PriRanGe: Privacy-Preserving Range-Constrained Intersection Query Over Genomic Data

Yaxi Yang, Yao Tong, Jian Weng, *Member, IEEE*, Yufeng Yi, Yandong Zheng, Leo Yu Zhang, *Member, IEEE* and Rongxing Lu, *Fellow, IEEE*

Abstract—Genomic data is being produced rapidly by both individuals and enterprises, and outsourcing this ever-increasing data into clouds is promising for cutting the cost of data owners and mining the wealth of genomic data at a larger scale. However, genome carries sensitive information about individuals, and it is challenging to securely and efficiently perform analysis on remotely hosted genomic databases. In this paper, we present a privacy-preserving range-constrained intersection query scheme on genomic data. To achieve security and efficiency, we propose a protocol to fulfill range-constrained intersection query, named PriRanGe. With PriRanGe, a client can securely query genomic data in a specific range in a database while keeping this whole process private. The security of our design targets genomic database confidentiality, query range/result confidentiality, and access pattern protection, and the advantage in efficiency is due to most employed primitives are symmetric. We thoroughly evaluated our design by security proof, experimental analysis and comparison to the state-of-the-art works, all of which support the conclusion that this design is both secure and fast.

Index Terms—Genomic Data, Range-constrained Intersection Query, Garbled Circuits, Secure Multiparty Computation.

1 Introduction

Technical improvements in DNA sequencing [1] have paved the way for genomic testing: the costs for sequencing a whole genome have fallen from 10 million USD to $< 1,000$ USD in the last ten years, and individuals can easily afford to test their original genomic data in a sequencing lab [2]. Powered with big genomic data, genomic testing services are becoming increasingly popular, e.g., MyHeritage[1], ancestry[2], and PatientsLikeMe[3]. Among all these tests and alike services, one underlying technique is to find the intersection between a piece of genomic query and a genomic database (e.g., CNGB, NCBI) within a specific range [3]. An example is shown in Fig. 1, after a client gets his/her genomic data from the sequence lab, he/she may want to request a genomic testing service and launch a range query to a database provider. Then, the database provider searches for the related variants within the queried range that is defined by a lower and upper position in the entire genome, and then responds the results to the client [4, 5].

- Y. Yang, J. Weng, Y. Yi are with the College of Information Science and Technology, Jinan University, Guangzhou, China. (email: E-mail: see yxyangjnu@gmail.com, cryptjweng@gmail.com, yyfeng5834@gmail.com)
- Y. Tong is with Guangzhou Fongwell Data Limited Company, Guangzhou 510632, China. (email: melody@fongwell.com)
- L. Zhang is with the School of Information Technology, Deakin University, VIC, Australia. (email: leo.zhang@deakin.edu.au)
- Y. Zheng and R. Lu are with the Faculty of Computer Science, University of New Brunswick, Fredericton, NB E3B 5A3, Canada (e-mail: yzheng8@unb.ca, rlu1@unb.ca).

1. https://myheritage.com
2. https://www.ancestry.com
3. https://www.patientslikeme.com

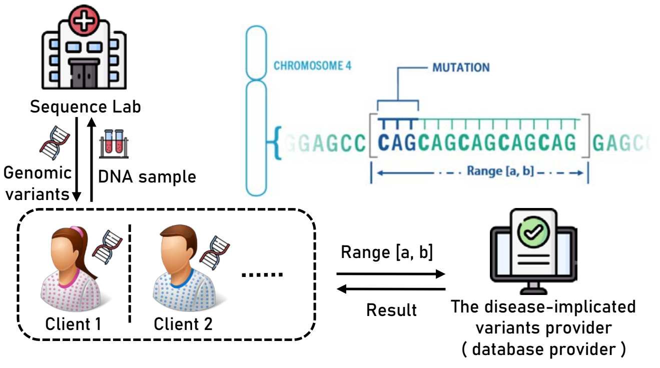


Fig. 1: The structure of genomic range query service.

However, it is well-accepted that genomic data is highly sensitive and can be seen as an identifier of an individual [6]. On the one hand, a client, who intends to retrieve genomic sequences of interest within the queried range, needs to be restricted. Otherwise, the client can identify an individual in the database by utilizing genomic domain knowledge, such as the relative frequency of variants in the entire genome, even though the database is made anonymous in the first place [7, 8]. Therefore, when receiving queries about some genomic variants of the database from a client, the provider needs to make this query private and constrain the returned matches to the exact range. On the other hand, the query result and the query range of the client also need to be private since the specific range of an individual's entire genome leaks information about the query sequence. If the range or query results are learned by the database provider, the provider can infer sensitive information (e.g., potential disease) about the client [4]. For example, it is reported in [4] that the queried range can actually imply the underlying test. Therefore, when receiving a client's query about some genomic variants of a database, the provider needs to ensure the returned result only matches the exact range without knowing the query and the query range. This dual privacy requirement on both the client query and the database makes it difficult to use traditional range query methods. For example, if we trivially apply the traditional range query methods in [9–11] to the case of genomic data, it will result in leakage of the query range and access pattern. To solve those privacy problems, some researchers utilize the techniques of secure multi-party computation (MPC) to build secure genomic query protocols [12–14]. However, those methods bring expensive client-side computation and communication burdens.

To relieve client-side burdens, outsourcing computation and storage to the cloud is now an effective method and has been widely studied [19, 20]. In the outsourced setting, the database provider securely outsources data to a (or more) cloud server(s). When a client issues a range query to the provider, the provider can delegate the cloud to process the client's query and search in the encrypted database in a privacy-preserving manner. Several privacy-preserving protocols [15, 16] are proposed to perform multiple types of queries on genomic databases in the outsourcing setting. However, none of these protocols consider the privacy of access pattern, i.e., the information on how the outsourced database is accessed is leaked. If the access pattern is exposed to the cloud server, the server can infer sensitive genomic information about the client's query and **db**, as reported in [21, 22]. Besides, there exist some works about range query in the outsourced setting [11, 23], but their schemes aim differently from ours. In particular, their designs aim to determine whether a given condition is met within a range of a database (i.e., general sense range query), while our design firstly determines a query range and then finds all the intersection of the query array and the defined range within **db** (i.e., genome-specific range query).

Motivated by the above arguments, in this paper, we propose an efficient privacy-preserving range-constrained intersection query scheme over genomic data (PriRanGe) in the outsourcing setting. Besides the confidentiality of the query and the database, we also aim to protect the access pattern of the range-constrained query over genomic data without sacrificing efficiency. The main challenge comes from the conflicting goals between the efficiency requirement and strong security requirements of each party (especially the access pattern protection).

In the literature of privacy-preserving applications (no genomic data-related), there are some customized works for computing private intersection in the outsourced setting [24, 25]. However, the proposed scheme in [24] leaks the size of the intersection to the cloud server, and [25] focuses on computing the cardinality of the intersection and the database owner needs to participate in the online computation. Hence, both designs do not protect the access pattern. There are also some general techniques, including Private Information Retrieval (PIR) [17, 26] and Oblivious RAM (ORAM) [27], that can be used to protect the access pattern privacy in the outsourced setting. But PIR [17, 26] relies on time-consuming homomorphic encryptions, and ORAM [27] requires the continuous shuffling of data chunks for every single access, while the number of elements in a database could be large. Clearly, trivially applying these methods to the range query of genomic data will only result in solutions with inferior security or efficiency.

As a workaround, we design the PriRanGe protocol by combining some basic MPC primitives, including secret sharing, garbled circuits, oblivious transfer (OT) extension [28], and oblivious distributed key pseudorandom function (Odk-PRF) [25]. The obvious advantage of these primitive collections is that most of them rely on symmetric encryption (with OT extension using a few asymmetric encryptions), which enables PriRanGe with the potential for high efficiency. In PriRanGe, we put forward two subprotocols, the Distributed Intersection Computation (DIC) protocol and the Secret-Shared Oblivious Shuffling (SOSF) protocol, to enable two non-colluding cloud servers to fulfill the range-constraint intersection query without knowing how the outsourced database is accessed and what is being accessed.

To sum up, the contributions of this work are as follows:

- We construct the PriRanGe protocol by using secret sharing, garbled circuits and Odk-PRF. We utilize this protocol to realize privacy-preserving range-constrained intersection query over genomic data.
- We design the DIC and SOSF protocols as building blocks of PriRanGe to enable two non-colluding cloud servers to fulfill the range-constraint intersection query from clients without leaking information about the access pattern. These protocols are of independent research interests in their own rights.
- We formally prove that our protocols are secure in the semi-honest adversary model. We experimentally evaluate PriRanGe on real human genomic data, which validates its efficiency.

The rest of this work is organized as follows. The related works are introduced in Sec. 2, and the systematic problem definition is presented in Sec. 3. Sec. 4 introduces the preliminary knowledge used in our design, followed by the building blocks and the details of PriRanGe in Sec. 5. The security analyses and experimental results are respectively given in Secs. 6 and 7. The concluding remarks are drawn in Sec. 9.

2 Related Works

In this section, we briefly review some recently proposed schemes about the secure genomic query based on a outsourced model, range query in outsourced setting, and oblivious shuffling methods, which is used to protect access pattern.

Secure Genomic Query in Outsourced Database. In the outsourced two-cloud setting, genomic databases are outsourced to two cloud servers and the main computation tasks are performed between those servers. In [29], Atallah et al. proposes a protocol that securely computes the edit distance of two sequences in two remote servers. To achieve better performance, there were some other works [15–18] under the model of two non-colluding semi-honest servers. In [15, 16], the encrypted databases were outsourced to one

TABLE 1: Comparison of different genomic query protocols.

