cdots s)$.

  The search user repeats the following operations k times:
  - At j-th time, convert each element $\mu \in \mathbf{Q'}$ into a single integer $H_{r_j}(\mu) \in \{-1, 1\}$.
  - For $1 \leq m \leq s$, calculate $d_{Q'^j_m} = \sum_{\mu \in Q'_m} H_{r_j}(\mu)$.
  - Send the trapdoor (i.e., the encrypted search request) as $T^j_{\mathbf{Q'}} = (d_{Q'^j_1}, d_{Q'^j_2}, d_{Q'^j_m}, \cdots, d_{Q'^j_s})$ to the cloud server.
- **Query** :

  The cloud server:
  - Receiving all the messages of $I^j_{\mathbf{A_i'}}$ and $T^j_{\mathbf{Q'}}$, $(i = 1, 2, \cdots N, j = 1, 2, \cdots k)$. Otherwise, abort.
  - For $1 \leq j \leq k$, modify $T^j_{\mathbf{Q'}}$ to $T^j_{\tilde{\mathbf{Q}}'} = (d_{Q'^j_1}, d_{Q'^j_2}, d_{Q'^j_m}, \cdots, d_{Q'^j_s}, 1)$.
  - For $1 \leq i \leq N$, modify $I^j_{\mathbf{A_i'}}$ to $I^j_{\tilde{\mathbf{A}}_i'} = (d_{A'^j_{i1}}, d_{A'^j_{i2}}, d_{A'^j_{im}}, \cdots, d_{A'^j_{is}}, -\frac{1}{2}(||I^j_{\mathbf{A_i'}}||_2)^2)$
  - Calculate $D^j_i = I^j_{\tilde{\mathbf{A}}_i'} \cdot T^j_{\tilde{\mathbf{Q}}'}$ and $\widehat{D_i} = \frac{\sum_{j=1}^k D^j_i}{k}$. Then, sort the $\widehat{D_i}$ from large to small.
  - Return the top $\mathcal{K}$ results to the authorized search user.

Fig. 4. Detailed description of the EFSS_I.

Further, we have

$$
\begin{aligned}
&E\Big[\Big(d'_{A_{im}} - d_{Q'_m}\Big)^4\Big] \\
&= E\left[\left(\sum_{\mu \in A'_{im} - Q'_m} H_r(\mu) - \sum_{\mu \in Q'_m - A'_{im}} H_r(\mu)\right)^4\right],
\end{aligned}
$$

where r denotes a secret key. Let Δ_m denote those elements in $Diff(A'_{im}, Q'_m)$, and $H'_r(\mu)$ denote the hash value for each $\mu \in \Delta_m$. Then, we have $E[(H'_r(\mu))^2] = E[(H'_r(\mu))^4] = 1$. By utilizing the Multinomial theorem, the above formula can be rewritten as below.

$$
\begin{aligned}
&E\left[\left(\sum_{\mu \in A'_{im} - Q'_m} H_r(\mu) - \sum_{\mu \in Q'_m - A'_{im}} H_r(\mu)\right)^4\right] \\
&= E\left[\left(\sum_{\mu \in \Delta_m} H'_r(\mu)\right)^4\right] \\
&= E\left[\sum_{\mu \in \Delta_m} (H'_r(\mu))^4 + 6 \sum_{\mu \in \Delta_m \& \mu_1 \neq \mu_2} H'_r(\mu_1)^2 H'_r(\mu_2)^2\right] \\
&= E\left[\sum_{\mu \in \Delta_m} (H'_r(\mu))^4 + 3\left(\sum_{\mu \in \Delta_m} (H'_r(\mu))^2\right)^2 - \sum_{\mu \in \Delta_m} (H'_r(\mu))^4\right] \\
&= d_{im} + 3(d^2_{im} - d_{im}) \\
&= 3d^2_{im} - 2d_{im}.
\end{aligned}
$$

Therefore, we can get

$$
\begin{aligned}
Var(X_i) &= Var\left[\sum_{m=1}^{m=s} (d_{A'_{im}} - d_{Q'_m})^2\right] = \sum_{m=1}^{m=s} Var(d_{A'_{im}} - d_{Q'_m})^2 \\
&= \sum_{m=1}^{j=s} E\Big[(d_{A'_{im}} - d_{Q'_m})^4\Big] - \sum_{m=1}^{j=s} E\Big[(d_{A'_{im}} - d_{Q'_m})^2\Big]^2 \\
&= \sum_{m=1}^{m=s} (3d^2_{im} - 2d_{im} - d^2_{im}) \\
&= \sum_{m=1}^{m=s} (2d^2_{im} - 2d_{im}).
\end{aligned}
$$

Thus, we define $E[\widehat{X_i}] = E[\frac{1}{k}\sum_{j=1}^{j=k}(d_{\mathbf{A_i'}^j} - d_{\mathbf{Q'}^j})^2] = \zeta$, and $Var[\widehat{X_i}] = Var[\frac{1}{k}\sum_{j=1}^{j=k}(d_{\mathbf{A_i'}^j} - d_{\mathbf{Q'}^j})^2] \leq \frac{\sum_{m=1}^{m=s}(2d^2_{im})}{k} = v$. Using the Chebyshev inequality, for any positive number ϵ, let $k = \frac{6v}{(\epsilon)^2}$, we have

$$Pr\{|\widehat{X_i} - \zeta| \geq \epsilon\} \leq \tfrac{1}{6}.$$

Therefore, we can control the error rate of our approximate algorithm by adjusting the cycle times k adaptively, which means that the more cycles, the higher accuracy our approximate algorithm will acquire.

4.2 EFSS_II

The main contribution of EFSS_I is to fulfill the DNA similarity search among encrypted DNA sequences. However, in reality, different users may have different data access authorities to each DNA sequence. For example, the attending

Implementation process of EFSS_II

- **Initialization** :
  – Data owner: pre-process all DNA data before being outsourced. That is, convert each DNA sequence $\mathbf{A_i}$ into its corresponding set of editing operations $\mathbf{A_i'}$, where $\mathbf{A_i'}$ is represented as $\mathbf{A_i'} = (A'_{i1}, A'_{i2}, \cdots A'_{im}, \cdots A'_{is}), (i = 1, 2 \cdots N, m = 1, 2 \cdots s)$. For the i-th DNA sequence, A'_{im} denotes the set of editing operations of the m-th subsequence.
  – Modify $\mathbf{A_i'}$ to $\widetilde{\mathbf{A_i}} = \{\mathbf{A_i'}, \beta_{i0}, \cdots \beta_{i\omega}, \gamma_{i0}, \cdots \gamma_{i\psi}\}$, where β_{il} and $\gamma_{i\rho}$, $l = \{0, 1, \cdots \omega\}$, $\rho = \{0, 1, \cdots \psi\}$ represent the corresponding coefficient of $f_i(x)$ and the dummy noise added by data owner, respectively.
- **Index generation** :
  For each set $\widetilde{\mathbf{A_i}}$, the data owner repeats the following operations k times:
  – At j-th time, randomly select a hash function $H_{r_j} : \mathbf{U} \longrightarrow \{-1, 1\}$, where r_j is the secret key, and $\mathbf{U}$ denotes the universe of all set elements.
  – Convert each element $\mu \in \mathbf{A_i'}$ into a single integer $H_{r_j}(\mu) \in \{-1, 1\}$.
  – For $1 \leq m \leq s$, calculate $d_{A'^j_{im}} = \sum_{\mu \in A'_{im}} H_{r_j}(\mu)$.
  – Calculate $\widetilde{\mathbf{A_i}}^j = (d_{A'^j_{i1}}, -\frac{1}{2}(d_{A'^j_{i1}})^2, \cdots, d_{A'^j_{is}}, -\frac{1}{2}(d_{A'^j_{is}})^2, \beta_{i0}, \cdots \beta_{i\omega}, \gamma_{i0}, \cdots \gamma_{i\psi})$.
  – For $1 \leq m \leq (2s + \omega + \psi + 2)$, compute $\widetilde{\mathbf{A_i}^j}'(\pi_{S_{kj}}(m)) = \widetilde{\mathbf{A_i}}^j(m)$.
  – Randomly select a $(2s + \omega + \psi + 2)$ dimensional binary vector S_j, and two $(2s + \omega + \psi + 2) \times (2s + \omega + \psi + 2)$ dimensional invertible matrices M_{1j}, M_{2j}.
  – For $1 \leq m \leq (2s + \omega + \psi + 2)$, **if** $S_j(m) = 0$, **then** $D^{j'}_{\mathbf{A_i}}(m) = D^{j''}_{\mathbf{A_i}}(m) = \widetilde{\mathbf{A_i}^j}'(m)$. **Otherwise** $D^{j'}_{\mathbf{A_i}}(m) + D^{j''}_{\mathbf{A_i}}(m) = \widetilde{\mathbf{A_i}^j}'(m)$, where both $D^{j'}_{\mathbf{A_i}}(m)$ and $D^{j''}_{\mathbf{A_i}}(m)$ are randomly chosen by the data owner.
  – Send the encrypted index as $I^j_{\mathbf{A_i}} = \{M_{1j} D^{j'}_{\mathbf{A_i}}, M_{2j} D^{j''}_{\mathbf{A_i}}\}$ to the cloud server, and all the secret keys (used to generate the secure index) to the authorized search user.
- **Trapdoor generation** :
  Search user:
  – Preprocessing: For a query request, i.e., a DNA sequence $\mathbf{Q}$ that needs to be matched, generate the set of its editing operations $\mathbf{Q'}$. There $\mathbf{Q'}$ is represented as $\mathbf{Q'} = (Q'_1, Q'_2, \cdots, Q'_m, \cdots, Q'_s)$, $(m = 1, 2 \cdots s)$.
  – Modify $\mathbf{Q'}$ to $\widetilde{\mathbf{Q}} = \{\mathbf{Q'}, \eta^0, \cdots \eta^\omega, \gamma'_0, \cdots \gamma'_\psi\}$, where η represents the role of current search user, and γ'_ρ, $\rho = \{0, 1, \cdots \psi\}$ denotes the corresponding dummy noise added by search user.
  The search user repeats the following operations k times:
  – At j-th time, for $1 \leq m \leq s$, **if** $Q'_m \subseteq \mathbf{Q^N}$, **then** $d_{Q'^j_m} = (0, 0)$. **Otherwise**, calculate $d_{Q'^j_m} = (\sum_{\mu \in Q'_m} H_{r_j}(\mu), 1)$.
  – Calculate $\widetilde{\mathbf{Q}}^j = (d_{Q'^j_1}, \cdots d_{Q'^j_s}, \eta^0, \cdots \eta^\omega, \gamma'_{i0}, \cdots \gamma'_{i\psi})$.
  – For $1 \leq m \leq (2s + \omega + \psi + 2)$, compute $\widetilde{\mathbf{Q}^j}'(\pi_{S_{kj}}(m)) = \widetilde{\mathbf{Q}}^j(m)$.
  – For $1 \leq m \leq (2s + \omega + \psi + 2)$, **if** $S_j(m) = 1$, **then** $D^{j'}_{\mathbf{Q}}(m) = D^{j''}_{\mathbf{Q}}(m) = \widetilde{\mathbf{Q}^j}'(m)$. **Otherwise** $D^{j'}_{\mathbf{Q}}(m) + D^{j''}_{\mathbf{Q}}(m) = \widetilde{\mathbf{Q}^j}'(m)$, where both $D^{j'}_{\mathbf{Q}}(m)$ and $D^{j''}_{\mathbf{Q}}(m)$ are randomly chosen by the search user.
  – Send the trapdoor (i.e., the encrypted search request) $T^j_{\mathbf{Q}} = \{M^{-1}_{1j} D^{j'}_{\mathbf{Q}}, M^{-1}_{2j} D^{j''}_{\mathbf{Q}}\}$ to the cloud server.
- **Query** :
  The cloud server:
  – Receiving all the messages of $I^j_{\mathbf{A_i}}$ and $T^j_{\mathbf{Q}}$, $(i = 1, 2, \cdots N, j = 1, 2, \cdots k)$. Otherwise, abort.
  – For each $I^j_{\mathbf{A_i}}$ and and $T^j_{\mathbf{Q}}$, calculate $D^j_i = I^j_{\mathbf{A_i}} \cdot T^j_{\mathbf{Q}}$, where $1 \leq j \leq k$.
  – Calculate $\widehat{D_i} = \frac{\sum_{j=1}^k D^j_i}{k}$ and sort the $\widehat{D_i}$ from large to small.
  – **If** $|\widehat{D_i}| - D < 0$, save $\widehat{D_i}$. **Otherwise** abort it.
  – Return the top $\mathcal{K}$ results to the authorized search user.

Fig. 5. Detailed description of the EFSS_II.

doctors of breast cancer probably have not any access rights except breast cancer patient's DNA sequences. Besides, sometimes we do not need to query over entire DNA collections, where specific DNA sequences controlling particular genetic genes (such as genes controlling physical characteristics, mammary cancer, diabetes, etc.) should be the focus of our attention. What's more, EFSS_I cannot guarantee the unlinkability of trapdoors since the query frequencies are leaked to the cloud server. As a result, the cloud server can easily observe the relationship of trapdoors and grab the search user's query histories, which may threaten the confidentiality of users' DNA data. Based on the situation above, in this section, we propose EFSS_II, an enhanced model to achieve Boolean query and data access control over encrypted cloud data with a higher security level.

EFSS_II is shown in Fig. 5. Concretely, we leverage following methods to realize the proposed functions and security requirements.