Properties	Al et al. [15]	Sun et al. [4]	Nassar et al. [16]	Cheng et al. [17]	Schneider et al. [18]	PriRanGe
Outsourced setting	✓	✓	✓	✓	✓	✓
Queried range privacy	✗	✓	✓	✓	✓	✓
Queried result privacy	✓	✓	✗	✓	✓	✓
Hide access pattern	✗	✗	✗	✓	✗	✓
Query method	OPE + HE	MSSE	HE	GC + HE	OT + SS	GC + OT
Query type	Rank query	Range query	Similarity query	Similarity query	Similarity query	Range query

Notes. OPE corresponds to order preserving encryption. MSSE corresponds to multi-keyword symmetric searchable encryption. HE is homomorphic encryption. GC represents garbled circuit. SS represents secret sharing.

cloud server and the encrypted keys are stored in another server. Then a client would launch a genomic query with those two servers. However, the schemes in [15, 16] need the clients to stay online and communicate with each servers, which results in high communication cost on the client-side. To solve this problem, Cheng et al. [17] and Schneider et al. [18] adopted the same model that outsourced all the databases and computation to two cloud servers. Next, the two cloud servers performed protocols on secret shared data and returned the query results to the clients.

Range Query in Outsourced Setting. In the literature, various privacy-preserving range query schemes are proposed [9–11, 23, 30–33] for range query in a outsourced setting. Li et al. [9] designed a PB-tree structure and associated algorithms to support searching and updating a database in the cloud. Zuo et al. [10] propose a range query protocol via a binary tree, which was a symmetric-key-based method for a private database. Besides, Liang et al. [11] presented a privacy-preserving range query in a public cloud. They utilized a tree-based structure and homomorphic encryption to realize their goal. However, those methods do not consider the information leakage of the access pattern. Next, Wu et al. [23] propose an efficient multi-dimensional range query via a cube encoding method. For constrained range query, Li and Teng et al. [31, 34] propose constrained range query protocols for spatial queries within a bounded range or a geometric range. However, their methods did not focus on the security problems in constrained range query.

As for the genomic scenario, there exist some works about range query for genomic databases [2, 4, 5, 12, 35]. In [2, 12], the authors used additively homomorphic encryption to count the number of genetic markers matching in a certain range of digitized genomes. However, their methods are relatively inefficient and take hours to search in million-sized genomic variants. To improve the efficiency, Kolesnikov et al. [35] propose an efficient method based on the OT extension. This method could also be applied to genomic sub-string matching and extended to the case of genomic variants counts within a range if the proposed protocol is executed in a bitwise manner. In [5], the authors propose a verifiable range query scheme. When a client wanted to query a certain range in a database, the client needed to utilize a proposed signature scheme to prove that his/her queried sequence was genuine and not tampered with. However, those methods did not consider hiding the searching range of their queries. Sun et al. [4] propose a range query over space-consuming raw genomic data based on a multi-keyword symmetric searchable encryption scheme and a hierarchical index structure. But their system model involves a trusted authority for encrypting and decrypting query range and returned results, which reduces its practicality for real usage. To sum up, existing range query schemes still have some limitations in the terms of efficiency and security. For clarity, we summarize the differences between our work and previous secure schemes in Table 1.

Oblivious Shuffling. There are some works about obliviously sorting an entire array of elements in different application scenarios [17, 36–38]. In [17, 37], the authors employed homomorphic encryption to realize a shuffling method. Specifically, [17] suggests a secure shuffling protocol in a outsourced setting, which has the similar functionality of our SOSF protocol. However, these methods tend to be inefficient because of the burden introduced by homomorphic public-key encryption. The works in [36, 38] provide shuffling-and-compute methods based on the garbled circuits. To be specific, Huang et al. [36] designed a shuffling network to shuffle an intersection results of two sets to protect the privacy of input sets. Cheng et al. [38] used a permutation matrix to multiply a set before sorting it. Both the proposed shuffling methods can cut off the link between the original input set and the output set, which could protect the access pattern. Similarly, our SOSF protocol also aims to protect the access pattern through oblivious shuffling. But unlike previous techniques, it is customized for XOR-shared values in the delegate setting. Moreover, as will be shown in Sec. 7, our SOSF protocol outperforms Cheng's shuffling protocol [17].

3 System Overview

In this section, we first present our system model, and then give the definition of range-constrained intersection query, and outline the considered threats to the system and the goals we want to achieve.

3.1 System Model

As shown in Fig. 2, our system model consists of three main entities: the database providers $\mathcal{DBP}$, two cloud servers $\mathcal{CS}_1$, $\mathcal{CS}_2$, and a client $\mathcal{C}$. Their respective roles are defined as follows.

- $\mathcal{DBP}$: The database provider $\mathcal{DBP}$ is a genomic testing facility and possesses a large collection of genomic variants database **db**. $\mathcal{DBP}$ wants to delegate his/her database to the clouds for remote hosting and management, since $\mathcal{DBP}$ might be a medical center and is not expertise

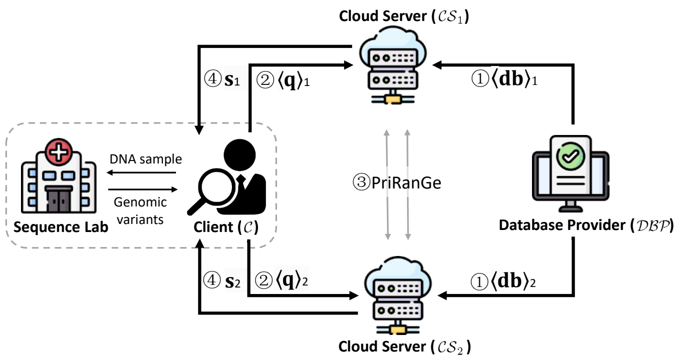

Cloud Server ($\mathcal{CS}_1$)
④$\mathbf{s}_1$
②$\langle \mathbf{q} \rangle_1$
①$\langle \mathbf{db} \rangle_1$
DNA sample
Genomic variants
Sequence Lab
Client ($\mathcal{C}$)
③PriRanGe
Database Provider ($\mathcal{DBP}$)
④$\mathbf{s}_2$
②$\langle \mathbf{q} \rangle_2$
①$\langle \mathbf{db} \rangle_2$
Cloud Server ($\mathcal{CS}_2$)

Fig. 2: Overview of our system model.

in data management and not powerful in storage and computing. However, the cloud servers cannot be fully trusted. $\mathcal{DBP}$ will split his/her data into two shares $\langle \mathbf{db} \rangle_1$, $\langle \mathbf{db} \rangle_2$ by secret sharing and upload the two shares to two cloud servers, respectively.

- $\mathcal{C}$: A client $\mathcal{C}$ can always use his/her DNA sample (e.g., blood or saliva) to get his genomic variants from a sequence lab. If $\mathcal{C}$ wants to search in $\mathcal{DBP}$ to check whether he/she has some potential diseases, $\mathcal{C}$ will prepare his/her query array $\mathbf{q}$ and specify a range that possibly contains the query $\mathbf{q}$ and its variants. Then, $\mathcal{C}$ divides the query array $\mathbf{q}$ into two shares, $\langle \mathbf{q} \rangle_1$ and $\langle \mathbf{q} \rangle_2$, and sends them to $\mathcal{CS}_1$ and $\mathcal{CS}_2$, who will then privately retrieve query results.
- $\mathcal{CS}_1$, $\mathcal{CS}_2$: The two servers $\mathcal{CS}_1$ and $\mathcal{CS}_2$, who are deployed between the client $\mathcal{C}$ and the database provider $\mathcal{DBP}$, have abundant bandwidth, storage, and computing resources. Upon receiving the shares of query array $\mathbf{q}$ (from $\mathcal{C}$) and $\mathbf{db}$ (from $\mathcal{DBP}$), the two servers interactively perform the PriRanGe protocol while keeping $\mathbf{q}$ and $\mathbf{db}$ private. Finally, the servers return the partial query results $\mathbf{s}_1$ and $\mathbf{s}_2$ separately to $\mathcal{C}$, who then combines them together to obtain the complete query result.

3.2 Range-Constrained Intersection Query

Human DNA consists of four types of bases ([A]denine, [C]ytosine, [T]hymine, [G]uanine), and different types of mutations might occur if an individual has potential diseases. A genomic variant marks different types of mutations of nucleotide, such as [A] $\rightarrow$ [G] [5]. An individual's DNA includes lots of mutations, and each mutation includes a name of position (pos) in DNA and a base letter (v) at this position. Under our system model, if the client $\mathcal{C}$ wants to check whether he/she has some genetic diseases, $\mathcal{C}$ will launch a range-constrained intersection query between his/her query array $\mathbf{q}$ and $\mathbf{db}$. This problem is formalized as follows:

Definition 1. ***(Range-constrained intersection query)*** *Suppose that* $\boldsymbol{db} = \{db_1, \cdots, db_n\}$ *is a genomic variants database. Each variant* db_i *in* $\boldsymbol{db}$ *consists of a position* $db_i.pos$ *and a specific variants* $db_i.v$*. Then, give a query array* $\boldsymbol{q} = \{q_1, \cdots, q_m\}$ *within a range of* $[q_1.pos, q_m.pos]$*, the same range needs to be constrained in* $\boldsymbol{db}$*, denoted as* $\boldsymbol{db}_\alpha$*. Therefore, the range-constrained intersection query is to compute the intersection of* $\boldsymbol{q}.v$ *and* $\boldsymbol{db}_\alpha.v$*, where*

$$\begin{cases} \boldsymbol{db}_\alpha \subset \boldsymbol{db}, \\ \boldsymbol{db}_\alpha = \{db_{t_1}, \cdots, db_{t_m}\}, \\ [db_{t_1}.pos, db_{t_m}.pos] = [q_1.pos, q_m.pos]. \end{cases}$$

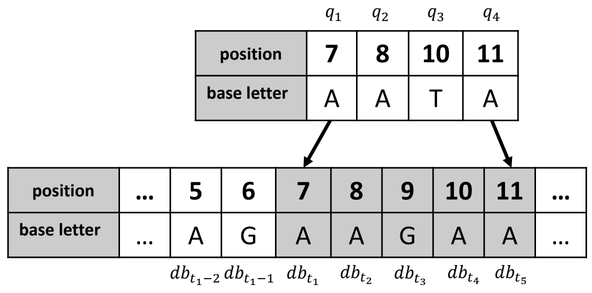

q_1 q_2 q_3 q_4
position 7 8 10 11
base letter A A T A
position … 5 6 7 8 9 10 11 …
base letter … A G A A G A A …
db_{t_1-2} db_{t_1-1} db_{t_1} db_{t_2} db_{t_3} db_{t_4} db_{t_5}

Fig. 3: An example of range-constrained intersection query.

For better understanding, we give an example of range-constrained intersection query in Fig. 3. From this figure, after constraining the exact range in $\mathbf{db}$, the final intersection result can be easily determined if no security is required.

3.3 Threat Model

In this work, the database provider and the client are considered to be honest and will strictly follow the protocol. However, we assume the two cloud servers are semi-honest, which means they will honestly follow the proposed protocol but they try to learn and deduce the information about $\mathbf{db}$, $\mathbf{q}$ and the query result [17, 18]. Similar to the literature studies that use two clouds [17, 18, 38, 39], $\mathcal{CS}_1$ and $\mathcal{CS}_2$ are assumed to not collude with each other because of interest conflict. Note that our work focuses on privacy preservation issues, some other attacks for this system are beyond the scope of this paper and will be discussed in our future work. In conclusion, the security requirements that our paper aims to achieve have three folds:

- *Database confidentiality:* The two servers should not learn any information of $\mathcal{DBP}$'s database $\mathbf{db}$. Besides, client $\mathcal{C}$ should only obtain the intersection results within his/her queried range and learn nothing about other elements of $\mathbf{db}$.
- *Query confidentiality:* After receiving the shares of the database and query range respectively from $\mathcal{DBP}$ and $\mathcal{C}$, the two servers perform the PriRanGe protocol and return queried results to $\mathcal{C}$. During the protocol execution process, servers should not learn anything about $\mathcal{C}$'s query and the results.
- *Access pattern protection:* If the two servers $\mathcal{CS}_1$ and $\mathcal{CS}_2$ have responded to multiple queries and returned all results to $\mathcal{C}$, servers may record the access patterns of all queries and link the patterns to launch sophisticated attacks, for example, inferring positions of genomic variants of $\mathbf{db}$ or revealing sensitive information about the queries [17, 38]. So, to protect access pattern, both servers should be made oblivious of real access patterns among multiple queries.

Even though we consider $\mathcal{C}$ in our model is honest, cloud servers can use authentication methods to restrict $\mathcal{C}$'s access and ensure $\mathcal{C}$ is registered, which makes our cloud-based query protocol resistant to more security threats. There are some password-based authentication methods [40–43] that can be straightforwardly applied in the PriRanGe protocol.

3.4 Design Goals

Generally speaking, our goal is to delegate the range-constrained intersection query scheme over genomic data to the cloud while preserving data privacy. Under the constraint of the system and threat models, this work aims to achieve:

- *Privacy preservation:* The basic requirement of our protocols is privacy preservation. We aim to not only preserve the privacy of the outsourced genomic database, query requests, as well as query results, but also resist the leakage of access pattern.
- *Efficiency:* In order to achieve the above security goals, it will inevitably introduce extra computations for performing intersection query on secret-shared genomic data. In the proposed scheme, we aim to cut down the computation cost to perform the intersection query while optimizing the execution time.

4 Preliminaries

In this section, we introduce the necessary background knowledge about genomic sequence query and some cryptographic building blocks that our work is based on.

4.1 Secure Computation

Yao's Garbled Circuit: It is a generic approach for constructing a secure two party protocol of any function f formed by a Boolean circuit. In a garbled-protocol, one party (generator) needs to prepare the encrypted circuits of function f. Another party (evaluator) then obliviously computes the output of the circuits without learning anything about the generator's private inputs. To achieve this, the generator first maps symmetric keys (labels) for each wire of every gate of a circuit, and then generates an encrypted truth table. After getting the truth table, the evaluator will perform an OT protocol to get the keys without revealing his private inputs. More details of this process can found in [36, 44]. The equality test ($\mathsf{EQ}(\cdot)$) is a simple test used in this paper. It takes x and y as input, and outputs 1 if x equals y, and 0 otherwise.

Boolean Secret Sharing: For an l-bit value x, it is XOR-shared into two values over $\mathbb{Z}_2^{\ell}$. Those two values are denoted as $\langle x\rangle_1$ and $\langle x\rangle_2$ respectively, where $\langle x\rangle_1, \langle x\rangle_2 \in \mathbb{Z}_2^{\ell}$. Then we use $\langle x\rangle_1 \oplus \langle x\rangle_2 = x$ to recover x.

Oblivious PRF: Oblivious Pseudorandom Function (OPRF) is one of approaches of MPC, and is achieved by OT extension. OPRF is a protocol in which one party (sender) gets a key k of the Pseudorandom Function (PRF) F while the party (receiver) can obtain the PRF value $\mathsf{F}(k, x)$ with input value x.

Notably, Duong et al. [25] proposes the oblivious distributed key pseudorandom function, which possesses XOR-based homomorphic property. To be specific, Odk-PRF allows the receiver to provide ϕ XOR-shared inputs $\{\langle x\rangle_1, \cdots, \langle x\rangle_\phi\}$. Then this functionality produces distributed keys $\{\langle k\rangle_1, \cdots, \langle k\rangle_\phi\}$ related to the inputs for the sender, and the receiver obtains $\mathsf{F}(\langle k\rangle_i, \langle x\rangle_i)$ $(1 \le i \le \phi)$. In Odk-PRF, the real key can be determined by XORing all distributed keys $k = \oplus_{i=1}^{\phi}\langle k\rangle_i$, and the input can be determined by XORing n inputs as $x = \oplus_{i=1}^{\phi}\langle x\rangle_i$.

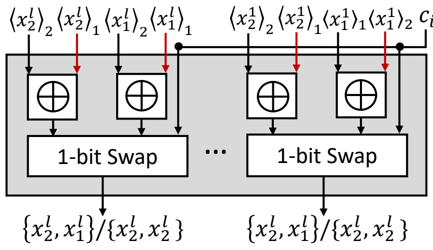

$\langle x_2^l\rangle_2$ $\langle x_2^l\rangle_1$ $\langle x_1^l\rangle_2$ $\langle x_1^l\rangle_1$
$\langle x_2^1\rangle_2$ $\langle x_2^1\rangle_1$ $\langle x_1^1\rangle_1$ $\langle x_1^1\rangle_2$
$c_i=0/1$
1-bit Swap
...
1-bit Swap
$\{x_2^l, x_1^l\}/\{x_2^l, x_2^l\}$
$\{x_2^l, x_1^l\}/\{x_2^l, x_2^l\}$

Fig. 4: The basic circuits of swapping network.

The XOR-based homomorphic property of Odk-PRF ensures $\oplus_{i=1}^{\phi}\mathsf{F}(\langle k\rangle_i, \langle x\rangle_i) = \mathsf{F}(\oplus_{i=1}^{\phi}\langle k\rangle_i, \oplus_{i=1}^{\phi}\langle x\rangle_i) = \mathsf{F}(k, x)$. The formal definition of Odk-PRF functionality is shown as follows:

- $\mathsf{KeyGen}(1^\lambda) \to k$: it takes a security parameter λ as input, and outputs a secret key k.
- $\mathsf{KeyShare}(k, \phi) \to \{\langle k\rangle_1, \cdots, \langle k\rangle_\phi\}$: it takes a PRF key k and the number ϕ as inputs, and outputs ϕ shared keys $\langle k\rangle_1, \cdots, \langle k\rangle_\phi$, where $k = \oplus_{i=1}^{\phi}\langle k\rangle_i$.
- $\mathsf{KeyEval}(\langle k\rangle_i, \langle x\rangle_i) \to F(\langle k\rangle_i, \langle x\rangle_i)$: it takes a PRF key $\langle k\rangle_i$ and a XOR-shared value $\langle x\rangle_i$ as inputs, and outputs a PRF value $\mathsf{F}(\langle k\rangle_i, \langle x\rangle_i)$, where F is an OPRF function.

The security guarantee of Odk-PRF comes from the property of PRF: $\mathsf{F}(k, x)$, $F(\langle k\rangle_i$ and $\langle x\rangle_i)$ are randomized and reveal information about k and x with a negligible probability (e.g., $2^{-\lambda}$).

Circuit-based Shuffling Network: Shuffling Network is an approach for swapping the elements in an array. The work in [36] proposes a shuffling network based on Yao's garbled circuit. In a slightly more detail, a shuffling network is formed with some basic circuits, and each of them takes $2n$ inputs along with an additional set of control bits. Each of the control bits determines whether the fixed pair of elements should be swapped or not.