4.2.1 Variant of Secure kNN Computation

As shown in Fig. 4, in the jth iteration of EFSS_I, the encrypted index of $\mathbf{A_i}$ is generated as $I^j_{\mathbf{A_i'}} = (d_{A'^j_{i1}}, d_{A'^j_{i2}}, d_{A'^j_{im}}, \ldots, d_{A'^j_{is}})$. It may give a chance to the cloud server deducing the relationship between reference genome **Ref** and $\mathbf{A_i}$. Specifically, based on the public **Ref**, the cloud server can easily get the similarity between **Ref** and $\mathbf{A_i}$ by computing $E[d^2_{\mathbf{A_i'}}] = \sum_{j=1}^s |A'_{ij}|$. For example, if $E[d^2_{\mathbf{A_i'}}] = 0$, the cloud server can almost assert that $\mathbf{Ref} = \mathbf{A_i}$, thus it can further obtain the content of $\mathbf{A_i}$.

To address this problem, in our EFSS_II, we adopt a variant of secure kNN computation [36] to solve above problem. Concretely, in our EFSS_II, we first define a pseudorandom permutation function π as follows.

$$\pi : \{0,1\}^* \times \{0,1\}^{2s+\omega+\psi+2} \rightarrow \{0,1\}^{2s+\omega+\psi+2}.$$

Then, as shown in **Index generation** and **Trapdoor generation**, $\widetilde{\mathbf{A_i}}^j$ and $\widetilde{\mathbf{Q}}^j$ will be permutated as $\widetilde{\mathbf{A_i}}^{j\prime}(\pi_{S_{kj}}(m)) = \widetilde{\mathbf{A_i}}^j(m)$ and $\widetilde{\mathbf{Q}}^{j\prime}(\pi_{S_{kj}}(m)) = \widetilde{\mathbf{Q}}^j(m)$, respectively, where S_{kj} is the secret key in the jth iteration. Next, a $(2s+\omega+\psi+2)$ dimensional binary vector S_j and two $(2s+\omega+\psi+2)\times(2s+\omega+\psi+2)$ invertible matrices M_{1j}, M_{2j} will be adopted to further randomize $\widetilde{\mathbf{A_i}}^{j\prime}$ and $\widetilde{\mathbf{Q}}^{j\prime}$. In this way, the cloud server cannot obtain any additional information among the encrypted indexes/trapdoors because of the random connection of ciphertexts.

4.2.2 Achieving Fine-Grain Query

In our EFSS_II, each DNA sequence $\mathbf{A_i}$ is divided into fixed sub-blocks, each of which represents a DNA fragment that controls a specific gene. It will facilitate the Boolean query over encrypted cloud data. More concretely, we assume that set $\mathbf{Q^N}$ (which is a subset of all gene types) contains those types of genes that are not considered in current query. As shown in **Trapdoor generation**, at jth iteration, if $Q'_m \subseteq \mathbf{Q^N}$, $d_{Q'^j_m}$ will be set as (0, 0). Otherwise, $d_{Q'^j_m} = (\sum_{\mu\in Q'_m} H_{r_j}(\mu), 1)$. By doing so, when the cloud server executes the operation (i.e, $I^j_{\mathbf{A_i}} \cdot T^j_{\mathbf{Q}}$) over encrypted DNA sequences, every DNA fragment belonged to $\mathbf{Q^N}$ will be filtered out, since we have $d_{Q'^j_m} \times (d_{A'^j_{i(2m-1)}}, -\frac{1}{2}(d_{A'^j_{i(2m)}})^2) = 0$ (shown in the next subsection). Hence, the Boolean query such as mixed "AND" and "NO" operations on genes can be achieved.

4.2.3 Data Access Control

As discussed in Section 4.2, different users may have different access rights to DNA sequences. Without loss of generality, assume there are a total of $\mathcal{R}$ roles in our EFSS_II, where $\mathcal{R} = (\delta_1, \delta_2, \ldots, \delta_\omega)$. Then, for each DNA sequence $\mathbf{A_i}$, we define a role polynomial as below.

$$f_i(x) = \prod_{select\ \delta_{ij}\in\mathcal{R}} (x-\delta_{ij}) = \sum_{j=0}^{j=\omega} \beta_{ij}x^j, \tag{5}$$

where $f_i(x)$ is the role polynomial, and δ_{ij}, $j=(1,2,\ldots\omega)$ denotes a role of $\mathbf{A_i}$. Here we use ϕ_i to represent the degree of $f_i(x)$, which satisfies $\beta_{ij}=0$ when $(\phi_i < j \le \omega)$.

Further, we assume that set $\mathcal{R} = (\delta_1, \delta_2, \ldots, \delta_\omega)$ is subject to the following conditions.

$$\begin{aligned} &\delta_1 > D > \max\left\{\left|\sum_{m=1}^{m=2s}(\widetilde{\mathbf{A_i}}^j(m)\cdot\widetilde{\mathbf{Q}}^j(m))\right|\right\} \\ &(i=1,2,\ldots N, j=1,2,\ldots k, m=1,2,\ldots 2s) \\ &\sum_{j=1}^{j=g-1}\delta_j + 2D < \delta_g, (g=2,3,\ldots\omega). \end{aligned} \tag{6}$$

Next, we introduce how EFSS_II achieves the data access control under the encrypted environment.

As shown in Fig. 5, we have

$$\begin{aligned} D_i^j &= I^j_{\mathbf{A_i}}\cdot T^j_{\mathbf{Q}} = \{M_{1j}D^{j\prime}_{\mathbf{A_i}}, M_{2j}D^{j\prime\prime}_{\mathbf{A_i}}\}\cdot\{M^{-1}_{1j}D^{j\prime}_{\mathbf{Q}}, M^{-1}_{2j}D^{j\prime\prime}_{\mathbf{Q}}\} \\ &= D^{j\prime}_{\mathbf{A_i}}\cdot D^{j\prime}_{\mathbf{Q}} + D^{j\prime\prime}_{\mathbf{A_i}}\cdot D^{j\prime\prime}_{\mathbf{Q}} = \widetilde{\mathbf{A_i}}^{j\prime}\cdot\widetilde{\mathbf{Q}}^{j\prime} = \widetilde{\mathbf{A_i}}^j\cdot\widetilde{\mathbf{Q}}^j \\ &= \sum_{m=1}^{m=2s}(\widetilde{\mathbf{A_i}}^j(m)\cdot\widetilde{\mathbf{Q}}^j(m)) + \sum_{l=0}^{l=\omega}\beta_{il}\eta^l + \sum_{\rho=0}^{\rho=\phi}\gamma_{i\rho}\cdot\gamma'_\rho, \end{aligned}$$

where the dummy noises selected in our model obey a normal distribution with mean and variance as $(0,\theta^2)$. θ^2 is a small positive number.

Case 1. if a search user can access data $\mathbf{A_i}$, this means that the user's role $\eta\in\mathcal{R}$ is a root of $f_i(x)$. Then, we have $\sum_{l=0}^{l=\omega}\beta_{il}\eta^l = 0$. Since each dummy noise is selected from the normal distribution $(0,\theta^2)$, we have $E[\sum_{\rho=0}^{\rho=\phi}\gamma_{i\rho}\cdot\gamma'_\rho] = 0$. Therefore, following equation is established.

$$\begin{aligned} \widehat{D_i} &= \frac{\sum_{j=1}^{k} D_i^j}{k} \approx E\left[I^j_{\mathbf{A_i}}\cdot T^j_{\mathbf{Q}}\right] \\ &= E\left[\sum_{m=1}^{m=2s}(\widetilde{\mathbf{A_i}}^j(m)\cdot\widetilde{\mathbf{Q}}^j(m)) + 0 + 0\right] \\ &= E\left[\sum_{m=1}^{m=s}\left(d_{A'^j_{i(2m-1)}}, -\frac{1}{2}\left(d_{A'^j_{i(2m)}}\right)^2\right)\cdot d_{Q'^j_m}\right]. \end{aligned} \tag{7}$$

According to the definition $D > \max\left\{\left|\sum_{m=1}^{m=s}(d_{A'^j_{im}}, -\frac{1}{2}(d_{A'^j_{im}})^2)\cdot d_{Q'^j_m}\right|\right\}$, we have $|\widehat{D_i}| - D < 0$, which satisfies the restrictions in **Query** process. Moreover, $d_{Q'^j_m}\times(d_{A'^j_{i(2m-1)}}, -\frac{1}{2}(d_{A'^j_{i(2m)}})^2) = 0$ is established for each Q'_m belonged to $\mathbf{Q^N}$, hence the mixed "AND" and "NO" operations on genes is also completed.

The correctness of EFSS_II can be easily derived from EFSS_I because the results returned by the two models are consistent. At this point, our EFSS_II enables the functions of Boolean query and data access control over encrypted cloud data.

Case 2. if a search user cannot access data $\mathbf{A_i}$, this means that the user's role $\eta\in\mathcal{R}$ is not a root of $f_i(x)$. Based on the Eqn. (7), we have

$$\begin{aligned} \widehat{D_i} &= \frac{\sum_{j=1}^{k} D_i^j}{k} \approx E\left[I^j_{\mathbf{A_i}}\cdot T^j_{\mathbf{Q}}\right] \\ &= E\left[\sum_{m=1}^{m=2s}(\widetilde{\mathbf{A_i}}^j(m)\cdot\widetilde{\mathbf{Q}}^j(m)) + \sum_{l=0}^{l=\omega}\beta_{il}\eta^l + 0\right]. \end{aligned}$$

Reviewing the constraints in the Eqn. (6), we know that $\delta_1 > D > \max\left\{\left|\sum_{m=1}^{m=2s}(\widetilde{\mathbf{A_i}}^j(m)\cdot\widetilde{\mathbf{Q}}^j(m))\right|\right\}$, and $\sum_{j=1}^{j=g-1}\delta_j + 2D < \delta_g$ for any $g=(2,3,\ldots\omega)$. Then, we have $|\sum_{l=0}^{l=\omega}\beta_{il}\eta^l| \ge \delta_2 - \delta_1 > 2D$. Therefore, the following equation is established.

$$\begin{aligned} |\widehat{D_i}| - D &\approx \left|E\left[I^j_{\mathbf{A_i}}\cdot T^j_{\mathbf{Q}}\right]\right| - D \\ &= \left|\left[\sum_{m=1}^{m=2s}(\widetilde{\mathbf{A_i}}^j(m)\cdot\widetilde{\mathbf{Q}}^j(m)) + \sum_{l=0}^{l=\omega}\beta_{il}\eta^l\right]\right| - D \\ &> \left|\sum_{l=0}^{l=\omega}\beta_{il}\eta^l\right| - \left|\sum_{m=1}^{m=2s}(\widetilde{\mathbf{A_i}}^j(m)\cdot\widetilde{\mathbf{Q}}^j(m))\right| - D \\ &> (|\delta_2-\delta_1|) - D - D \\ &> 0. \end{aligned}$$

Therefore, $\widehat{D_i}$ will be aborted since this search user does not meet our pre-defined data access mechanism.

Summary of the *Case 1* and *Case 2*, we can assure that EFSS_II can achieve all the functions and the security requirements we claim.

4.3 Enhanced Scheme

As the description on EFSS_I and EFSS_II, each DNA sequence $\mathbf{A_i}$ is compressed into its corresponding set of editing operations $\mathbf{A_i'}$. This conversion can efficiently reduce the number of elements to be matched in the querying process. However, each encrypted index still needs to be performed inner product operations (i.e., $I^j_{\mathbf{A_i}} \cdot T^j_{\mathbf{Q}}$) with the trapdoor, which leads to high computational overheads. To address this problem, we update EFSS_II to EFSS_II (ES) by utilizing K-means clustering algorithm to further improve the efficiency. Specifically, in **Initialization** process, the data owner first uses the K-means clustering algorithm to divide all DNA sequences into K classes and generates the corresponding encrypted indexes for all DNA sequences and K cluster centers. Next, in **Query** process, given a threshold **t** (selected by the search user), the cloud server first computes $I^j_{\kappa_\mathbf{i}} \cdot T^j_{\mathbf{Q}}$ to find those cluster centers satisfying $Diff|(\kappa_i{}', \mathbf{Q}')| \leq \mathbf{t}$, where $\kappa_i, i = (1, 2, \ldots K)$ is a cluster center, and $\kappa_i{}'$ is the set of its editing operations. Then, only those encrypted indexes belonged to selected centers will be further computed in the following querying. In this way, a large number of unrelated DNA sequences will be filtered out, which significantly reduces the ciphertext matching time. Since the K-means clustering algorithm can be easily applied to our EFS-S_II, for the sake of simplicity, there we only give a K-means clustering algorithm as an example (Please see Algorithm 1 in Appendix, available in the online supplemental material) to describe how to find the K classes over encrypted DNA data.

5 Security Definitions and Analysis

In this section, we will analyze the security of our EFSS_II. As shown in Section 2.2, the security requirements of our schemes are confidentiality of DNA sequence data, privacy protection of index and trapdoor, and the unlinkability of trapdoors. Concretely, the raw DNA sequences have been encrypted by encryption-before-outsourcing methods [37], [38], which have been widely recognized to be secure. Therefore, in this section, we mainly focus on the security features of index and trapdoor, and discuss the privacy protection of both of them in detail.

Before discussing the security of our proposed scheme, we first introduce some notions used in the proof process.

- *History*: History contains a set Ω of raw DNA sequences, an index set I_Ω constructed based on Ω, and a set of queries $\mathcal{G} = (\mathbf{Q}_1, \mathbf{Q}_2, \ldots \mathbf{Q}_k)$ sent to the cloud server. They are denoted as a tuple $H = (\Omega, I_\Omega, \mathcal{G})$.
- *View*: View is the encrypted data of H. Specifically, given the secret key $\boldsymbol{sk} = (r_j, S_j, M_{1j}, M_{2j}, S_{kj})$ at jth iteration, the view of H can be described as $V(H) = (Enc_{\boldsymbol{sk}}(\Omega), Enc_{\boldsymbol{sk}}(I_\Omega), Enc_{\boldsymbol{sk}}(\mathcal{G}))$, where $Enc_{\boldsymbol{sk}}(I_\Omega)$ and $Enc_{\boldsymbol{sk}}(\mathcal{G})$ represent the encrypted indexes and trapdoors respectively.
- *Trace of history*: The trace of history $Trace(H)$ denotes the information captured by the cloud server, such as the search and access patten leaked from querying process. Concretely, $Trace(H)$ can be represented as $Trace(H) = (Trace(\mathbf{Q}_1), Trace(\mathbf{Q}_2), \ldots, Trace(\mathbf{Q}_k))$, where $Trace(\mathbf{Q}_s), s = (1, 2, \ldots k)$ indicates the similarity score between encrypted index $I_{\mathbf{A_i}}$ $(i = 1, 2, \ldots, N)$ and trapdoor $T_{\mathbf{Q}_s}$.