As shown in Fig. 4, the basic circuits of the shuffling network consists of a series of 1-bit swap modules, two XOR circuits, different input and output wires. The black input wires represent the circuits generator's inputs, and the red ones are controlled by the evaluator. Let c_i represents the i-bit of the permutation seed c. When parsing through this shuffling network, the shares of two ℓ-bit values, $x_1 = x_1^{\ell}\cdots x_1^2 x_1^1$ and $x_2 = x_2^{\ell}\cdots x_2^2 x_2^1$, will be merged by the XOR circuits and swapped according to the value of c_i. If $c_i = 0$, the order of x_1 and x_2 will not be swapped. Otherwise, the evaluator will get the swapped values x_2 and x_1. Specifically, a Waksman network is utilized in a swapping network to produce any of the $n!$ permutation function π with n-bit input c for security.

Next, we extend the above example to the general case. The generator can prepare *a swap circuit* with the inputs of an array $\mathbf{x}_1$ and permutation function π. Then the evaluator will evaluate the circuits with the input of another array $\mathbf{x}_2$. Finally, the evaluator will learn a shuffled array $\pi(\mathbf{x}_1 \oplus \mathbf{x}_2)$. Importantly, the type of the gates (and so π) is known only to the generator. The evaluator learns nothing about generator's private inputs since the circuits are constructed via Yao's garbled circuit.

5 The Construction of PriRanGe

This section starts with the notation used in our protocol and the main idea of PriRanGe, followed by the details of its two building blocks: the DIC Protocol and the SOSF Protocol, and ends with the full-fledged design of PriRanGe.

TABLE 2: Notations

Symbol	Description
λ	The security parameter.
$\mathsf{PRG}(\cdot)$	A pseudo random number generator.
$\mathcal{CS}_{1/2}$	The first cloud server and the second cloud server.
$\mathbf{q}$	The queried set of $\mathcal{C}$.
$\langle \mathbf{q} \rangle_{1/2}$	The share of $\mathbf{q}$ to $\mathcal{CS}_{1/2}$.
$\langle \mathbf{q}.v \rangle_{1/2}$	The share of the base letter of $\mathbf{q}$ in $\mathcal{CS}_{1/2}$.
$\langle \mathbf{q}.pos \rangle_{1/2}$	The share of the position of $\mathbf{q}$ in $\mathcal{CS}_{1/2}$.
$\mathbf{db}$	The set of genomic variants of $\mathcal{DBP}$.
$\langle \mathbf{db} \rangle_{1/2}$	The share of $\mathbf{db}$ to $\mathcal{CS}_{1/2}$.
$\langle \mathbf{db}.v \rangle_{1/2}$	The share of the base letter of $\mathbf{db}$ in $\mathcal{CS}_{1/2}$.
$\langle \mathbf{db}.pos \rangle_{1/2}$	The share of the position of $\mathbf{db}$ in $\mathcal{CS}_{1/2}$.

5.1 Notation

For better readability, we list the notations used in PriRanGe in this section. As shown in Table. 2, a queried array of $\mathcal{C}$ is $\mathbf{q} = \{q_i\}_{i=1}^{m}$, and an array of genomic variants of $\mathcal{DBP}$ is $\mathbf{db} = \{db_j\}_{j=1}^{n}$. One share of the queried array is $\langle \mathbf{q} \rangle_1$, and another share is $\langle \mathbf{q} \rangle_2$. Similarly, the two shares of the database is denoted as $\langle \mathbf{db} \rangle_1$ and $\langle \mathbf{db} \rangle_2$. Then, $\langle \mathbf{q}.pos \rangle_{1/2}$ and $\langle \mathbf{q}.v \rangle_{1/2}$ respectively represent the shared position array and base letter array, which comprise the query array. Similarly, the share of position and base letter of the database are denoted as $\langle \mathbf{db}.v \rangle_{1/2}$ and $\langle \mathbf{db}.v \rangle_{1/2}$, respectively.

5.2 Main Idea of PriRanGe

Firstly, $\mathcal{C}$ is advised by the sequence lab to confirm the range of Single Nucleotide Polymorphisms (SNPs) which he/she needs to query. As we mentioned above, we assume $\mathcal{C}$'s query array is $\mathbf{q} = \{q_1, \cdots, q_m\}$ and the queried range is $[q_1.pos, q_m.pos]$. Then, $\mathcal{C}$ gives the XOR shares of his/her query array to $\mathcal{CS}_1$ and $\mathcal{CS}_2$ for an intersection query. Therefore, $\mathcal{CS}_1$ obtains $\langle \mathbf{q} \rangle_1 = \{\langle q_1 \rangle_1, \cdots, \langle q_m \rangle_1\}$, and $\mathcal{CS}_2$ obtains $\langle q \rangle_2 = \{\langle q_1 \rangle_2, \cdots, \langle q_m \rangle_2\}$, where $\langle \mathbf{q} \rangle_1 \oplus \langle \mathbf{q} \rangle_2 = \mathbf{q}$. Both $\langle \mathbf{q} \rangle_1$ and $\langle \mathbf{q} \rangle_2$ have m elements, and each element consists of the shares of positions and variants, e.g., $\langle q_i \rangle_1 = (\langle q_i.pos \rangle_1, \langle q_i.v \rangle_1), i \in [1, m]$.

Similarly, $\mathcal{DBP}$ outsources his/her genomic variants database $\mathbf{db} = \{db_1, \cdots, db_n\}$ to $\mathcal{CS}_1$ and $\mathcal{CS}_2$ because of the shortage of local computing and storage resources, normally $m << n$. Therefore, $\mathcal{CS}_1$ and $\mathcal{CS}_2$ obtain one XOR-share of $\mathbf{db}$, denoted as $\langle \mathbf{db} \rangle_1$ and $\langle \mathbf{db} \rangle_2$, respectively. Both $\langle \mathbf{db} \rangle_1$ and $\langle \mathbf{db} \rangle_2$ have n items, and each item also consists of the shares of position and variants, e.g., $\langle db_j \rangle_1 = (\langle db_j.pos \rangle_1, \langle db_j.v \rangle_1), j \in [1, n]$.

After $\mathcal{CS}_1$ and $\mathcal{CS}_2$ get shares from $\mathcal{C}$ and $\mathcal{DBP}$, the two servers jointly perform the PriRanGe protocol for responding a range-constrained intersection result to $\mathcal{C}$. The PriRanGe protocol consists of the following three steps:

Step 1: Constraining a queried range. $\mathcal{CS}_1$ and $\mathcal{CS}_2$ firstly constrain the range of query array, which is $[q_1.pos, q_m.pos]$, and select all elements in this range of the database, denoted as $\mathbf{db}_\alpha = \{db_{t_1}, \cdots, db_{t_m}\}$, $|\mathbf{db}_\alpha| = m$. Specifically, servers can not know plaintexts of the queried range $[q_1.pos, q_m.pos]$ in this process. We make use of Yao's garbled circuit, which is introduced in Sec. 4.1, to achieve this goal.

Step 2: Computing distributed intersection. After constraining the queried range, $\mathcal{CS}_1$ possesses two XOR-shared arrays $\langle \mathbf{q}.v \rangle_1$ and $\langle \mathbf{db}_\alpha.v \rangle_1$, and $\mathcal{CS}_2$ also prepares two XOR-shared arrays $\langle \mathbf{q}.v \rangle_2$ and $\langle \mathbf{db}_\alpha.v \rangle_2$. During this step, the two servers interactively calculate the intersection of two arrays $\mathbf{q}.v$ and $\mathbf{db}_\alpha.v$, while the result and the size of $\mathbf{q}.v \cap \mathbf{db}_\alpha.v$ can not be revealed to servers. The plaintext of intersection results is only revealed to $\mathcal{C}$. For this purpose, we design the DIC protocol to calculating the intersection of secret shared arrays, as will be discussed in Sec. 5.3.

Step 3: Shuffling the outsourced database. The shuffling process is formalized as follows: $\mathcal{CS}_1$ owns one share $\langle \mathbf{db} \rangle_1$ and randomly picks a permutation function π_1. Similarity, $\mathcal{CS}_2$ owns another share $\langle \mathbf{db} \rangle_2$ and randomly picks a permutation function π_2. After oblivious shuffling, $\mathcal{CS}_1$ obtains a shuffled share $\langle \mathbf{db}' \rangle_1 = \pi_1\pi_2(\langle \mathbf{db} \rangle_1)$ and $\mathcal{CS}_2$ obtains the other shuffled share $\langle \mathbf{db}' \rangle_2 = \pi_1\pi_2(\langle \mathbf{db} \rangle_2)$, where

$$\begin{cases} \langle \mathbf{db} \rangle_1 \oplus \langle \mathbf{db} \rangle_2 = \mathbf{db}, \\ \langle \mathbf{db}' \rangle_1 \oplus \langle \mathbf{db}' \rangle_2 = \pi_1\pi_2(\mathbf{db}). \end{cases}$$

To realize this functionality, we design the SOSF protocol, as will be discussed in Sec. 5.4. This protocol can be executed by two servers offline, which means they shuffle the databases multiple times to obtain multiple shuffled databases before/after **Steps 1 and 2**. And the cloud servers access to different databases in different queries.

5.3 The Distributed Intersection Computation Protocol

Without loss of generality, we assume two arrays $\mathbf{x} = \{x_1, \cdots, x_m\}$ and $\mathbf{y} = \{y_1, \cdots, y_m\}$ are XOR-shared to two servers respectively. Therefore, $\mathcal{CS}_1$ takes two arrays of shared values $\langle \mathbf{x} \rangle_1 = \{\langle x_1 \rangle_1, \cdots, \langle x_m \rangle_1\}$ and $\langle \mathbf{y} \rangle_1 = \{\langle y_1 \rangle_1, \cdots, \langle y_m \rangle_1\}$ as the inputs of DIC protocol, and $\mathcal{CS}_2$ has the inputs $\langle \mathbf{x} \rangle_2 = \{\langle x_1 \rangle_2, \cdots, \langle x_m \rangle_2\}$ and $\langle \mathbf{y} \rangle_2 = \{\langle y_1 \rangle_2, \cdots, \langle y_m \rangle_2\}$. Then, we utilize the DIC protocol to compute the intersection of $\mathbf{x}$ and $\mathbf{y}$ while keeping the elements in those two arrays and the size of the intersection private for servers.