Based on the notions illustrated in above, we divide the privacy of index, trapdoor, and the unlinkability of trapdoors into two classes named *Index Privacy* and *Trapdoor Privacy*. Both of them will be analyzed with the definition of indistinguishability under same-closeness-pattern chosen-plaintext attacks (IND-CLS-CPA).

5.1 Formal Security Definitions

1) *Index Privacy:* Informally, in our EFSS _II, the index privacy under IND-CLS-CPA means that given two datasets $\{C_0, C_1\} \subseteq \{\mathbf{A_i'}|i = 1, 2 \cdots N\}$, a P.P.T adversary A can access both the secure index and trapdoor generation algorithm under certain constraints. However, adversary A cannot distinguish the two datasets $\boldsymbol{C_0}$ and $\boldsymbol{C_1}$.

Definition 1 (IND-CLS-CPA Index Privacy). *Let* $\prod = (KeyGen, GenIndex, GenTrapdoor, Query)$ *be a probabilistic EFSS_II scheme based on security parameter* λ*, we design a security game between an adversary* A *and a challenger* B*:*

Initial: Adversary A submits two datasets $\boldsymbol{C_0}$ and $\boldsymbol{C_1}$ with the same number of records to the challenger B, where $\boldsymbol{C_0} = \{\mathbf{A}'_{0,1}, \mathbf{A}'_{0,i}, \ldots \mathbf{A}'_{0,\nu}\}$, and $\mathbf{A}'_{0,i}$ is the set of editing operations of a DNA sequence $\mathbf{A}_{0,i}$.

Setup: Challenger B runs the $keygen(1^\lambda)$ to acquire secret key $\boldsymbol{sk} = (r_j, S_j, M_{1j}, M_{2j}, S_{kj})(j = 1, 2, \ldots, k)$ and keeps it privately.

Phase 1: Adversary A can select a number of requests and send them to the challenger B. These requests include the following types:

- *GenIndex Request:* On the jth GenIndex request, adversary A generates a dataset $\boldsymbol{C_j'} = \{\mathbf{A}'_{j,1}, \mathbf{A}'_{j,i}, \ldots \mathbf{A}'_{j,\nu}\}$ and submits it to the challenger B, where $\mathbf{A}'_{j,i} = (A'_{j,i1}, A'_{j,i2}, A'_{j,im}, \ldots, A'_{j,is})$, $m = (1, 2, \ldots, s)$. For each $A'_{j,im}$, the challenger B uses the secret key r_j to encrypt it as $d'_{A_{j,im}} = \sum_{\mu \in A'_{j,im}} H_{r_j}(\mu)$. Then, $\mathbf{A}'_{j,i}$ will be rewritten as $$\mathbf{A}''_{j,i} = \Bigg(\underbrace{d'_{A_{j,i1}}, -\frac{1}{2}(d'_{A_{j,i1}})^2, \ldots, d'_{A_{j,is}}, -\frac{1}{2}(d'_{A_{j,is}})^2}_{2s}, \underbrace{\beta_{j,i0}, \ldots \beta_{j,i\omega}}_{\omega+1}, \underbrace{\gamma_{j,i0}, \ldots \gamma_{j,i\psi}}_{\psi+1}\Bigg).$$ For $m = 1$ to $m = 2s + \omega + \psi + 2$. $\mathbf{A}''_{j,i}$ will be permuted using secret key S_{kj}, i.e., $\widetilde{\mathbf{A}_{j,i}}(\pi_{S_{kj}}(m)) = \mathbf{A}''_{j,i}(m)$. Next, the challenger B exploits the secret key consisted of a $(2s + \omega + \psi + 2)$ dimensional binary vector S_j and two $(2s + \omega + \psi + 2) \times (2s + \omega + \psi + 2)$ invertible matrices M_{1j}, M_{2j} to further encrypt $\widetilde{\mathbf{A}_{j,i}}$. Finally, the challenger B answers this request with encrypted index as $\boldsymbol{I_{C_j'}} \leftarrow GenIndex(r_j, S_{kj}, S_j, M_{1j}, M_{2j}, \boldsymbol{C_j'})$.
- *GenTrapdoor Request:* On the jth GenTrapdoor request, adversary A selects a plaintext request $\mathbf{Q}_j = (Q_{j,1}, Q_{j,2}, Q_{j,m}, \ldots, Q_{j,s}, \eta^0_j, \eta^1_j, \ldots \eta^\omega_j, \gamma_{j,0}, \gamma_{j,1}, \ldots \gamma_{j,\psi})$ and a set $\mathbf{Q_j^N}$, where $\mathbf{Q_j^N}$ denotes those types of genes that adversary A does not want to query on the jth request. Both of them will be sent to challenger B. For each $Q_{j,m}$, $(m = 1, 2, \ldots s)$, if

$Q_{j,m} \in \mathbf{Q_j^N}$, challenger B sets $d_{Q_{j,m}} = (0,0)$. Otherwise, $d_{Q_{j,m}} = (\sum_{\mu \in Q_{j,m}} H_{r_j}(\mu), 1)$, where r_j indicates the secret key on the jth hash function. Then $\mathbf{Q}_j$ will be rewritten as $\mathbf{Q}'_j = \Big(\underbrace{d_{Q_{j,1}}, d_{Q_{j,2}}, \ldots d_{Q_{j,s}}}_{2s}, \underbrace{\eta_j^0, \eta_j^1, \ldots \eta_j^\omega}_{\omega+1}, \underbrace{\gamma_{j,0}, \gamma_{j,1}, \ldots \gamma_{j,\psi}}_{\psi+1}\Big)$. Challenger B will further permute $\mathbf{Q}'_j$ with the secret key S_{kj}, i.e., $\mathbf{Q}''_j(\pi_{S_{kj}}(m)) = \mathbf{Q}'_j(m)$ for $m = 1$ to $m = 2s + \omega + \psi + 2$. Next, the challenger Bexploits the secret key consisted of a $(2s + Y + U + 1)$ dimensional binary vector S_j and two $(2s + \omega + \psi + 2) \times (2s + \omega + \psi + 2)$ invertible matrices M_{1j}^{-1}, M_{2j}^{-1} to further encrypt $\mathbf{Q}''_j$. Finally, the challenger B answers this request with encrypted trapdoor $T_{\mathbf{Q}_j} \leftarrow GenTrapdoor(r_j, S_{kj}, S_j, M_{1j}^{-1}, M_{2j}^{-1}, Q_j)$.

Please note that both C'_j and $\mathbf{Q}_j$ are subjected to

1) $Trace(Q_0) = Trace(Q_1) = \cdots Trace(Q_k)$, for every $\mathbf{Q}_j, j = (1, 2, \ldots k)$ submitted in *GenTrapdoor request* phase.
2) $(C_0 - C_1) \bigcup (C_1 - C_0) \notin C'_j$.

Challenge: with C_0 and C_1 selected in *Initial*, challenger B throws a coin to decide a variable $b \in \{0, 1\}$, and returns $I_{C_b} \leftarrow GenIndex(r_b, S_{kb}, S_b, M_{1b}, M_{2b}, C_b)$ to the adversary A.

Phase 2: In this phase, a number of indexes and trapdoors generation requests can be also adaptively selected by adversary A and be submitted to the challenger B, but these requests should also subject to the constraint conditions in *Phase 1*.

Guess: Adversary A takes a guess b' of b.

There $\prod$ will be considered secure against same- closeness-pattern chosen-plaintext attacks if for any P.P.T adversary A in the game above, it at most has a negligible advantage

$$\mathbf{Adv}_{\prod,\mathsf{A}}^{\mathsf{IND-CLS-CPA-Index}}(1^\lambda) = \left|pr[b' = b] - \tfrac{1}{2}\right| \leq negl(\lambda),$$

where $negl(\lambda)$ denotes a negligible probability with parameter λ.

2) *Trapdoor Privacy:* Similarly, the trapdoor privacy under IND-CLS-CPA means that given two query requests $\mathbf{Q}_0$ and $\mathbf{Q}_1$, a P.P.T adversary A can access both the secure index and trapdoor generation algorithm under certain constraints. However, adversary A cannot distinguish the two query requests $\mathbf{Q}_0$ and $\mathbf{Q}_1$.

Definition 2 (IND-CLS-CPA Trapdoor Privacy). *Let* $\prod = (KeyGen, GenIndex, GenTrapdoor, Query)$ *be a probabilistic EFSS_II scheme based on security parameter* λ*, we design a security game between an adversary* A *and a challenger* B*:*

Initial: Adversary A submits two query requests $\mathbf{Q}_0$ and $\mathbf{Q}_1$ with same dimensions to challenger B, where $Q_i = (Q_{i,1}, Q_{i,2}, Q_{i,m}, \ldots, Q_{i,s}, \eta_i^0, \eta_i^1, \ldots \eta_i^\omega, \gamma'_{i,1}, \gamma'_{i,2}, \ldots \gamma'_{i,\psi})$, $i = (0, 1)$.

Setup: Challenger B runs the $keygen(1^\lambda)$ to create secret key $\boldsymbol{sk} = (r_j, S_j, M_{1j}, M_{2j}, S_{kj}) (j = 1, 2, \ldots, k)$ and keeps it privately.

Phase 1: Adversary A can select a number of requests sent to the challenge B, and these requests include the following types:

- *GenIndex Request:* On the jth GenIndex request, adversary A generates a dataset $C_j = \{\mathbf{A}'_{j,1}, \mathbf{A}'_{j,i}, \ldots \mathbf{A}'_{j,v}\}$ and submits it to the challenger B, where $\mathbf{A}'_{j,i} = (A'_{j,i1}, A'_{j,i2}, A'_{j,im}, \ldots, A'_{j,is})$, $m = (1, 2, \ldots, s)$. For each $A'_{j,im}$, the challenger B uses the secret key r_j to encrypt it as $d'_{A_{j,im}} = \sum_{\mu \in A'_{j,im}} H_{r_j}(\mu)$. Then, $\mathbf{A}'_{j,i}$ will be rewritten as $\mathbf{A}''_{j,i} = \Big(\underbrace{d'_{A_{j,i1}}, -\frac{1}{2}(d'_{A_{j,i1}})^2, \ldots, d'_{A_{j,is}}, -\frac{1}{2}(d'_{A_{j,is}})^2}_{2s}, \underbrace{\beta_{j,i0}, \ldots \beta_{j,i\omega}}_{\omega+1}, \underbrace{\gamma_{j,i0}, \ldots \gamma_{j,i\psi}}_{\psi+1}\Big)$. For $m = 1$ to $m = 2s + \omega + \psi + 2$. $\mathbf{A}''_{j,i}$ will be permuted by utilizing secret key S_{kj}, i.e., $\widetilde{\mathbf{A}_{j,i}}(\pi_{S_{kj}}(m)) = \mathbf{A}''_{j,i}(m)$. Next, the challenger B exploits the secret key consisted of a $(2s + \omega + \psi + 2)$ dimensional binary vector S_j and two $(2s + \omega + \psi + 2) \times (2s + \omega + \psi + 2)$ invertible matrices M_{1j}, M_{2j} to further encrypt $\widetilde{\mathbf{A}_{j,i}}$. Finally, the challenger B answers this request with encrypted index as $I_{C_j} \leftarrow GenIndex(r_j, S_{kj}, S_j, M_{1j}, M_{2j}, C_j)$.
- *GenTrapdoor Request:* On the jth GenTrapdoor request, adversary A selects a plaintext request $\mathbf{Q}'_j = (Q_{j,1}, Q_{j,2}, Q_{j,m}, \ldots, Q_{j,s}, \eta_j^0, \eta_j^1, \ldots \eta_j^\omega, \gamma_{j,0}, \gamma_{j,1}, \ldots \gamma_{j,\psi})$ and a set $\mathbf{Q_j^N}$, where $\mathbf{Q_j^N}$ denotes those types of genes that adversary A does not want to query on the jth request. Both of them will be sent to challenger B. For each $Q_{j,m}$, $(m = 1, 2, \ldots s)$, if $Q_{j,m} \in \mathbf{Q_j^N}$, challenger B sets $d_{Q_{j,m}} = (0, 0)$, otherwise, $d_{Q_{j,m}} = (\sum_{\mu \in Q_{j,m}} H_{r_j}(\mu), 1)$, where r_j indicates the secret key on the jth hash function. Then $\mathbf{Q}'_j$ will be rewritten as $\mathbf{Q}''_j = \Big(\underbrace{d_{Q_{j,1}}, d_{Q_{j,2}}, \ldots d_{Q_{j,s}}}_{2s}, \underbrace{\eta_j^0, \eta_j^1, \ldots \eta_j^\omega}_{\omega+1}, \underbrace{\gamma_{j,0}, \gamma_{j,1}, \ldots \gamma_{j,\psi}}_{\psi+1}\Big)$. Challenger B will further permute $\mathbf{Q}''_j$ by pseudorandom permutation with the secret key S_{kj}, i.e., $\mathbf{Q}'''_j(\pi_{S_{kj}}(m)) = \mathbf{Q}''_j(m)$ for $m = 1$ to $m = 2s + \omega + \psi + 2$. Next, the challenger Bexploits the secret key consisted of a $(2s + Y + U + 1)$ dimensional binary vector S_j and two $(2s + \omega + \psi + 2) \times (2s + \omega + \psi + 2)$ invertible matrices M_{1j}^{-1}, M_{2j}^{-1} to further encrypt $\mathbf{Q}''_j$. Finally, the challenger B answers this request with encrypted trapdoor $T_{\mathbf{Q}'_j} \leftarrow GenTrapdoor(r_j, S_{kj}, S_j, M_{1j}^{-1}, M_{2j}^{-1}, Q_j)$.

Please note that both C_j and $\mathbf{Q}'_j$ are subjected to

1) $Trace(Q_0) = Trace(Q_1)$, for all C_j submitted in *GenIndex request* phase.
2) $\mathbf{Q}'_j \notin \{Q_0, Q_1\}$.

Challenge: with Q_0 and Q_1 selected in *Initial*, challenger B throws a coin to decide a variable $b \in \{0, 1\}$, and returns $T_{Q_b} \leftarrow GenIndex(r_b, S_{kb}, S_b, M_{1b}, M_{2b}, Q_b)$ to the adversary A.

Phase 2: In this phase, a number of indexes and trapdoors generation requests can be also adaptively selected by adversary A and be submitted to the challenger B, but these requests should also be subjected to the constraint conditions in *Phase 1* as well.

Guess: The adversary A takes a guess b′ of b.

There $\prod$ will be considered secure against same-closeness-pattern chosen-plaintext attacks if for any P.P.T adversary A in the game above, it at most has a negligible advantage

$$\mathbf{Adv}_{\prod,\mathsf{A}}^{\mathsf{IND-CLS-CPA-Trapdoor}}(1^\lambda) = \left|pr[b' = b] - \tfrac{1}{2}\right| \le negl(\lambda),$$

where $negl(\lambda)$ denotes a negligible probability with parameter λ.

5.2 Security Analysis

Theorem 1. *Our EFSS_II is IND-CLS-CPA index secure under the game of Definition 1.*

Proof. The P.P.T adversary A can access the secure index generation algorithm to get plaintext - ciphertext pairs $(\boldsymbol{C'_j}, \boldsymbol{I_{C_j'}})$. However, if A does not know the secret key $\boldsymbol{sk} = (r_j, S_j, M_{1j}, M_{2j}, S_{kj})$ on jth $GenIndex$ request, the split process of $\widetilde{\mathbf{A_i}}^j$ (shown in Fig. 5) is transparent to adversary A, and he needs to reconstruct the two random $(2s+\omega+\psi+2)$-dimensional vectors $D^{j'}_{\mathbf{A_i}}$ and $D^{j'}_{\mathbf{A_i}}$. However, these two vectors will be further encrypted as $M_{1j}D^{j'}_{\mathbf{A_i}}$ and $M_{2j}D^{j''}_{\mathbf{A_i}}$, respectively, where M_{1j} and M_{2j} are also two $(2s+\omega+\psi+2)\times(2s+\omega+\psi+2)$-dimensional unknown matrices to adversary A. Thus, there are a total of $2(2s+\omega+\psi+2)|\widetilde{\mathbf{A_i}}^j|$ unknown variables in $D^{j'}_{\mathbf{A_i}}$ and $D^{j''}_{\mathbf{A_i}'}$, and $2(2s+\omega+\psi+2)^2$ unknown variables in M_{1j} and M_{2j}. Therefore, it is very hard for adversary A to restore the matrixes M_{1j} and M_{2j}. Although some research results [37], [39] can address this problem partly by linear analysis, instead of restoring the matrixes M_{1j} and M_{2j}. However, it is unrealistic to find $(2s+\omega+\psi+2)$ linear independent equations in our scheme since more than 99.5 percent of the DNA sequences between any two irrelevant people are identical [8]. Moreover, even an attacker can recover the two vectors $D^{j'}_{\mathbf{A_i}}$ and $D^{j'}_{\mathbf{A_i}'}$, and obtain the value of $d_{A'^j_{im}} = \sum_{\mu\in A'_{im}} H_{r_j}(\mu)$ for $1 \le m \le s$, he still cannot get any information about the contents of A'_{im} due to the one-way property of hash function. Therefore, our EFSS_II is IND-CLS-CPA index secure under the game of Definition 1. □

Theorem 2. *Our EFSS_II is IND-CLS-CPA trapdoor secure under the game of Definition 2.*

Proof. Because the trapdoor is generated in the same way compared with encrypted index, the security analysis of trapdoor privacy under IND-CLS-CPA can be achieved by a similar way. For simplicity, here we omit the specific proof process. □

Unlinkability of Trapdoor. To demonstrate the unlinkability of Trapdoors, here we estimate the probabilities of collision occurrence between two trapdoors.

As discussed in Section 4.2, the trapdoor is made up of $T^j_{\mathbf{Q}} = \{M_{1j}D^{j'}_{\mathbf{Q}}, M_{2j}D^{j''}_{\mathbf{Q}}\}$. Specifically, given a query request $\mathbf{Q_j}$, each element of it will be first mapped into $\{-1, 1\}$ through the hash function. For convenience, we assume that there are a total of 2^σ possible mapped results. Next, a pseudorandom permutation function π is also used to permute the position of each element in $\mathbf{Q^{j'}}$. It also has $(2s+\omega+\psi+2)$ possible replaced positions. Besides, assume the size of each dummy noise added in $\mathbf{Q'_j}$ is φ bits, there are $\omega 2^\varphi$ possible options for dummy noise. Next, as shown in Fig. 5, we use a $(2s+\omega+\psi+2)$ dimensional binary vector S_j to split the vector $\widetilde{\mathbf{Q}^{j'}}$, where $\widetilde{\mathbf{Q}^{j'}}(m)$ will be split into two random values (i.e., $D^{j'}_{\mathbf{Q}}(m)$ and $D^{j''}_{\mathbf{Q}}(m)$) when $S_j(m) = 0$. Thus, assume the number of '0' in S_j is ν, and each dimension of $D^{j'}_{\mathbf{Q}}(m)$ and $D^{j''}_{\mathbf{Q}}(m)$) is ι bits, there are a total of $2^{\iota\nu}$ possible values for split process. Note that φ, ν and ι are selected independently to each other. Therefore, for any two trapdoors, we can calculate that the probability of them are identical as follows.

$$\begin{aligned} P_1 &= \frac{1}{2^\sigma\cdot(2s+\omega+\psi+2)!\cdot\omega 2^\varphi\cdot 2^{\iota\nu}} \\ &= \frac{1}{(2s+Y+U+1)!\cdot\omega 2^{\sigma+\varphi+\iota\nu}}. \end{aligned} \tag{8}$$

Therefore, if we select a lager φ, ν and ι, we can ensure the higher security.

6 Performance Evaluation

In this section, we conduct experiments to evaluate the performance of our proposed schemes. The experimental configuration is as follows: All the experiments are compiled in the Python language with version 3.7.4, where each user (i.e., the data owner or search user) is simulated by one HP i7 8550u notebook, which has one 1.8GHZ CPU, 8G flash memory, 256SSD and 500G mechanical hard disk, and assigned with 64-bit Windows 10 operating system. The " Cloud " is simulated with two Lenovo servers which have 2 Intel(R) Xeon(R) E5-2620 2.10GHZ CPU, 64 GB RAM, 512SSD, 4 TB mechanical hard disk and runs on the Ubuntu 18.04 operating system. The Wagner-Fischer algorithm [35] is adopted to convert each raw DNA sequence into its set of editing operations. The secret key in each round j, i.e., the $(2s+\omega+\psi+2)$ dimensional binary vector S_j and two $(2s+\omega+\psi+2)\times(2s+\omega+\psi+2)$ dimensional invertible matrices M_{1j}, M_{2j}, are generated under the guidance of the standard secure kNN computation [36]. The 2-wise universal hash function H_{r_j} in each round j is selected from `murmurhash64` [40], which can be used to customize a hash function with a fixed range of values. In addition, the pseudorandom permutation function π used in our model is replaced by the classic algorithm Fisher-Yates-shuffle [41].

We select data in batches from the publicly available 1,000 Genomes Project [34]. Specifically, to demonstrate the matching accuracy of our protocol under different genomic sequences, we choose genomic sequences on five distinct chromosomes, i.e., Chromosome 2 (133019901C133020615) (contain 1,888 genes), Chromosomes 3 (75785026-75788496)(contain 1,469 genes), Chromosome 21 (9907190C9909277) (contain 357 genes), Chromosome 22 (22720578C22722138)(contain 545 genes), and Chromosome Y (10027986-10029907) (contain 86 genes). As mentioned before, every raw DNA sequence will be converted to its set of editing operations and stored in a VCF (Variation Call Format) file. Please note that we unify all the files to a fixed size and make each file contain the same length of DNA. It will be beneficial to the implementation of the entire program. For more persuasive, the state-of-the-art works [8], [9], [12] are

TABLE 3
Comparison of Functionality With Existing Models

Literature \ Function	Matching	Threat model	Access control	Boolean query	Interaction	Data type
[33]	Similarity	Honest-but-curious	No	No	Non-interactive	Textual
[32]	Similarity	Honest-but-curious (two-servers)	No	No	Interactive	Textual
[9]	Similarity	Honest	No	No	Interactive	Genetic
[8], [11]	Similarity	Honest-but-curious	No	No	Interactive	Genetic
[10]	Similarity	Honest-but-curious (two-servers)	No	No	Interactive	Genetic
[12]	Similarity	Honest-but-curious	No	No	Non-interactive	Genetic
[22], [23]	Similarity	Honest-but-curious	No	No	Non-interactive	Textual
[13], [30]	Exact	Honest-but-curious	Yes	No	Non-interactive	Textual
[31]	Exact	Honest-but-curious	No	Yes	Non-interactive	Textual
[17]	Exact	Honest-but-curious	Yes	Yes	Non-interactive	Textual
[18]	Exact	Honest-but-curious	No	Yes	Non-interactive	Textual
Our model	Similarity	Honest-but-curious	Yes	Yes	Non-interactive	Genetic

also exploited to compare the performance (i.e., computation and communication costs) with our EFSS. These works are very similar to ours, as they also focus on DNA data and are committed to designing efficient similarity queries among gene sequences in the ciphertext environment.

6.1 Functionality

We first compare the functionality of our model with state-of-the-art works. As shown in Table 3, keyword/string matching in the ciphertext is not an emerging research field, and many privacy-preserving approaches have been designed and applied to different scenarios. However, we can see that existing models (such as [8], [9], [11]) for genomic data always require multiple interactions between the user and the server. This is mainly due to the nature of the length of the genome sequence. As introduced before, the length of a gene sequence always ranges from hundreds of thousands to millions of base pairs [8]. To speed up the matching between DNA sequences in ciphertext, an alternative is to outsource most ciphertext computing operations to two non-collusive servers [11] or to perform matching with the assistance of users providing decryption services [8], [9]. However, in real-world applications, it is difficult to ensure that the two service providers do not collude, even if the two entities belong to completely different operators [29], [42]. On the other hand, SMC based works [8], [9] requires all parties remaining online during the entire querying process, which significantly limits the robustness of these solutions to sporadic events such as network delays and device damage. Scheme [12] breaks the defects of [8], [9] in communication. However, it uses the traditional predicate encryption scheme [43] based on bilinear mapping during the string matching process, which inevitably leads to high computation and communication overheads. On the other hand, to our best knowledge, existing solutions for genomic data are unable to support the requirements of Boolean queries and data access control. Although there have been some research results on data access control [13], [30] and Boolean query [17], [18], [31] in the field of text-based ciphertext searching, as explained before, it will incur huge computation cost if we directly exploit them to perform privately DNA sequence matching. Moreover, the heterogeneity of the application scenarios also makes these solutions difficult to be adopted in DNA queries. Compared with these schemes, our EFSS is non-interactive, and can efficiently support mixed Boolean queries as well as data access control over encrypted DNA data. We first design a private approximation algorithm to convert the edit distance computation problem to the symmetric set difference size approximation problem, which can significantly reduce the number of elements that need to be matched under ciphertext. Then, we design a novel Boolean search method to achieve the complicated logic query such as mixed "AND" and "NO" operations on genes. Moreover, since the search strategy can avoid unnecessary DNA sequences matched during the query process, users can also gain the benefit of receiving accurate results with a short period. In the end, by utilizing a variant of polynomial based design, EFSS is able to support data access control during the query process, where each DNA sequence is accessible to users who are authorized.

6.2 Accuracy

In this section, we analyze the query accuracy of our EFSS. It is worth noting that EFSS_II is the improvement of EFSS_I, which is an extension of its functionality without affecting the accuracy of EFSS_I. For the sake of simplicity, here we only take EFSS_II as an example. As discussed in Section 4.1, we exploit the CLN Theorem to estimate the expectation of $(d_{\mathbf{A}'_i} - d_{\mathbf{Q}'})^2, i \in \{1, 2\}$, which inevitably introduces errors in the matching process. To a certain extent, it harms the accuracy of the final returned results. We have given a formal theoretical analysis to show that this error can be mitigated by increasing the cycle times k. To verify our argument, we choose genomic sequences on five distinct chromosomes from the publicly available 1,000 Genomes Project [34]. The specific characteristics of the selected datasets are shown in Table 4. We can observe that each chromosome contains at least hundreds of genes, which constitutes a DNA sequence based on tens of millions of base pairs. However, for different individuals, the variability between two genomic sequences belonged to the same type of chromosome is very small ($\leq 6.16\%$). This verifies that most of the genome sequences between different people are the same, which is also the prerequisite for the smooth execution of our approximation algorithm (refer to Section 3.2). Then, we outsource the ciphertexts of selected datasets to the cloud and ask it to return top-$\mathcal{K}$ closest results for a given search sequence.