Formally, we present the DIC protocol in Algorithm 1. Firstly, $\mathcal{CS}_1$ acts as Odk-PRF's receiver and performs m times OPRF with $\mathcal{CS}_2$ (sender), who obtains m keys $\{\langle k_1 \rangle_2, \cdots, \langle k_m \rangle_2\}$ in lines 2-3. Then $\mathcal{CS}_1$ and $\mathcal{CS}_2$ obtain two PRF arrays $\mathsf{F}(\langle k \rangle_2, \langle x \rangle_1)$ and $\mathsf{F}(\langle k \rangle_2, \langle y \rangle_2)$ respectively, in lines 4-5. Similarly, $\mathcal{CS}_2$ acts as a receiver and also performs m times OPRF with $\mathcal{CS}_1$ (sender) in lines 6-9. Therefore, $\mathcal{CS}_1$ and $\mathcal{CS}_2$ obtain $\mathsf{F}(\langle \mathbf{k} \rangle_1, \langle \mathbf{y} \rangle_1)$ and $\mathsf{F}(\langle \mathbf{k} \rangle_1, \langle \mathbf{x} \rangle_2)$ respectively. Those processes are equivalent to execute $\mathsf{KeyShare}(k, \phi)$ m times with $\phi = 2$ in the Odk-PRF functionality.

Subsequently, $\mathcal{CS}_1$ and $\mathcal{CS}_2$ utilize the XOR-based homomorphic property of Odk-PRF to calculate the PRF values of the PRF arrays on they own. As shown in lines 11 and 16, $\mathcal{CS}_1$ computes:

$$\begin{aligned} L_1 &= \mathsf{F}(\langle \mathbf{k}\rangle_1, \langle \mathbf{x}\rangle_1) \oplus \mathsf{F}(\langle \mathbf{k}\rangle_2, \langle \mathbf{y}\rangle_1) \\ &= \{\mathsf{F}(\langle k_1\rangle_1, \langle x_1\rangle_1) \oplus \mathsf{F}(\langle k_1\rangle_2, \langle y_1\rangle_1), \cdots, \\ &\quad \mathsf{F}(\langle k_m\rangle_1, \langle x_m\rangle_1) \oplus \mathsf{F}(\langle k_m\rangle_2, \langle y_m\rangle_1)\} \\ &= \{\mathsf{F}(k_1, \langle x_1\rangle_1 \oplus \langle y_1\rangle_1), \cdots, \mathsf{F}(k_m, \langle x_m\rangle_1 \oplus \langle y_m\rangle_1)\}, \end{aligned}$$

and $\mathcal{CS}_2$ computes:

$$\begin{aligned} L_1 &= \mathsf{F}(\langle \mathbf{k}\rangle_1, \langle \mathbf{x}\rangle_2) \oplus \mathsf{F}(\langle \mathbf{k}\rangle_2, \langle \mathbf{y}\rangle_2) \\ &= \{\mathsf{F}(\langle k_1\rangle_1, \langle x_1\rangle_2) \oplus \mathsf{F}(\langle k_1\rangle_2, \langle y_1\rangle_2), \cdots, \\ &\quad \mathsf{F}(\langle k_m\rangle_1, \langle x_m\rangle_2) \oplus \mathsf{F}(\langle k_m\rangle_2, \langle y_m\rangle_2)\} \\ &= \{\mathsf{F}(k_1, \langle x_1\rangle_2 \oplus \langle y_1\rangle_2), \cdots, \mathsf{F}(k_m, \langle x_m\rangle_2 \oplus \langle y_m\rangle_2)\}, \end{aligned}$$

After that, $\mathcal{CS}_1$ uses the PRF array L_1 and a random number array $\mathbf{s}_1 = \{s_1, s_2, \cdots, s_m\}$, which are produced by a $\mathsf{PRG}(\cdot)$ seeded with s, to interpolate a polynomial $\mathsf{P}(\cdot)$ in lines 12-14. The coefficients of $\mathsf{P}(\cdot)$ are sent to $\mathcal{CS}_2$. Then, $\mathcal{CS}_2$ takes the calculated PRF array L_2 as the inputs of $\mathsf{P}(\cdot)$ to obtain the array $\mathbf{s}_2 = \{s'_1, s'_2, \cdots, s'_m\}$.

If $y_i = x_i$ $(1 \le i \le m)$, from the fact $\langle y_i\rangle_2 \oplus \langle y_i\rangle_1 = \langle x_i\rangle_2 \oplus \langle x_i\rangle_1$, we can get $\langle y_i\rangle_1 \oplus \langle x_i\rangle_1 = \langle y_i\rangle_2 \oplus \langle x_i\rangle_2$. So the PRF values of them equal to each other, i.e., $\mathsf{F}(k_1, \langle y_i\rangle_1 \oplus \langle x_i\rangle_1) = \mathsf{F}(k_1, \langle y_i\rangle_2 \oplus \langle x_i\rangle_2)$. It is remarked that for an authenticated client who is granted access to intersection result of $\mathbf{x}$ and $\mathbf{y}$, it suffices to collect $\mathbf{s}_1$ (from $\mathcal{CS}_1$) and $\mathbf{s}_2$ (from $\mathcal{CS}_2$). Then the client will compare elements s_i and s'_i: $s_i = s'_i$ implies that the element x_i is also in the array $\mathbf{y}$. Therefore, the client can obtain $\mathbf{x} \cap \mathbf{y}$.

5.4 The Secret-Shared Oblivious Shuffling Protocol

The SOSF Protocol aims to shuffle the original XOR-shared arrays $\langle \mathbf{db}\rangle_1 = \{\langle db_1\rangle_1, \cdots, \langle db_n\rangle_1\}$ and $\langle \mathbf{db}\rangle_2 = \{\langle db_1\rangle_2, \cdots, \langle db_n\rangle_2\}$ into two new XOR-shared arrays $\langle \mathbf{db}'\rangle_1 = \{\langle db_{\pi_2\pi_1(1)}\rangle_1, \cdots, \langle db_{\pi_2\pi_1(n)}\rangle_1\}$ and $\langle \mathbf{db}'\rangle_2 = \{\langle db_{\pi_2\pi_1(1)}\rangle_2, \cdots, \langle db_{\pi_2\pi_1(n)}\rangle_2\}$, where the permutation function π_1 is only held by $\mathcal{CS}_1$ and π_2 is only held by $\mathcal{CS}_2$. As shown in Algorithm 2, we can realize the SOSF Protocol based on the circuit-based swapping technique introduced in Sec. 4.

In Algorithm 2, $\mathcal{CS}_1$ first prepares *a swap circuit*, denoted as C^1_{swap}, with inputs of a random array $\mathbf{u}$ and a permutation function π_1 as lines 1-4. Then $\mathcal{CS}_2$ works as a circuit evaluator with inputs of a random array $\mathbf{r}$ to compute $\pi_1(\mathbf{u} \oplus \mathbf{r})$ in line 7. After that, $\mathcal{CS}_2$ permutes the array $\pi_1(\mathbf{u} \oplus \mathbf{r})$ with π_2 to obtain a random array $\pi_2\pi_1(\mathbf{u} \oplus \mathbf{r})$, and takes it as a new share of $\langle \mathbf{db}\rangle$. For correctness, $\mathcal{CS}_1$ needs to obtain the array $\pi_2\pi_1(\langle \mathbf{db}\rangle \oplus \mathbf{u} \oplus \mathbf{r})$ as its new share of $\langle \mathbf{db}\rangle$. To this end, $\mathcal{CS}_1$ generates another *swap circuit* C^2_{swap} with inputs of a random array $\mathbf{v}$ and the permutation function π_1. Thus, $\mathcal{CS}_2$ can evaluate $\pi_1(\langle \mathbf{db}\rangle_2 \oplus \mathbf{r} \oplus \mathbf{v})$ in lines 15-16. After that, $\mathcal{CS}_2$ prepares the last *swap circuit* C^3_{swap} with inputs of $\pi_1(\langle \mathbf{db}\rangle_2 \oplus \mathbf{r} \oplus \mathbf{v})$ and a permutation function π_2. Upon receiving C^3_{swap}, $\mathcal{CS}_1$ inputs $\pi_1(\langle \mathbf{db}\rangle_1 \oplus \mathbf{u} \oplus \mathbf{v})$ and obtains $\pi_2\pi_1(\mathbf{db} \oplus \mathbf{u} \oplus \mathbf{r})$ in lines 21-23. Clearly, neither of the servers can recover the original index of $\mathbf{db}$, and no information about $\mathbf{db}$ is revealed to servers.