As shown in Table 5, EFSS_II is able to return fairly correct results under the different datasets. Although EFSS_II

TABLE 4
Characteristics of the Selected Datasets

Dataset \ Characteristic	Number of genes	Length	Ave.- △ [1]	Max.-△ [2]	Variability [3]
Chromosome 2	1,888	242193529	4070479.48	11193818.56	$\leq 4.63\%$
Chromosome 3	1,469	198295559	6351472.95	12214371.05	$\leq 6.16\%$
Chromosome 21	357	47347896	612550.29	1225100.58	$\leq 2.59\%$
Chromosome 22	545	48674127	93007.89	186015.77	$\leq 0.39\%$
Chromosome Y	86	59363566	892280.06	1978785.53	$\leq 3.34\%$

[1] *Ave.-△ denotes the average of all edit distances between two genomic sequence in the dataset.*
[2] *Max.-△ denotes the maximal edit distances between two genomic sequence in the dataset.*
[3] *Variability is the ratio of the maximum edit distance to the length of the chromosome in the current database.*

inevitably returns a small number of erroneous results as the value of $\mathcal{K}$ (i.e., the top-$\mathcal{K}$ closest results to the search sequence) increases, it still shows excellent performance in accuracy ($\geq$90%). Besides, as analyzed in Section 4.1, the query accuracy can be improved by adjusting the number of the cycle times k, which means that increasing the number of cycle times k is beneficial to the accuracy. Of course, this is at the expense of additional computation and communication overhead (analyzed in Sections 6.3 and 6.4). On the other hand, we can see that EFSS_II shows a higher accuracy on chromosomes 21 and 22 than on chromosomes 2 and 3. This is because the chromosomes 21 and 22 have a lower variability rate than those of the other two types of chromosomes, which helps our approximation algorithm to perform better under these datasets.

6.3 Communication Overhead

We now discuss the communication overhead of our proposed schemes, i.e., EFSS_I, EFSS_II and EFSS_II(ES). Besides, the state-of-the-art works [8], [9], [12] are also analyzed in this part to present more visibility. These works are very similar to ours, as they also focus on DNA data and are committed to designing efficient similarity queries among gene sequences in the ciphertext environment.

TABLE 5
Accuracy of EFSS_II

Dataset	k	$\mathcal{K}=1$	$\mathcal{K}=3$	$\mathcal{K}=5$	$\mathcal{K}=7$
Chromosome 2	5	96.48%	93.35%	91.18%	90.43%
	10	97.33%	96.57%	94.23%	92.64%
	15	98.47%	97.34%	95.40%	95.01%
Chromosome 3	5	96.27%	94.03%	90.86%	90.13%
	10	97.38%	96.59%	94.73%	93.17%
	15	98.49%	97.19%	94.93%	95.17%
Chromosome 21	5	99.46%	96.34%	95.12%	94.33%
	10	100%	99.74%	98.32%	97.15%
	15	100%	100%	99.73%	98.37%
Chromosome 22	5	99.57%	96.39%	95.07%	94.68%
	10	100%	99.49%	98.20%	97.67%
	15	100%	100%	99.81%	98.50%
Chromosome Y	5	99.18%	96.21%	95.36%	94.01%
	10	99.38%	99.17%	98.81%	96.63%
	15	99.47%	99.25%	98.97%	97.16%

6.3.1 Theoretical Analysis

The theoretical analysis results are shown in Table 6. For the convenience of the description, we first divide the communication overhead of non-interactive schemes (i.e., EFSS_I, EFSS_II, EFSS_II(ES) and [12]) into three parts (i.e., **Index generation**, **Trapdoor generation** and **Query**), and then discuss the cost of each part in detail. Conversely, for interactive protocols [8] and [9], since the generated overhead stems from the inseparable multiple interactions between the server and the search user, we only summarize their total communication overhead.

Specifically, in the phase of **Index generation**, for EFSS_I, the data owner needs to send encrypted index of $\mathbf{A_i}$ as $I^j_{\mathbf{A'_i}} = (d_{A'^j_{i1}}, d_{A'^j_{i2}}, d_{A'^j_{im}}, \ldots, d_{A'^j_{is}})$ to the cloud server, where $j \in [1, k]$ and $i \in [1, N]$. Assume that the size of each $d_{A'^j_{im}}$ (i.e., the sum of all $H_{hr}(\mu), \mu \in A'_{im}$ at jth iteration) is α bits, the total size of the encrypted index for N genome genes is $N \times k \times (s \times \alpha)$ bits. For EFSS_II, since each encrypted index sent to the cloud server is formed as $I^j_{\mathbf{A_i}} = \{M_{1j}D^{j'}_{\mathbf{A_i}}, M_{2j}D^{j''}_{\mathbf{A_i}}\}$, where M_{1j}, M_{2j} are two $(2s+\omega+\psi+2) \times (2s+\omega+\psi+2)$ dimensional invertible matrices, the total cost for N genome genes are $2N \times k \times (2s+\omega+\psi+2) \times \alpha$ bits. Different from EFSS_II, EFSS_II(ES) requires the data owner to divide all DNA sequences into K classes, and generates the corresponding encrypted indexes for N genome genes and K cluster centers. Therefore, the total ciphertext size is $2k \times (N+K) \times (2s+\omega+\psi+2) \times \alpha$ bits. For work [12], given a DNA sequence $\mathbf{A_i}$ with length $\mathcal{T}$, it first adopts the suffix tree structure to represent the DNA sequence, which will produce $\frac{\mathcal{T}(\mathcal{T}+1)}{2}$ subsequences for $\mathbf{A_i}$. Then, it needs to generate four ciphertexts (assume that the size of each ciphertext is also α bits) for each subsequence and sends it to the server. Thus, this process requires a total of $4N \times \alpha \times \frac{\mathcal{T}(\mathcal{T}+1)}{2}$ bits to be transmitted for N genome genes. In the phase of **Trapdoor generation**, for EFSS_I, given a query request, i.e., a genome gene $\mathbf{Q}$, the search user needs to send the trapdoor (i.e., the encrypted search request) as $T^j_{\mathbf{Q'}} = (d_{Q'^j_1}, d_{Q'^j_2}, d_{Q'^j_m}, \ldots, d_{Q'^j_s})$ to the cloud server. Thus, the total size of the trapdoor is $k \times (s \times \alpha)$ bits. Since the trapdoor generated by EFSS_II and EFSS_II(ES) are consistent, that is, sending $T^j_{\mathbf{Q}} = \{M_{1j}D^{j'}_{\mathbf{Q}}, M_{2j}D^{j''}_{\mathbf{Q}}\}$ to the server for a given query request $\mathbf{Q}$, the cost of both EFSS_II and EFSS_II(ES) are $2k \times (2s+\omega+\psi+2) \times \alpha$ bits. For work [12], given a query request $\mathbf{Q}$, the search user needs to generate $(\frac{\mathcal{T}(\mathcal{T}+1)}{2}+2)$ ciphertexts and submits them to the cloud, where $\frac{\mathcal{T}(\mathcal{T}+1)}{2}$

TABLE 6
Comparison of Communication Overhead (Bits)

Model	Communication Overhead		
	Index generation	**Trapdoor generation**	**Query**
EFSS_I	$N \times k \times (s \times \alpha)$	$k \times (s \times \alpha)$	$\mathcal{K} \times \alpha$
EFSS_II	$2N \times k \times (2s + \omega + \psi + 2) \times \alpha$	$2k \times (2s + \omega + \psi + 2) \times \alpha$	$\mathcal{K} \times \alpha$
EFSS_II (ES)	$2k \times (N + K) \times (2s + \omega + \psi + 2) \times \alpha$	$2k \times (2s + \omega + \psi + 2) \times \alpha$	$\mathcal{K} \times \alpha$
[12]	$4N \times \alpha \times \frac{\mathcal{T}(\mathcal{T}+1)}{2}$	$(\frac{\mathcal{T}(\mathcal{T}+1)}{2} + 2) \times \alpha$	$\mathcal{K} \times \alpha \times \frac{\mathcal{T}(\mathcal{T}+1)}{2}$
[8]	$(N \times \tau \times \alpha) + N(\vert\mathtt{GC}\vert + 2)\alpha + (\mathcal{K} \times \alpha)$		
[9]	$(N \times \digamma \times \alpha) + N \times \digamma(\vert\mathtt{GC}\vert + 2)\alpha + (\mathcal{K} \times \alpha)$		

ciphertexts are for matching under encrypted domain, and the other two ciphertexts are for verification. As a result, this require a total of $(\frac{\mathcal{T}(\mathcal{T}+1)}{2} + 2) \times \alpha$ bits to be transmitted. In the **Query** phase, since all EFSS_ I, EFSS_II and EFSS_II(ES) are required to return $\mathcal{K}$ closest results, the communication overhead incurred in this process is $\mathcal{K} \times \alpha$. For work [12], since each query sequence is divided into $\frac{\mathcal{T}(\mathcal{T}+1)}{2}$ subsequences, the server needs to return $\mathcal{K} \times \frac{\mathcal{T}(\mathcal{T}+1)}{2}$ results for all subsequences, which produces a total cost of $\mathcal{K} \times \alpha \times \frac{\mathcal{T}(\mathcal{T}+1)}{2}$ bits.

Next, we analyze the communication cost of works [8] and [9]. As described before, both [8] and [9] resort to multiple interactions between the server and the user to support string matching between DNA sequences under ciphertext. To achieve this, garbled circuits $\mathtt{GC}$ [28] is exploited to provide secure interaction. For simplicity, we assume that the size of the garbled circuit constructed in [8] and [9] is $|\mathtt{GC}|$ bits. For work [8], the size of each encrypted DNA sequence is $\tau \times \alpha$ bits, where τ denotes the number of hash functions used to generate encrypted DNA sequences. Thus, the total communication overhead required for N sequences is $(N \times \tau \times \alpha)$ bits. In the query process, for each pair of sequences, the server and each user need to interact once to obtain the edit distance between the two sequences. To return $\mathcal{K}$ closest results, a total of $N(|\mathtt{GC}| + 2)\alpha + (\mathcal{K} \times \alpha)$ bits of data need to be transmitted between the server and the user. Therefore, the total communication overhead required for [8] is $(N \times \tau \times \alpha) + N(|\mathtt{GC}| + 2)\alpha + (\mathcal{K} \times \alpha)$ bits. Work [9] also divides each DNA sequence into multiple subsequences before performing matching in the ciphertext. Assume that each DNA sequence is divided into $\digamma$ subsequences, the data owner needs to upload $(N \times \digamma \times \alpha)$ bits of ciphertext to the server for N genome genes. In query process, to return $\mathcal{K}$ closest results, a total of $N \times \digamma(|\mathtt{GC}| + 2)\alpha$ bits of data need to be transmitted between the server and the user. Hence, the total communication overhead required for [9] is $(N\times \digamma \times \alpha) + N \times \digamma(|\mathtt{GC}| + 2)\alpha + (\mathcal{K} \times \alpha)$ bits.

6.3.2 *Experimental Results*

We conduct experiments to further demonstrate the communication overhead of our EFSS. From the above analysis, we are convinced that the communication complexity of EFSS (i.e., EFSS_I, EFSS_II, EFSS_II(ES)) is mainly related to the number of genomes N in the dataset and the number of cycles k in the execution process. Without loss of generality, the dataset used in experiments is selected from batches in chromosome 21 which contains 700 independent genomes. Each genome has a length of 47347896 and contains 357 different gene sequences. As before, we also divide the scheme into three parts (i.e.,**Index generation**, **Trapdoor generation** and **Query**) and discuss the communication overhead of each part separately.

Index generation. The communication cost in the phase of **Index generation** is shown in Fig. 6. We can observe that the amount of data generated by EFSS increases linearly with the number of genomes/cycles in the system. Compared with EFSS_I, EFSS_II increases the dimensions of each encrypted index from s to $(2s + \omega + \psi + 2)$ for supporting the functions of Boolean query and data access control (the number of users' role/dummy noises are respectively set as 20 and 10 in our experiments), which inevitably incurs additional communication overhead. Similarly, different from EFSS_II, EFSS_II(ES) requires the data owner to divide all DNA sequences into K classes, and sends the additional K cluster centers to the cloud in addition the encrypted indexes for N genome genes. Therefore, compared with EFSS_I and EFSS_II, EFSS_II(ES) requires more communication resources to submit all encrypted indexes to the cloud.

Trapdoor generation. Given a query request, i.e., a genome gene $\mathbf{Q}$, we can see that the cost of generating a single trapdoor of $\mathbf{Q}$ is mainly related to the number of k (shown in Table 6). Fig. 7 presents the communication cost of generating a single trapdoor as the number of cycle times k increases. Similar to the previous, since EFSS_II increases the dimensions of the trapdoor from s to $(2s + \omega + \psi + 2)$, the size of the trapdoor generated by EFSS_II is slightly larger than the one EFSS_I generated. Also, since the trapdoor generated by

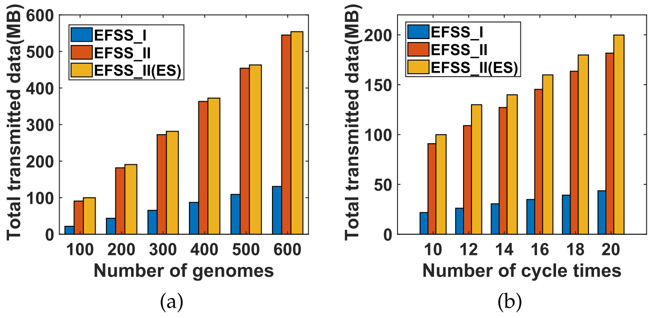


Fig. 6. Communication overhead for building all encrypted indexes. (a) For the different number of genomes with the same number of cycle times, $k = 10$. (b) For the different number of cycle times with the same number of genomes, $N = 100$.

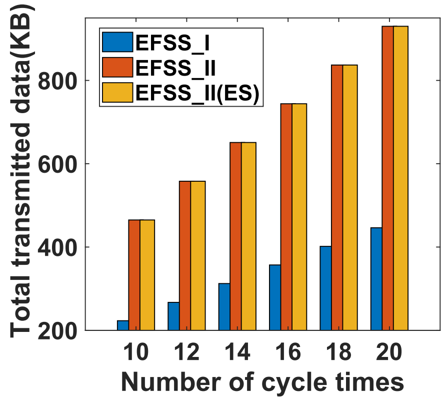


Fig. 7. Communication overhead for building a trapdoor.

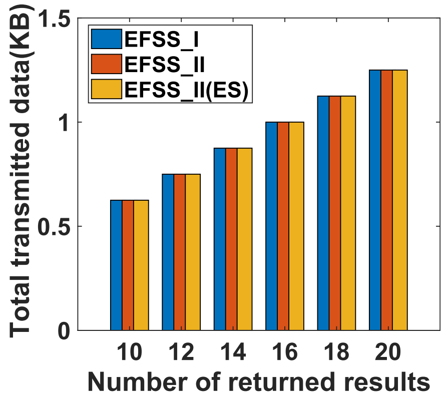


Fig. 8. Communication overhead for returning $\mathcal{K}$ results.

EFSS_II and EFSS_II(ES) are consistent, the communication overhead generated by these two schemes is the same.

Query. Table 6 shows that the communication overhead of the EFSS_I, EFSS_II and EFSS_II(ES) is only related to the number of returned results $\mathcal{K}$ during the query phase. Fig. 8 presents the communication cost as the number of returned results $\mathcal{K}$ increases. Due to the consistency of the communication complexity of the three schemes, we can see that the cost of them is the same.

Comparison with existing approaches. To demonstrate the efficiency of our EFSS, we also conduct experiments to compare EFSS with the existing approaches [8], [12] and [9]. For impartiality, all comparative experiments were done in the same experimental environment, i.e., each user is simulated by one HP i7 8550u notebook, which has one 1.8 GHZ CPU, 8 G flash memory, 256SSD, and 500 G mechanical hard disk, and assigned with 64-bit Windows 10 operating system. The "Cloud" is simulated with two Lenovo servers which have 2 Intel(R) Xeon(R) E5-2620 2.10 GHZ CPU, 64 GB RAM, 512SSD, 4 TB mechanical hard disk and runs on the Ubuntu 18.04 operating system. The dataset used in experiments is also selected from batches in chromosome 21, which contains 700 independent genomes. Each genome has a length of 47347896 and contains 357 different gene sequences. For work [12], we adopt the cryptographic primitive of predicate encryption [43] and Java Pairing Based Cryptography Library (jPBC) [44] to implement its proposed scheme, which is also the same as the configuration used in work [12]. Please note that since we use Python language to implement all the simulation experiments in this paper, we made a little modification of implementing work [12], that is, using the JPype [45] combined with Python to call the package of jPBC. For work [8], the implementation is constructed by exploiting the garbled circuit protocol designed by [46] and the oblivious transfer (OT) with OT extension [47] (consistent with the configuration in [8]). In addition, for impartiality, the 4-wise universal hash function used in [8] is also selected from `murmurhash64` [40]. The implementation of [9] is built under the Secure Computation API library (SCAPI) [26], where the adopted techniques (i.e., the Yao with free-XOR technique [25] and the OT extension [27]) are consistent with [9].

As shown in Table 7, we can observe that our EFSS shows superior performance in communication overhead compared to [8], [12] and [9]. This is mainly due to the following reasons: For work [12], given a DNA sequence $\mathbf{A_i}$ with length $\mathcal{T}$, it first adopts $\frac{\mathcal{T}(\mathcal{T}+1)}{2}$ subsequences to represent $\mathbf{A_i}$, and then sends a total of $4N \times \frac{\mathcal{T}(\mathcal{T}+1)}{2}$ ciphertexts to the cloud. This will generate huge communication overhead for the long genomic sequence (the length of each genome is 47347896 in our experiments). Besides, as illustrated in Table 6, compared with EFSS, [12] is required to generate more ciphertexts in the phases of both **Trapdoor generation** and **Query**. This inevitably incurs additional communication costs. Both [8] and [9] resort to garbled circuits $\mathtt{GC}$ [28] to support string matching between DNA sequences under ciphertext. However, given a query sequence $\mathbf{Q}$, the $\mathtt{GC}$ protocol will be called N and $N \times \digamma$ times in [8] and [9], respectively. We know that the $\mathtt{GC}$ is always large even for building a simple function. For example, in our experiments, the average number of gates in the circuit is 3149350. This is very large and requires huge communication cost to perform string matching between $\mathbf{Q}$ and N DNA sequences. Different from

TABLE 7
Comparison of Communication Overhead With Different Number of Genome Sequences

Literature \ Genomes	$N = 100$	$N = 200$	$N = 300$	$N = 400$	$N = 500$	$N = 600$
EFSS_I	21.78(MB)	43.57(MB)	65.36(MB)	87.15(MB)	108.95(MB)	130.73(MB)
EFSS_II	90.82(MB)	181.64(MB)	272.05(MB)	363.28(MB)	454.10(MB)	544.92(MB)
EFSS_II(ES)	99.9(MB)	190.72(MB)	281.54(MB)	372.36(MB)	463.18(MB)	554(MB)
[12]	28.23(GB)	56.47(GB)	84.69(GB)*	112.89(GB)*	142.10(GB)*	169.32(GB)*
[8]	242.3(MB)	484.6(MB)	726.9(MB)	969.2(MB)	1.183(GB)	1.419(GB)
[9]	1.781(GB)	3.562(GB)	5.343(GB)	7.125(GB)	8.906(GB)	10.68(GB)

**Since recording the amount of data generated in work [12] is extremely time consuming, the communication cost with symbol * is extrapolated based on the complexity shown in Table 6.*

TABLE 8
Comparison of Computation Overhead

Model	Computation Overhead		
	Index generation	**Trapdoor generation**	**Query**
EFSS_I	$Nk(\mathcal{HT} + (\mathcal{T} - 1)\mathcal{D})$	$k(\mathcal{HT} + (\mathcal{T} - 1)\mathcal{D})$	$N(k\mathcal{P} + k\mathcal{D} + \mathcal{M})$
EFSS_II	$Nk(\mathcal{HT} + (\mathcal{T} - 1)\mathcal{D} + (s-1)\mathcal{M} +$ $(2s + \omega + \psi + 2)\mathcal{L} + 2\mathcal{P})$	$k(\mathcal{HT} + (\mathcal{T} - 1)\mathcal{D} + (2s + \omega + \psi + 2)\mathcal{L} + 2\mathcal{P})$	$N(k\mathcal{P} + k\mathcal{D} + \mathcal{M})$
EFSS_II	$(N + \mathcal{K})k(\mathcal{HT} + (\mathcal{T} - 1)\mathcal{D} + (s-1)\mathcal{M} +$ $(2s + \omega + \psi + 2)\mathcal{L} + 2\mathcal{P})$	$k(\mathcal{HT} + (\mathcal{T} - 1)\mathcal{D} + (2s + \omega + \psi + 2)\mathcal{L} + 2\mathcal{P})$	$\frac{2N}{\mathcal{K}}(k\mathcal{P} + k\mathcal{D} + \mathcal{M})$
[12]	$2N\mathcal{T}(\mathcal{T} + 1)(2\mathcal{M}_R + (r_1 + r_2 + r_3 + r_4)\mathcal{E}_P)$	$2\mathcal{T}(\mathcal{T} + 1)(\mathcal{M}_R + (r_5 + r_6 + r_7 + r_8)\mathcal{E}_M)$	$\frac{3N\mathcal{T}(\mathcal{T}+1)}{2}\mathcal{B}_T$
[8]	$N\tau(\mathcal{HT} + \mathcal{O}_{\texttt{GC}}) + N\tau\mathcal{D} + N\mathcal{M}$		
[9]	$N\digamma\mathcal{O}^*_{\texttt{GC}}$		

Work [9] has other types of operations, such as blocking the DNA sequence used for the query, and comparing operations between strings to find the top $\mathcal{K}$ results. However, these operations can be considered negligible relative to performing garbled circuits.

[8], [12] and [9], our EFSS does not require multiple interactions between the server and the user. Moreover, we design a private approximation algorithm to convert the edit distance computation problem between two sequences to the symmetric set difference size approximation problem, which can significantly reduce the number of elements that need to be matched under ciphertext. Based on this, for each DNA sequence $\mathbf{A_i}$, we just require the data owner to send k encrypted indexes to the cloud, where a constant k (such as k = 10) is enough to provide a good query accuracy(shown in Section 6.2). Therefore, compared with works [8], [12] and [9], our EFSS shows superior performance in communication overhead.

6.4 Computation Overhead

In this section, we discuss the computation overhead of EFSS, i.e., EFSS_I, EFSS_II and EFSS_II(ES). Similarly, the state-of-the-art works [8], [9], [12] are also analyzed in this part to present more comparability.

6.4.1 *Theoretical Analysis*

The theoretical analysis results are shown in Table 8. We also divide the cost of non-interactive schemes (i.e., EFSS_I, EFSS_II, EFSS_II(ES) and [12]) into three parts (i.e., **Index generation**, **Trapdoor generation** and **Query**), and then discuss the cost of each part in detail. Conversely, for interactive protocols [8] and [9], we only summarize their total communication overhead. To facilitate the description of the various types of operations involved in EFSS, we use the symbol $\mathcal{H}$ to represent a hash operation. One addition and multiplication operations are represented by $\mathcal{D}$ and $\mathcal{M}$, respectively. Besides, the symbol $\mathcal{P}$ is used to represent an inner product operation of a matrix and a vector, and $\mathcal{L}$ is a pseudo-random permutation. For [12], we use $\mathcal{E}_P$ to represent an exponential operation under group P, and $\mathcal{M}_R$ represents a multiplication operation under group R. Besides, we use $\mathcal{B}_T$ to represent a bilinear map under the group T. For [8] and [9], we use $\mathcal{O}_{\texttt{GC}}$ to indicate a garbled circuit protocol called between the server and a user.

Specifically, in the phase of **Index generation**, for EFSS_I, given the editing operations $\mathbf{A'_i}$ of a DNA sequence $\mathbf{A_i}$, the data owner first converts each element $\mu \in \mathbf{A'_i}$ into a single integer $H_{r_j}(\mu) \in \{-1, 1\}$, and then calculates the sum of all $H_{hr}(\mu), \mu \in A'_{im}$. This total requires $Nk(\mathcal{HT})$ hash operations and $Nk(\mathcal{T} - 1)\mathcal{D}$ additions. For EFSS_II, given a set of editing operations $\mathbf{A'_i}$ with length of $\mathcal{T}$, the data owner requires $\mathcal{HT}$ hash operations to convert each element $\mu \in \mathbf{A'_i}$ into a single integer $H_{r_j}(\mu) \in \{-1, 1\}$. Then, the data owner needs $(\mathcal{T} - 1)\mathcal{D}$ additions to calculate $d_{A'^j_{im}}$ for every $m \in [1, s]$. Different from EFSS_I, EFSS_II requires $(2s + \omega + \psi + 2)\mathcal{L}$ pseudo-random permutations to change the position of each element, and resorts to $2\mathcal{P}$ inner products between the vector and the matrix to obtain the encrypted index $I^j_{\mathbf{A_i}}$. Therefore, the total number of mathematical operations for N sequences is $(N + \mathcal{K})k(\mathcal{HT} + (\mathcal{T} - 1)\mathcal{D} + (s - 1)\mathcal{M} + (2s + \omega + \psi + 2)\mathcal{L} + 2\mathcal{P})$. The computation cost of EFSS_II(ES) in the phase of **Index generation** is similar to EFSS_II. Compared to EFSS_II, EFSS_II(ES) requires the data owner to divide all DNA sequences into K classes and generate encrypted indexes for the K cluster centers. Therefore, it needs extra computation overhead to complete this process (shown in Table 8). For work [12], given a DNA sequence $\mathbf{A_i}$ with length $\mathcal{T}$, it first adopts the suffix tree structure to represent the DNA sequence, which will produce $\frac{\mathcal{T}(\mathcal{T}+1)}{2}$ subsequences for $\mathbf{A_i}$. Then, it needs to generate four ciphertexts for each subsequence. Thus, this process requires a total of $2N\mathcal{T}(\mathcal{T} + 1)(2\mathcal{M}_R + (r_1 + r_2 + r_3 + r_4)\mathcal{E}_P)$ mathematical operations for N sequences, where r_1 to r_4 are the random values selected by the data owner.

In the phase of **Trapdoor generation**, for EFSS_I, given a query request, i.e., a genome gene $\mathbf{Q}$, the search user requires $k(\mathcal{HT})$ hash operations to convert each element $\mu \in \mathbf{Q}'$ into a single integer $H_{r_j}(\mu) \in \{-1, 1\}$, and $k(\mathcal{T} - 1)\mathcal{D}$ additions to generate the trapdoor. Since the trapdoor generated by EFSS_II and EFSS_II(ES) are consistent, as shown in Table 8, the cost of the two schemes is $k(\mathcal{HT} + (\mathcal{T} - 1)\mathcal{D} + (2s + \omega + \psi + 2)\mathcal{L} + 2\mathcal{P})$. For work [12], given a query request $\mathbf{Q}$, the search user needs to generate $(\frac{\mathcal{T}(\mathcal{T}+1)}{2} + 2)$ ciphertexts, where $\frac{\mathcal{T}(\mathcal{T}+1)}{2}$ ciphertexts are for matching under encrypted domain, and the other two ciphertexts are for verification. Thus, this total requires $2\mathcal{T}(\mathcal{T} + 1)(\mathcal{M}_R + (r_5 + r_6 + r_7 + r_8)\mathcal{E}_M)$ mathematical operations for N sequences, where r_5 to r_8 are the random values selected by the data owner. In the **Query** phase, For EFSS_I and EFSS_II, the total mathematical operations are $N(k\mathcal{P} + k\mathcal{D} + \mathcal{M})$ for executing similarity query between $\mathbf{Q}$ and N DNA sequences.