Algorithm 1: Distributed Intersection Computation

```
Require:
    $\mathcal{CS}_1$: two shared arrays $\langle \mathbf{x}\rangle_1$, $\langle \mathbf{y}\rangle_1$;
    $\mathcal{CS}_2$: two shared arrays $\langle \mathbf{x}\rangle_2$, $\langle \mathbf{y}\rangle_2$.
Ensure:
    $\mathcal{CS}_1$: an array $\mathbf{s}_1$;
    $\mathcal{CS}_2$: an array $\mathbf{s}_2$.
1: $\mathcal{CS}_1$ and $\mathcal{CS}_2$:
2: $\mathcal{CS}_1$ acts as Odk-PRF's receiver with input arrays $\langle \mathbf{x}\rangle_1$;
3: $\mathcal{CS}_2$ acts as Odk-PRF's sender and obtains a key array:
       $\langle \mathbf{k}\rangle_2 = \{\langle k_1\rangle_2, \cdots, \langle k_m\rangle_2\}$;
4: $\mathcal{CS}_1$ obtains:
       $\mathsf{F}(\langle \mathbf{k}\rangle_2, \langle \mathbf{x}\rangle_1) = \{\mathsf{F}(\langle k_1\rangle_2, \langle x_1\rangle_1), \cdots, \mathsf{F}(\langle k_m\rangle_2, \langle x_m\rangle_1)\}$;
5: $\mathcal{CS}_2$ calculates:
       $\mathsf{F}(\langle \mathbf{k}\rangle_2, \langle \mathbf{y}\rangle_2) = \{\mathsf{F}(\langle k_1\rangle_2, \langle y_1\rangle_2), \cdots, \mathsf{F}(\langle k_m\rangle_2, \langle y_m\rangle_2)\}$;
6: $\mathcal{CS}_2$ acts as Odk-PRF's receiver with input arrays $\langle \mathbf{x}\rangle_2$;
7: $\mathcal{CS}_1$ acts as Odk-PRF's sender and obtains a key array:
       $\langle \mathbf{k}\rangle_1 = \{\langle k_1\rangle_1, \cdots, \langle k_m\rangle_1\}$;
8: $\mathcal{CS}_2$ obtains:
       $\mathsf{F}(\langle \mathbf{k}\rangle_1, \langle \mathbf{x}\rangle_2) = \{\mathsf{F}(\langle k_1\rangle_1, \langle x_1\rangle_2), \cdots, \mathsf{F}(\langle k_m\rangle_1, \langle x_m\rangle_2)\}$;
9: $\mathcal{CS}_1$ calculates:
       $\mathsf{F}(\langle \mathbf{k}\rangle_1, \langle \mathbf{y}\rangle_1) = \{\mathsf{F}(\langle k_1\rangle_1, \langle y_1\rangle_1), \cdots, \mathsf{F}(\langle k_m\rangle_1, \langle y_m\rangle_1)\}$;
10: $\mathcal{CS}_1$:
11: Calculates:
       $\mathsf{F}(\langle \mathbf{k}\rangle_1, \langle \mathbf{x}\rangle_1) \oplus \mathsf{F}(\langle \mathbf{k}\rangle_2, \langle \mathbf{y}\rangle_1)$
12: Generates $m$ random values:
       $\mathbf{s}_1 = \{s_1, \cdots, s_m\} \leftarrow \mathsf{PRG}(s)$;
13: Interpolates a degree $(m-1)$ polynomial $\mathsf{P}(\cdot)$ over points:
       $(\mathsf{F}(k_1, \langle x_1\rangle_1 \oplus \langle y_1\rangle_1), s_1), \cdots, (\mathsf{F}(k_m, \langle x_m\rangle_1 \oplus \langle y_m\rangle_1), s_m)$;
14: Sends the coefficient of polynomial $\mathsf{P}(\cdot)$ to $\mathcal{CS}_2$.
15: $\mathcal{CS}_2$:
16: Calculates:
       $\mathsf{F}(\langle k\rangle_1, \langle \mathbf{x}\rangle_2) \oplus \mathsf{F}(\langle k\rangle_2, \langle \mathbf{y}\rangle_2)$
17: Evaluates the polynomial $\mathsf{P}(\cdot)$ with $m$ inputs:
       $\mathsf{F}(k_1, \langle x_1\rangle_2 \oplus \langle y_1\rangle_2), \cdots, \mathsf{F}(k_m, \langle x_m\rangle_2 \oplus \langle y_m\rangle_2)$;
18: Collects all the output as a set:
       $\mathbf{s}_2 = \{s'_1, \cdots, s'_m\}$.
```

5.5 PriRanGe Description

With the above building blocks, we are ready to present the full-fledged design for range-constrained intersection on genomic data.

Step 1: Constraining a queried range. During this process, two servers first process their own data from $\mathcal{C}$ and $\mathcal{DBP}$ locally. For $\mathcal{CS}_1$, it has two sets $\langle \mathbf{q}.pos\rangle_1 = \{\langle q_i.pos\rangle_1\}_{i=1}^m$ and $\langle \mathbf{db}.pos\rangle = \{\langle db_j.pos\rangle_1\}_{j=1}^n$, and then it XORs all the shares in those two sets one by one to get a new array $\mathbf{Tr}_1 = \{Tr_1^{ij}\}_{i=1,j=1}^{m,n}$, where $Tr_1^{ij} = \langle q_i.pos\rangle_1 \oplus \langle db_j.pos\rangle_1$. Similarly, $\mathcal{CS}_2$ performs the same computation on its private two sets $\langle \mathbf{q}.pos\rangle_2$ and $\langle \mathbf{db}.pos\rangle_2$ to obtain $\mathbf{Tr}_2 = \{Tr_2^{ij}\}_{i=1,j=1}^{m,n}$, where $Tr_2^{ij} = \langle q_i.pos\rangle_2 \oplus \langle db_j.pos\rangle_2$.

Next, the two servers interactively constrain the queried range of $\mathcal{DBP}$'s data without leaking the plaintext of the queried range and $\mathcal{DBP}$'s data. We utilize Yao's garbled circuit to realize a function that securely executes a conditional statement. This interactive operation is shown in Algorithm 3.

In this algorithm, the functionality $\mathsf{EQ}(\cdot)$ represents the equality test. The output of this process is a $0/1$ array τ, which represents whether the corresponding shares of servers are within the queried range. Therefore, $\mathcal{CS}_1$ (resp. $\mathcal{CS}_2$) can determine a shared array of $\mathcal{DBP}$'s genomic variants, denoted as $\langle \mathbf{db}_\alpha\rangle_1 = \{\langle db_{t_1}\rangle_1, \cdots, \langle db_{t_m}\rangle_1\}$ (resp. $\langle \mathbf{db}_\alpha\rangle_2 = \{\langle db_{t_1}\rangle_2, \cdots, \langle db_{t_m}\rangle_2\}$). For example, if $\tau[i][j] =$

Algorithm 2: Secret-shared Oblivious Shuffling

```
Require:
    $\mathcal{CS}_1$: $\langle \mathbf{db} \rangle_1 = \{\langle db_1 \rangle_1, ..., \langle db_n \rangle_1\}$,
        two random vectors $\mathbf{u} = \{u_1, ..., u_n\}$,
        $\mathbf{v} = \{v_q, ..., v_n\}$,
        a permutation $\pi_q$.
    $\mathcal{CS}_2$: $\langle \mathbf{db} \rangle_2 = \{\langle db_1 \rangle_2, ..., \langle db_n \rangle_2\}$,
        a random vector $\mathbf{r} = \{r_1, ..., r_n\}$,
        a permutation $\pi_2$.
Ensure:
    $\mathcal{CS}_1$: $\langle \mathbf{db}' \rangle_1 = \{\langle db_{\pi_2\pi_1(1)} \rangle_1, \cdots, \langle db_{\pi_2\pi_1(n)} \rangle_1\}$;
    $\mathcal{CS}_2$: $\langle \mathbf{db}' \rangle_2 = \{\langle db_{\pi_2\pi_1(1)} \rangle_2, \cdots, \langle db_{\pi_2\pi_1(n)} \rangle_2\}$.
1: $\underline{\mathcal{CS}_1}$:
2: Prepares the swap circuit $\mathsf{C}^1_{swap}$, and $\pi_1$ is control bits;
3: Takes $\mathbf{u} = \{u_1, ..., u_n\}$ as the input of $\mathsf{C}^1_{swap}$;
4: Gives $\mathsf{C}^1_{swap}$ to $\mathcal{CS}_2$.
5: $\underline{\mathcal{CS}_2}$:
6: Takes $\mathbf{r} = \{r_1, ..., r_n\}$ as the input of $\mathsf{C}^1_{swap}$;
7: Computes:
       $\pi_1(\mathbf{u} \oplus \mathbf{r}) = \{u_{\pi_1(1)} \oplus r_{\pi_1(1)}, ..., u_{\pi_1(n)} \oplus r_{\pi_1(n)}\}$;
8: Permutes $\pi_1(\mathbf{u} \oplus \mathbf{r})$ with $\pi_2$.
9: Takes $\pi_2\pi_1(\mathbf{u} \oplus \mathbf{r})$ as $\langle \mathbf{db}' \rangle_2$.
10: $\underline{\mathcal{CS}_1}$:
11: Prepares the swap circuit $\mathsf{C}^2_{swap}$, and $\pi_1$ is control bits;
12: Takes $\mathbf{v} = \{v_q, ..., v_n\}$ as $\mathsf{C}^2_{swap}$'s input,
13: Give $\mathsf{C}^2_{swap}$ to $\mathcal{CS}_2$.
14: $\underline{\mathcal{CS}_2}$:
15: Takes $\langle \mathbf{db} \rangle_2 \oplus \mathbf{r}$ as the input of $\mathsf{C}^2_{swap}$;
16: Computes $\pi_1(\langle \mathbf{db} \rangle_2 \oplus \mathbf{r} \oplus \mathbf{v})$;
17: Prepares the swap circuit $\mathsf{C}^3_{swap}$, and $\pi_2$ is control bits;
18: Takes $\pi_1(\langle \mathbf{db} \rangle_2 \oplus \mathbf{r} \oplus \mathbf{v})$ as the input of $\mathsf{C}^3_{swap}$;
19: Gives $\mathsf{C}^3_{swap}$ to $\mathcal{CS}_1$.
20: $\underline{\mathcal{CS}_1}$:
21: Takes $\pi_1(\langle \mathbf{db} \rangle_1 \oplus \mathbf{u} \oplus \mathbf{v})$ as the input of $\mathsf{C}^3_{swap}$;
22: Computes:
       $\pi_2\pi_1(\langle \mathbf{db} \rangle_1 \oplus \langle \mathbf{db} \rangle_2 \oplus \mathbf{u} \oplus \mathbf{v} \oplus \mathbf{v} \oplus \mathbf{r})$
     $= \pi_2\pi_1(\mathbf{db} \oplus \mathbf{u} \oplus \mathbf{r})$;
23: Takes $\pi_2\pi_1(\mathbf{db} \oplus \mathbf{u} \oplus \mathbf{r})$ as $\langle \mathbf{db}' \rangle_1$.
```