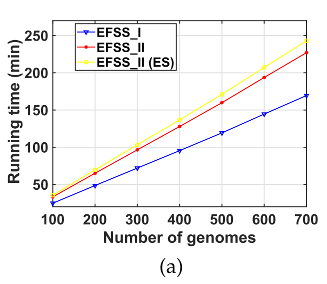


Fig. 9. Time for generating all encrypted indexes. (a) For the different number of genomes with the same number of cycle times, k = 10. (b) For the different number of cycle times with the same number of genomes, $N = 100$.

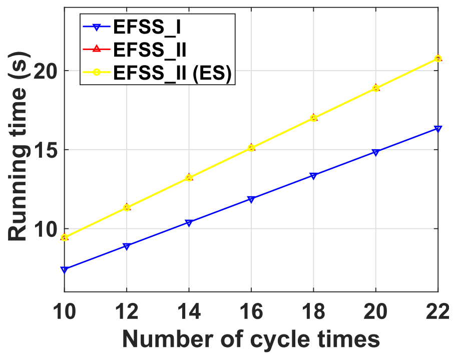


Fig. 10. Computation overhead for building a trapdoor.

However, for EFSS_II(ES), given a threshold **t** (selected by the search user), the cloud server first finds those cluster centers satisfying $Diff|(\kappa_i', \mathbf{Q}')| \leq \mathbf{t}$, where $\kappa_i, i = (1, 2, \ldots K)$ is a cluster center, and κ_i' is the set of its editing operations. Then, only those encrypted indexes belonged to selected centers will be further computed in the following querying. Therefore, the average of total mathematical operations is $\frac{2N}{\mathcal{K}}(k\mathcal{P} + k\mathcal{D} + \mathcal{M})$. For work [12], each query sequence is divided into $\frac{\mathcal{T}(\mathcal{T}+1)}{2}$ subsequences. The server needs 3 bilinear map operations under a group T to generate the ciphertext of each subsequence, which requires a total of $\frac{3\mathcal{T}(\mathcal{T}+1)}{2}\mathcal{B}_T$ bilinear map operations. Both [8] and [9] resort to garbled circuits GC to support string matching between DNA sequences under ciphertext. For work [8], the data owner first requires $N\tau\mathcal{HT}$ hash operations to generate encrypted ciphertexts for N sequence. In the query process, for each pair of sequences, the server and user need to interact once to obtain the edit distance between them. To return $\mathcal{K}$ closest results, the total of mathematical operations is $N\tau(\mathcal{O}_{\mathtt{GC}}) + N\tau\mathcal{D} + N\mathcal{M}$. Work [9] also divides each DNA sequence into $\mathcal{F}$ subsequences before performing matching in the ciphertext and needs garbled circuits GC to perform string matching for each pair of sequences. Thus, the total of mathematical operations is $N\mathcal{F}\mathcal{O}_{\mathtt{GC}}$. Please note that work [9] has other types of operations, such as blocking the DNA sequence used for the query, and comparing operations between strings to find the top $\mathcal{K}$ results. However, these operations can be considered negligible relative to performing garbled circuits.

6.4.2 *Experimental Results*

We also conduct experiments to demonstrate the computation overhead of our EFSS. From the above analysis, we can infer that the computation complexity of EFSS (i.e., EFSS_I, EFSS_II, EFSS_II(ES)) is mainly related to the number of genomes N in the dataset and the number of cycles k in the execution process. Without loss of generality, the dataset used in experiments is also selected from batches in chromosome 21, which contains 700 independent genomes. Each genome has a length of 47347896 and contains 357 different gene sequences. As before, we also divide the scheme into three parts (i.e.,**Index generation**, **Trapdoor generation** and **Query**) and discuss the computation overhead of each part separately.

Index generation. The computation cost in the phase of **Index generation** is shown in Fig. 9. We can observe that the cost of EFSS (i.e., EFSS_I, EFSS_II and EFSS_II(ES)) increases linearly with the number of genomes/cycles in the system. Compared to EFSS_I, since EFSS_II increases the dimensions of each encrypted index from s to $(2s + \omega + \psi + 2)$ (the number of users' role ω and dummy noises ψ are set as 20 and 10 in our experiments), it inevitably incurs additional communication overhead. Similarly, different from EFSS_II, EFSS_II (ES) requires the data owner to divide all DNA sequences into K classes, and generates additional encrypted indexes for the K cluster centers. Thus, with the increase in the number of genomes/cycle times, the running time of EFSS_II(ES) is slightly higher than the other two schemes.

Trapdoor generation. Given a query request, i.e., a genome gene **Q**, we can see that the cost of generating the trapdoor of **Q** is mainly related to the number of k (shown in Table 8). Fig. 10 shows the running time of generating a single trapdoor as the number of cycle times k increases. Similar to the previous, since EFSS_II increases the dimensions of the trapdoor from s to $(2s + \omega + \psi + 2)$, it requires more mathematical operations compared with EFSS_I generated. Also, since the trapdoor generated by EFSS_II and EFSS_II (ES) are consistent, the cost generated by these two schemes is the same.

Query. Figs. 11a and 11b show that the running time of EFSS with different number of genomes/cycles. We already know that the computation complexity of EFSS_I and EFSS_II is the same (i.e., $N(k\mathcal{P} + k\mathcal{D} + \mathcal{M})$) from Table 8. However, since EFSS_II increases both the dimensions of the trapdoor and encrypted index from s to $(2s + \omega + \psi + 2)$, it requires to perform more inner product operations between the vector and the matrix, thereby obtaining the closest matching DNA sequences. On the other hand, we can observe that the running time of EFSS_II(ES) is far less than both EFSS_I and EFSS_II. One of the major reasons is the K-means clustering algorithm utilized in the query process. As described in Section 4.3, by calling the K-means clustering algorithm, the cloud server first computes $I_{\kappa_i}^j \cdot T_{\mathbf{Q}}^j$ to find those cluster centers satisfying $Diff|(\kappa_i', \mathbf{Q}')| \leq \mathbf{t}$, where $\kappa_i, i = (1, 2, \ldots K)$ is a cluster center, and κ_i' is the set of its editing operations. Then, only those encrypted indexes belonged to selected centers will be further computed in our following querying, which can enormously decrease the matched number among indexes and trapdoors.

Besides, Fig. 11c shows that EFSS_II(ES) still maintains a significant advantage in computation cost even comparing with the works [8], [9], [12] (called GENSETS, PPPM and PPSP, respectively). This is mainly due to the following reasons: For work [12], each query sequence is divided into

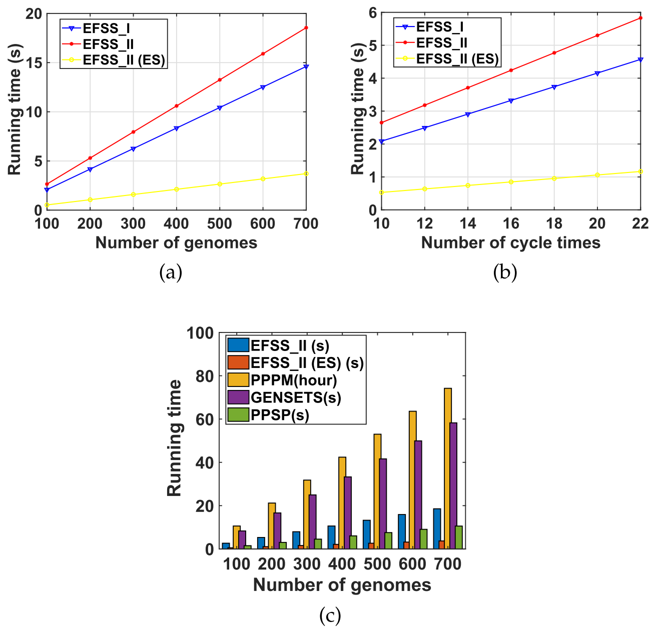


Fig. 11. Time for query. (a) For the different number of genomes with the same number of cycle times, $k = 10$. (b) For the different number of cycle times with the same number of genomes, $N = 100$. (c) For the different number of users with the same number of cycle times $k = 10$.

$\frac{\mathcal{T}(\mathcal{T}+1)}{2}$ subsequences. The server needs 3 bilinear map operations under a group T to generate the ciphertext of each subsequence, which will generate huge computation overhead (a total of $\frac{3\mathcal{T}(\mathcal{T}+1)}{2}\mathcal{B}_T$ bilinear map operations) for a long genomic sequence. Both [8] and [9] resort to garbled circuits `GC` [28] to support string matching between DNA sequences under ciphertext. However, given a query sequence $\mathbf{Q}$, the `GC` protocol will be called $N\tau$ and $N\mathcal{F}$ times in [8] and [9], respectively. We know that the `GC` is always large even for building a simple function. This is very large and requires huge computation cost to perform string matching between $\mathbf{Q}$ and N DNA sequences. Different from [8], [12] and [9], our EFSS design a private approximation algorithm to convert the edit distance computation problem between two sequences to the symmetric set difference size approximation problem, which can significantly reduce the number of elements that need to be matched under ciphertext. Based on this, we update EFSS_II to EFSS_II (ES) by utilizing K-means clustering algorithm to further improve the efficiency. Therefore, compared with works [8], [12] and [9], our EFSS shows superior performance in computation overhead.

7 Related Works

DNA similarity search over encrypted genetic data is usually deemed to have two different ways. The first model assumes each party only has its data and every party works together to complete a specific query task. Secure multi-party computation is always applied to this scenario. The second way is the secure outsourcing of computation model. In this model, all of the genetic data will be encrypted and outsourced to a semi-trusted party (such as cloud), and the query tasks are mainly executed by the cloud server privately. Specifically, Atallah *et al.* [48] proposed a privacy-preserving protocol to compute the edit distances between two DNA sequences. Later, to alleviate the computational overhead of [48], Jha *et al.*[11] also designed a secure query model to improve the computation efficiency. However, due to the cumbersome interaction between multiple participants, both of them incur huge communication and computing overhead. Wang *et al.* [49] designed a distributed framework to achieve privately genetic computing according to the different sensitivity levels. However, with the latest research results, the conception of what genes are insensitive is still an uncertain problem. Recently, Wang *et al.* [8] proposed a similar patient query system called GENSETS, which can achieve unprecedentedly high efficiency and precision of DNA similarity query. However, multiple interactions between users may significantly restrict the system's applied scope. Also, user's access rights and fine-grained queries such as mixed "AND" and "NO" operations on genes were not taken into account in their work. Gilad Asharov *et al.* [9] also designed an efficient approximation algorithm to compute the edit distances between sequences even with high divergence among individual genomes. Unfortunately, similar to [8], it was also only from the perspective of efficiency to devise the whole scheme. Lu *et al.* [50] proposed a secure outsourcing scheme called GWAS, which aimed to discover the relationships between gene mutations and some diseases by utilizing statistical information of genes. Chase *et al.* [51] proposed a DNA string matching algorithm exploiting symmetric searchable encryption. However, due to the defect of the deterministic encryption algorithm, the search pattern in their protocol was revealed to the cloud server. Besides, the function of the fuzzy DNA sequence query was not supported in their proposed model. Wang *et al.* [12] proposed a DNA sequence match scheme over encrypted cloud data, which only required one round of communication compared with previous protocols. However, their model sacrificed huge communication costs because of miscellaneous ciphertext processing based on the bilinear group. In general, most existing models are only devised from the perspective of efficiency. Moreover, other common and significant performance indicators, especially data access control and personalized search requirements (i.e., complex logic queries) are rarely taken into account. In this paper, we design an Efficient and Fine-grained Similarity Search (EFSS) scheme which enables retrieving resemble DNA sequences over encrypted cloud data. Specifically, we first propose an approximation algorithm to compute the edit distances between two sequences privately. Then, we present an efficient data access control strategy to manage the permissions of different users. In addition, by executing the custom query requests, our EFSS is supportive of mixed Boolean queries.

8 Conclusion

In this paper, we have proposed an efficient DNA similarity search scheme (EFSS) which enables fine-grained query and data access control over encrypted cloud data. In order to illustrate our purpose in a gradual way, we have proposed two secure DNA similarity model (i.e., EFSS_I and EFSS_II). We have given a formal and comprehensive analysis for error rate, data privacy and performance evaluation. For the future work, we intend to improve the accuracy and efficiency of our proposed schemes utilizing index formed tree structure. Besides, large-scale data updating such as data deletion, data insertion will be also considered in our future research.

ACKNOWLEDGMENTS

This work was supported by the National Key R&D Program of China under Grants 2017YFB0802300 and 2017YFB0802000, the National Natural Science Foundation of China under Grants 61972454, 61802051, 61772121, 61728102, and 61472065, the Peng Cheng Laboratory Project of Guangdong Province PCL2018KP004, the Guangxi Key Laboratory of Cryptography and Information Security under Grant GCIS201804.

REFERENCES

[1] H. I. E. Ozercan, "Realizing the potential of blockchain technologies in genomics," *Genome Res.*, vol. 28, no. 9, pp. 1255–1263, 2018.

[2] M. Tao, K. Ota, M. Dong, and Z. Qian, "AccessAuth: Capacity-aware security access authentication in federated-IoT-enabled V2G networks," *J. Parallel Distrib. Comput.*, vol. 118, pp. 107–117, 2018.

[3] W. Sun, N. Zhang, W. Lou, and T. Hou, "When gene meets cloud: Enabling scalable and efficient range query on encrypted genomic data," in *Proc. IEEE Conf. Comput. Commun.*, 2017, pp. 37–46.