Algorithm 3: Range Constraint

```
Require:
    $\mathcal{CS}_1$ inputs $\mathbf{Tr}_1$;
    $\mathcal{CS}_2$ inputs $\mathbf{Tr}_2$.
Ensure:
    $\mathcal{CS}_1$ outputs an array $\tau[\,][\,]$.
1: $\underline{\mathcal{CS}_1 \text{ and } \mathcal{CS}_2}$:
2: for $i$ from $1$ to $m$
3:     for $j$ from $1$ to $n$
4:     if $\mathsf{EQ}(\mathbf{Tr}_1^{ij}, \mathbf{Tr}_2^{ij})$
5:         $\tau[i][j] == 1$
6:     else
7:         $\tau[i][j] == 0$
8: return $\tau$;
```

1, $\mathcal{CS}_1$ and $\mathcal{CS}_2$ learn that their shares $\langle db_j \rangle_1$ and $\langle db_j \rangle_2$ are within the queried range, and $\langle db_j \rangle_1 \oplus \langle db_j \rangle_2 = \langle db_j \rangle \in \mathbf{db}_\alpha$.

According to Yao's garbled circuit, the circuits evaluator runs OT protocol with the circuits generator to obliviously obtain its garbled inputs associated with its private inputs. Then the evaluator evaluates the garbled circuits to get final results. If the final result can be public, the evaluator will directly send the results to the generator. In our setup, $\mathcal{CS}_1$ and $\mathcal{CS}_2$ can be either the circuits generator or the circuits evaluator. Besides, we consider the final output is public to both servers.

Step 2: Computing distributed intersection. In this step, we utilize our DIC protocol. The two shared input arrays of $\mathcal{CS}_1$ (resp. $\mathcal{CS}_2$) is $\langle \mathbf{q}.v \rangle_1 = \{\langle q_1.v \rangle_1, \cdots, \langle q_m.v \rangle_1\}$ and $\langle \mathbf{db}_\alpha.v \rangle_1 = \{\langle db_{t_1}.v \rangle_1, \cdots, \langle db_{t_m}.v \rangle_1\}$ (resp. $\langle \mathbf{q}.v \rangle_2$ and $\langle \mathbf{db}_\alpha.v \rangle_2$), which are corresponding to the arrays $\langle \mathbf{x} \rangle_1$ and $\langle \mathbf{y} \rangle_1$ (resp. $\langle \mathbf{x} \rangle_2$ and $\langle \mathbf{y} \rangle_2$) in Algorithm 2.

As shown in this algorithm, upon receiving the PRF values of each element in the shared array $\langle \mathbf{q}.v \rangle_1$ obliviously (line 4), $\mathcal{CS}_1$ acts as Odk-PRF's sender to get a key array $\langle \mathbf{k} \rangle_1$ and computes $\mathsf{KeyEval}(\langle \mathbf{k} \rangle_1, \langle \mathbf{db}_\alpha.v \rangle_1)$ (lines 7, 9). Similarly, $\mathcal{CS}_2$ acts as Odk-PRF's sender to get a key array $\langle \mathbf{k} \rangle_2$ and computes $\mathsf{KeyEval}(\langle \mathbf{k} \rangle_2, \langle \mathbf{db}_\alpha.v \rangle_2)$ (lines 3, 5), then it receives the PRF values of each element in the shared array $\langle \mathbf{q}.v \rangle_2$ obliviously (line 8). After that, $\mathcal{CS}_1$ calculates $\mathsf{F}(\langle \mathbf{k} \rangle_2, \langle \mathbf{q}.v \rangle_1) \oplus \mathsf{F}(\langle \mathbf{k} \rangle_1, \langle \mathbf{db}_\alpha.v \rangle_1)$ and selects a random array $\mathbf{s}_1$ to interpolate a polynomial $\mathsf{P}(\cdot)$, whose coefficients are sent to $\mathcal{CS}_2$ (lines 11-14). Next, $\mathcal{CS}_2$ calculates the PRF values of the array $\mathsf{F}(\langle \mathbf{k} \rangle_1, \langle \mathbf{q}.v \rangle_2) \oplus \mathsf{F}(\langle \mathbf{k} \rangle_2, \langle \mathbf{db}_\alpha.v \rangle_2)$ and takes them as $\mathsf{P}(\cdot)$'s input. Then $\mathcal{CS}_2$ obtains the evaluated results $\mathbf{s}_2$ (line 18). $\mathbf{s}_1$ and $\mathbf{s}_2$ are sent to the queried client, who is able to tell which variants of his/her query array are within the queried range of $\mathcal{DBP}$'s database $\mathbf{db}$.

To sum up, the size of intersection and the shares owned by each server are private for another server. The security and correctness of this protocol are based on the properties of Odk-PRF.

Step 3: Shuffling the outsourced database. Once the $\mathcal{CS}_1$ and $\mathcal{CS}_2$ returned the queried results to $\mathcal{C}$, $\mathcal{CS}_1$ and $\mathcal{CS}_2$ can perform the SOSF protocol with each other to shuffle the database.

Overall, this protocol aims to protect $\mathcal{C}$'s access pattern since it breaks the link of the queried range in multiple queries. And the unlinkability of each query is able to prevent servers from inferring the plaintext of genomic data via statistical approaches. It is worth mentioning that, in real implementation, the SOSF protocol can be executed offline, i.e., multiple versions of the shuffled databases are prepared in advance and a different shuffled instance is used for each query. This implementation trick improves efficiency without sacrificing security of the access pattern.

6 Security Analyses

Our PriRanGe protocol allows a client to launch a query in a database over the clouds, therefore, we formalize our security analyses in the simulation-based real/ideal world model [45, 46]. According to the definition of simulation-based proof, the security of a protocol reduces to the indistinguishability of the adversaries' views in real/ideal worlds. The real world view (i.e., input, internal randomness, received messages from the other party) of a semi-honest attacker is from the execution of the PriRanGe protocol in the real world, and the view of the adversaries in the ideal world is simulated based on the inputs and outputs of the corresponding party in real world. Therefore, we will define the leakage of PriRanGe protocol and formalize the ideal world. Then, we can simulate the adversaries' view in ideal world. Before that, let us firstly take look at the formal definition of security requirements. We employ the composition theory mentioned in [38, 47, 48] to prove the security of in the semi-honest model.

Definition 2. *Suppose that a protocol Π securely implements a function $f(x, y)$, where x, y are the inputs of parties $\mathcal{A}$ and $\mathcal{B}$, respectively. Let $view_{\mathcal{A}}(x, y) = (x, r_{\mathcal{A}}, m_1, \cdots, m_k)$ and $view_{\mathcal{B}}(x, y) = (y, r_{\mathcal{B}}, m_1, \cdots, m_k)$ to denote the views of $\mathcal{A}$ and $\mathcal{B}$ respectively during the execution of Π, where $r_{\mathcal{A}}$ and $r_{\mathcal{B}}$ represent randomness of $\mathcal{A}$ and $\mathcal{B}$, and m_i denotes the i-th message passed between the two parties. Let $\mathcal{O}_{\mathcal{A}}(x, y)$ and $\mathcal{O}_{\mathcal{B}}(x, y)$ be Π's output for $\mathcal{A}$ and $\mathcal{B}$. Then we say the protocol Π is secure against semi-honest adversaries if for any probabilistic polynomial-time (PPT) adversary there exist PPT simulators $\mathcal{S}_1$ and $\mathcal{S}_2$ such that:*

$$\begin{aligned}(\mathcal{S}_1(x, f(x, y)), f(x, y)) &\approx_c (view_{\mathcal{A}}(x, y), \mathcal{O}_{\mathcal{A}}(x, y)),\\ (\mathcal{S}_2(y, f(x, y)), f(x, y)) &\approx_c (view_{\mathcal{B}}(x, y), \mathcal{O}_{\mathcal{B}}(x, y)),\end{aligned} \tag{1}$$

where $\approx_c$ denotes computational indistinguishability.

Inputs of $\mathcal{CS}_1$ and $\mathcal{CS}_2$: in our scheme, $\mathcal{CS}_1$ and $\mathcal{CS}_2$ are considered to be potential adversaries. So the information input to them are defined as follows.

- Inputs to $\mathcal{CS}_1$, denoted as $\mathcal{L}_1$, includes: one share of the query sequences $\langle \mathbf{q} \rangle_1 = \{\langle q_i \rangle_1\}_{i=1}^m$; one share of the database $\langle \mathbf{db} \rangle_1 = \{\langle db_j \rangle_1\}_{j=1}^m$; the swap circuit C_{swap} with the permutation π_1;
- Inputs to $\mathcal{CS}_2$, denoted as $\mathcal{L}_2$, includes: another share of the query sequences $\langle \mathbf{q} \rangle_2 = \{\langle q_i \rangle_2\}_{i=1}^m$; another share of the database $\langle \mathbf{db} \rangle_2 = \{\langle db_j \rangle_2\}_{j=1}^m$; the swap circuit C_{swap} with the permutation π_2.