[4] O. Choudhury *et al.*, "Enforcing human subject regulations using blockchain and smart contracts," *Blockchain Healthcare Today*, 2018. [Online]. Available: https://doi.org/10.30953/bhty.v1.10

[5] J. Wu, M. Dong, K. Ota, J. Li, and Z. Guan, "Big data analysis-based secure cluster management for optimized control plane in software-defined networks," *IEEE Trans. Netw. Service Manage.*, vol. 15, no. 1, pp. 27–38, Mar. 2018.

[6] G. Xu, H. Li, Y. Dai, K. Yang, and X. Lin, "Enabling efficient and geometric range query with access control over encrypted spatial data," *IEEE Trans. Inf. Forensics Security*, vol. 14, no. 4, pp. 870–885, Apr. 2019.

[7] G. Xu, H. Li, S. Liu, K. Yang, and X. Lin, "VerifyNet: Secure and verifiable federated learning," *IEEE Trans. Inf. Forensics Security*, vol. 15, pp. 911–926, 2020.

[8] X. Wang, Y. Huang, Y. Zhao, H. Tang, X. Wang, and D. Bu, "Efficient genome-wide, privacy-preserving similar patient query based on private edit distance," in *Proc. 22nd ACM SIGSAC Conf. Comput. Commun. Secur.*, 2015, pp. 492–503.

[9] G. Asharov, S. Halevi, Y. Lindell, and T. Rabin, "Privacy-preserving search of similar patients in genomic data," *Proc. Privacy Enhancing Technol.*, vol. 2018, no. 4, pp. 104–124, 2018.

[10] K. Cheng, Y. Hou, and L. Wang, "Secure similar sequence query on outsourced genomic data," in *Proc. Asia Conf. Comput. Commun. Secur.*, 2018, pp. 237–251.

[11] M. S. R. Mahdi, M. Z. Hasan, and N. Mohammed, "Secure sequence similarity search on encrypted genomic data," in *Proc. IEEE/ACM Int. Conf. Connected Health: Appl. Syst. Eng. Technol.*, 2017, pp. 205–213.

[12] B. Wang, W. Song, W. Lou, and Y. T. Hou, "Privacy-preserving pattern matching over encrypted genetic data in cloud computing," in *Proc. IEEE Conf. Comput. Commun.*, 2017, pp. 1–9.

[13] Y. Miao, R. Deng, X. Liu, K.-K. R. Choo, H. Wu, and H. Li, "Multi-authority attribute-based keyword search over encrypted cloud data," *IEEE Trans. Dependable Secure Comput.*, to be published, doi: 10.1109/TDSC.2019.2935044.

[14] G. Xu, H. Li, H. Ren, K. Yang, and R. H. Deng, "Data security issues in deep learning: Attacks, countermeasures and opportunities," *IEEE Commun. Mag.*, vol. 57, no. 11, pp. 116–122, Nov. 2019.

[15] G. Xu, H. Li, C. Tan, D. Liu, Y. Dai, and K. Yang, "Achieving efficient and privacy-preserving truth discovery in crowd sensing systems," *Comput. Secur.*, vol. 69, pp. 114–126, 2017.

[16] H. Li, Y. Yang, T. H. Luan, X. Liang, L. Zhou, and X. Shen, "Enabling fine-grained multi-keyword search supporting classified sub-dictionaries over encrypted cloud data," *IEEE Trans. Dependable Secure Comput.*, vol. 13, no. 3, pp. 312–325, May/Jun. 2016.

[17] K. He, J. Guo, J. Weng, J. Weng, J. K. Liu, and X. Yi, "Attribute-based hybrid boolean keyword search over outsourced encrypted data," *IEEE Trans. Dependable Secure Comput.*, to be published, doi: 10.1109/TDSC.2018.2864186.

[18] S. Jiang, X. Zhu, L. Guo, and J. Liu, "Publicly verifiable boolean query over outsourced encrypted data," *IEEE Trans. Cloud Comput.*, vol. 7, no. 3, pp. 799–813, Third Quarter 2019.

[19] H. Li, G. Xu, Q. Tang, X. Lin, and X. Shen, "Enabling efficient and fine-grained dna similarity search with access control over encrypted cloud data," in *Proc. Int. Conf. Wireless Algorithms Syst. Appl.*, 2018, pp. 236–248.

[20] M. Hao, H. Li, X. Luo, G. Xu, H. Yang, and S. Liu, "Efficient and privacy-enhanced federated learning for industrial artificial intelligence," *IEEE Trans. Ind. Informat.*, to be published, doi: 10.1109/TII.2019.2945367.

[21] H. Ren, H. Li, Y. Dai, K. Yang, and X. Lin, "Querying in Internet of Things with privacy preserving: Challenges, solutions and opportunities," *IEEE Netw.*, vol. 32, no. 6, pp. 144–151, Nov./Dec. 2018.

[22] Z. Fu, X. Wu, C. Guan, X. Sun, and K. Ren, "Toward efficient multi-keyword fuzzy search over encrypted outsourced data with accuracy improvement," *IEEE Trans. Inf. Forensics Security*, vol. 11, no. 12, pp. 2706–2716, Dec. 2016.

[23] J. Chen *et al.*, "ELIMFS: achieving efficient, leakage-resilient, and multi-keyword fuzzy search on encrypted cloud data," *IEEE Trans. Services Comput.*, to be published, doi: 10.1109/TSC.2017.2765323.

[24] G. Xu, H. Li, S. Liu, M. Wen, and R. Lu, "Efficient and privacy-preserving truth discovery in mobile crowd sensing systems," *IEEE Trans. Veh. Technol.*, vol. 68, no. 4, pp. 3854–3865, Apr. 2019.

[25] V. Kolesnikov and T. Schneider, "Improved garbled circuit: Free XOR gates and applications," in *Proc. Int. Colloquium Automata Lang. Program.*, 2008, pp. 486–498.

[26] Y. Ejgenberg, M. Farbstein, M. Levy, and Y. Lindell, "SCAPI: The secure computation application programming interface," *IACR Cryptol. EPrint Archive*, vol. 2012, 2012, Art. no. 629.

[27] G. Asharov, Y. Lindell, T. Schneider, and M. Zohner, "More efficient oblivious transfer and extensions for faster secure computation," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, 2013, pp. 535–548.

[28] P. Mohassel, M. Rosulek, and Y. Zhang, "Fast and secure three-party computation: The garbled circuit approach," in *Proc. 22nd ACM SIGSAC Conf. Comput. Commun. Secur.*, 2015, pp. 591–602.

[29] Z. Zhang *et al.*, "Achieving privacy-friendly storage and secure statistics for smart meter data on outsourced clouds," *IEEE Trans. Cloud Comput.*, vol. 7, no. 3, pp. 638–649, Third Quarter 2019.

[30] J. Li, X. Lin, Y. Zhang, and J. Han, "KSF-OABE: Outsourced attribute-based encryption with keyword search function for cloud storage," *IEEE Trans. Services Comput.*, vol. 10, no. 5, pp. 715–725, Sep./Oct. 2017.

[31] P. Xu, S. Tang, P. Xu, Q. Wu, H. Hu, and W. Susilo, "Practical multi-keyword and boolean search over encrypted E-mail in cloud server," *IEEE Trans. Services Comput.*, to be published, doi: 10.1109/TSC.2019.2903502.

[32] M. Kuzu, M. S. Islam, and M. Kantarcioglu, "Efficient similarity search over encrypted data," in *Proc. IEEE 28th Int. Conf. Data Eng.*, 2012, pp. 1156–1167.

[33] M. Strizhov and I. Ray, "Multi-keyword similarity search over encrypted cloud data," in *Proc. IFIP Int. Inf. Secur. Conf.*, 2014, pp. 52–65.

[34] IGSR and the 1000 genomes project, "International genome sample resource," 2018. Accessed: Mar. 2018. [Online]. Available: http://www.internationalgenome.org/

[35] W. J. Masek and M. S. Paterson, "A faster algorithm computing string edit distances," *J. Comput. Syst. Sci.*, vol. 20, no. 1, pp. 18–31, 1980.

[36] W. K. Wong, D. W.-L. Cheung, B. Kao, and N. Mamoulis, "Secure kNN computation on encrypted databases," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2009, pp. 139–152.

[37] H. Li, D. Liu, Y. Dai, and T. H. Luan, "Engineering searchable encryption of mobile cloud networks: When QoE meets QoP," *IEEE Wireless Commun.*, vol. 22, no. 4, pp. 74–80, Aug. 2015.

[38] H. Li, Y. Yang, Y. Dai, and Y. Xiang, "Achieving secure and efficient dynamic searchable symmetric encryption over medical cloud data," *IEEE Trans. Cloud Comput.*, to be published, doi: 10.1109/TCC.2017.2769645.

[39] H. Li, D. Liu, Y. Dai, T. H. Luan, and X. Shen, "Enabling efficient multi-keyword ranked search over encrypted mobile cloud data through blind storage," *IEEE Trans. Emerg. Topics Comput.*, vol. 3, no. 1, pp. 127–138, Mar. 2015.

[40] F. Yamaguchi and H. Nishi, "Hardware-based hash functions for network applications," in *Proc. 19th IEEE Int. Conf. Netw.*, 2013, pp. 1–6.

[41] M. Ahmad, P. M. Khan, and M. Z. Ansari, "A simple and efficient key-dependent S-box design using fisher-yates shuffle technique," in *Proc. Int. Conf. Secur. Comput. Netw. Distrib. Syst.*, 2014, pp. 540–550.

[42] L. Kuang, L. T. Yang, J. Feng, and M. Dong, "Secure tensor decomposition using fully homomorphic encryption scheme," *IEEE Trans. Cloud Comput.*, vol. 6, no. 3, pp. 868–878, Third Quarter 2018.

[43] J. Katz, A. Sahai, and B. Waters, "Predicate encryption supporting disjunctions, polynomial equations, and inner products," in *Proc. Annu. Int. Conf. Theory Appl. Cryptographic Techn.*, 2008, pp. 146–162.

[44] A. De Caro and V. Iovino, "jPBC: Java pairing based cryptography," in *Proc. IEEE Symp. Comput. Commun.*, 2011, pp. 850–855.

[45] "JPype, Java to python integration," 2013. [Online]. Available: http://jpype.sourceforge.net/

[46] S. Zahur, M. Rosulek, and D. Evans, "Two halves make a whole," in *Proc. Annu. Int. Conf. Theory Appl. Cryptographic Techn.*, 2015, pp. 220–250.

[47] Y. Ishai, J. Kilian, K. Nissim, and E. Petrank, "Extending oblivious transfers efficiently," in *Proc. Annu. Int. Cryptology Conf.*, 2003, pp. 145–161.

[48] M. J. Atallah and J. Li, "Secure outsourcing of sequence comparisons," *Int. J. Inf. Secur.*, vol. 4, no. 4, pp. 277–287, 2005.

[49] R. Wang, X. F. Wang, Z. Li, H. Tang, M. K. Reiter, and Z. Dong, "Privacy-preserving genomic computation through program specialization," in *Proc. 16th ACM Conf. Comput. Commun. Secur.*, 2009, pp. 338–347.

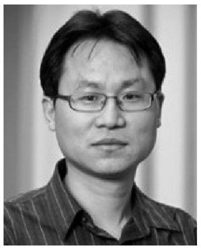
[50] W. Lu, Y. Yamada, and J. Sakuma, "Efficient secure outsourcing of genome-wide association studies," in *Proc. IEEE Security Privacy Workshops*, 2015, pp. 3–6.

[51] M. Chase and E. Shen, "Substring-searchable symmetric encryption," *Proc. Privacy Enhancing Technol.*, vol. 2015, no. 2, pp. 263–281, 2015.

Guowen Xu (Student Member, IEEE) received the BS degree in information and computing science from the Anhui University of Architecture (AUA), Hefei, China, in 2014. Currently, he is working toward the PhD degree in the School of Computer Science and Engineering, University of Electronic Science and Technology of China (UESTC), China. His research interests include cryptography, searchable encryption, and the privacy-preserving deep learning.

Hongwei Li (Senior Member, IEEE) is currently the head and a professor with the Department of Information Security, School of Computer Science and Engineering, University of Electronic Science and Technology of China, China. His research interests include network security and applied cryptography. He serves as an associate editor of the *IEEE Internet of Things Journal*, and the *Peer-to-Peer Networking and Applications*, the guest editor of the *IEEE Network* and the *IEEE Internet of Things Journal*. He is a distinguished lecturer of the IEEE Vehicular Technology Society.

Hao Ren (Student Member, IEEE) received the BS degree from the School of Information Science and Technology, Chengdu University of Technology (CDUT), Chengdu, China, in 2014. Currently, he is working toward the PhD degree in the School of Computer Science and Engineering, University of Electronic Science and Technology of China (UESTC), China. His research interests include cryptography, searchable encryption, and security and privacy of outsourcing data.



Xiaodong Lin (Fellow, IEEE) is currently an associate professor with the School of Computer Science, University of Guelph, Canada. His research interests include wireless network security, applied cryptography, computer forensics, software security, and wireless networking and mobile computing. He was a recipient of the Outstanding Achievement in Graduate Studies Award from the University of Waterloo, Waterloo, Canada.

Xuemin (Sherman) Shen (Fellow, IEEE) received the PhD degree from Rutgers University, Camden, New Jersey, in 1990. He is a professor with the Department of Electrical and Computer Engineering, University of Waterloo, Canada. His research include resource management in interconnected wireless/wired networks, wireless network security, social networks, smart grid, and vehicular ad hoc and sensor networks. He serves as the editor-in-chief for the *IEEE Internet of Things Journal*, the *Peer-to-Peer Networking and Application*, and the *IET Communications*; a founding area editor for the *IEEE Transactions on Wireless Communications*. He is a registered professional engineer of Ontario, Canada, an engineering institute of Canada fellow, a Canadian academy of engineering fellow, a Royal Society of Canada fellow, and a distinguished lecturer of the IEEE Vehicular Technology Society and Communications Society.