According to the above information, we construct two simulators $\mathcal{S}_1$ and $\mathcal{S}_2$ to simulate the real world view of $\mathcal{CS}_1$ and $\mathcal{CS}_2$. For Probability Polynomial Time (PPT) adversaries $\mathcal{A}_1$ (who corrupts $\mathcal{CS}_1$) and $\mathcal{A}_2$ (who corrupts $\mathcal{CS}_2$), their participation in the ideal world and interactions with the simulators are characterized as the following experiments:

- In **Step 1**, the adversary $\mathcal{A}_1$ computes the circuit of functionality $\mathsf{EQ}(\cdot)$ with $\mathcal{S}_1$.
- In **Step 2**, the adversary $\mathcal{A}_1$ chooses two shared input arrays $\langle \mathbf{q}.v \rangle_1$ and $\langle \mathbf{db}_\alpha.v \rangle_1$, and sends $\langle \mathbf{q}.v \rangle_1$ to $\mathcal{S}_1$. On receiving the message from $\mathcal{A}_1$, $\mathcal{S}_1$ randomly sets $\langle \mathbf{k} \rangle^{sim}$ and sends a PRF value $\mathsf{F}^{sim}(\langle \mathbf{k} \rangle^{sim}, \langle \mathbf{q}.v \rangle_1)$ for $\mathcal{A}_1$.
- In **Step 3**, the adversary $\mathcal{A}_1$ selects control bits ϕ_1 and a random array $\mathbf{u} = \{u_1, \cdots, u_n\}$ as the input of the swap circuit C^1_{swap} to $\mathcal{S}_1$. On receiving the circuit C^1_{swap}, $\mathcal{S}_1$ will randomly generate the simulated values $\langle \mathbf{db}' \rangle_2^{sim}$, and sends the simulated swap circuit C^{sim}_{swap} to $\mathcal{A}_1$.

Similarly, the interactions of $\mathcal{A}_2$ and the simulator $\mathcal{S}_2$ can be defined to produce the simulated view for $\mathcal{A}_2$. Note that, in the semi-honest threat model, the adversaries $\mathcal{A}_1$ and $\mathcal{A}_2$ will not corrupt the two servers $\mathcal{CS}_1$ and $\mathcal{CS}_2$ at the same time. Therefore, we only need to consider the scenarios where two clouds are respectively corrupted by two adversaries.

Next, we give the proof that the simulated views of $\mathcal{A}_1$ and $\mathcal{A}_2$ is indistinguishable from what $\mathcal{S}_1$ and $\mathcal{S}_2$ simulated, respectively. We note that secret-sharing operations and the garbled circuits in our PriRanGe protocol are secure under the semi-honest model. The EQ test and the swap circuits are applications of Yao's garbled circuits, whose formal security proof can be found in [49]. Besides, the security of Odk-PRF is based on the OT extension [25, 28]. With these facts, the security proof of our protocols is presented as follows.

Theorem 6.1. *As long as the Odk-PRF and secret-sharing operations are secure against semi-honest adversaries, the DIC protocol is secure under the semi-honest model.*

Proof. DIC protocol includes the generation of PRF values, the data transferred between $\mathcal{CS}_1$ and $\mathcal{CS}_2$ and a polynomial interpolation operations. Since the Odk-PRF generation is secure against semi-honest adversaries, any PPT adversaries cannot distinguish the simulator's views from the Odk-PRF outputs. Thus $\mathcal{A}_1$ can not distinguish the PRF value $\mathsf{F}(\langle \mathbf{k} \rangle_2, \langle \mathbf{q}.v \rangle_1)$ and the value $\mathsf{F}^{sim}(\langle \mathbf{k} \rangle^{sim}, \langle \mathbf{q}.v \rangle_1)$ simulated by $\mathcal{S}_1$. After $\mathcal{CS}_1$ obtains PRF values after performing OPRF with $\mathcal{CS}_2$, there is no data that transferred from $\mathcal{CS}_2$ to $\mathcal{CS}_1$. $\mathcal{A}_1$ who corrupts $\mathcal{CS}_1$ can not distinguish its views in real and ideal worlds. Secondly, the simulator $\mathcal{S}_2$ will simulate $\mathcal{A}_2$'s view $\mathcal{L}_2$. Therefore, $\mathcal{S}_2$ computes a $(m-1)$ polynomial $\mathsf{P}'(\cdot)$ with m randomly selected points. $\mathcal{A}_2$ can not distinguish $\mathsf{P}'(\cdot)$ and $\mathsf{P}(\cdot)$. In conclusion, because $\mathcal{S}_1$ (resp. $\mathcal{S}_2$) can simulate $\mathcal{A}_1$'s (resp. $\mathcal{A}_2$'s) view, the DIC protocol is secure under the semi-honest model. □

Theorem 6.2. *As long as secret-sharing operations, garbled circuits are secure against semi-honest adversaries, the SOSF protocol is secure under the semi-honest model.*

Proof. To prove the security of SOSF protocol, we first construct a simulator $\mathcal{S}_1$ to simulate $\mathcal{A}_1$'s view. In real world, the message C^3_{swap} that $\mathcal{A}_1$ received in line 19 of Algorithm 2 needs to be simulated. Then, $\mathcal{S}_1$ prepares the swap circuits C^{sim}_{swap}, whose inputs are m randomly picked strings $\{R_1, \cdots, R_m\} \subset \mathbb{Z}_2^l$ and a permutation function π'. Recall that C^3_{swap} is formed by garbled circuits for randomizing and shuffling the input of $\mathcal{A}_1$, the outputs of C^3_{swap} are random from $\mathcal{A}_1$'s view. So $\mathcal{A}_1$ cannot distinguish the evaluation results of C^3_{swap} and C^{sim}_{swap}. Next, we construct a simulator $\mathcal{S}_2$ to simulate $\mathcal{A}_2$'s view. Similar to the case of $\mathcal{A}_1$, $\mathcal{S}_2$ also constructs swap circuits with randomly picked inputs to simulate the messages $\mathcal{CS}_2$ received in lines 4 and 13 of Algorithm 3. To conclude, because $\mathcal{S}_1$ (resp. $\mathcal{S}_2$) can simulate $\mathcal{A}_1$'s (resp. $\mathcal{A}_2$'s) view, the SOSF protocol is secure under the semi-honest model. □

Besides, we also need to analyze that PriRanGe can meet the security requirements as stated before.

Database/query confidentiality. We only need to consider data confidentiality for the two servers ($\mathcal{CS}_1$ and $\mathcal{CS}_2$) and the client $\mathcal{C}$, because $\mathcal{DBP}$ remains offline after uploading the shares of $\mathbf{db}$. PriRanGe consists of constraining a queried range via Yao's garbled circuit, and performing DIC and SOSF protocols, all of which are provable secure as shown above. The database and query are thus confidential to the two servers. For $\mathcal{C}$, he/she receives results $\mathbf{s}_1$ and $\mathbf{s}_2$ from $\mathcal{CS}_1$ and $\mathcal{CS}_2$, respectively. Note that $\mathbf{s}_1$ and $\mathbf{s}_2$ are two sets of random string, which reveals nothing other than the desirable intersection result.

Access pattern protection. Access pattern refers to how **db** is accessed upon receiving $\mathcal{C}$'s query. If $\mathcal{CS}_1$ and $\mathcal{CS}_2$ observe multiple queries, they can record access patterns associated with different queries. For example, the most frequently accessed range (though the range and variants within the range are still confidential) may be linked with certain most common diseases. However, in our SOSF protocol, two servers will continuously shuffle the database share for every query. The pattern of how the data shares are accessed has been refreshed after each protocol instance. Therefore, the recorded pattern of $\mathcal{CS}_1$ and $\mathcal{CS}_2$ will become uniform.

7 Performance Evaluation

In this section, we analyze the performance of our PriRanGe protocol in terms of running time, computation and communication cost. Besides, we also compare PriRanGe protocol with Sun's range query protocol [4]. To clarify the efficiency of our protocol, we also make a comparison between existing shuffling method [17] and our SOSF protocol.

We implement all protocols in C++, specifically, garbled circuit and swap circuits are constructed by ABY framework [50] and Odk-PRF is achieved by libOTe [51].

Our experiments are conducted on a computer equipped with an AMD Ryzen 5 3600 3.60GHz CPU and 16GB of memory. With this computer, all tests are performed on two virtual Linux machines in the same Local Area Network (LAN). One of the virtual machine is used as $\mathcal{CS}_1$ and the other serves as $\mathcal{CS}_2$. And these two parties are connected to the local host with 1Gbps bandwidth and a 0.3ms Round-Trip Time (RTT) for the LAN setting. Specifically, we measure the total communication cost for all the messages transmitted between by $\mathcal{CS}_1$ and $\mathcal{CS}_2$, including the onetime setup cost incurred by keys generation and base-OT initialization. We measure the end-to-end running time as the time of transferring messages between two servers under the LAN setting.

In our experiments, we used Homo Sapiens Mitochondrion Complete Genome in NCBI [52] that consists of 16570 positions, and each position has a nucleotide. Therefore, it takes 17 bits to represent a genomic item in our experiments, which includes a position and a base letter. And we set symmetric security parameter $\lambda = 128$. To evaluate the scalability of our protocol, we expand our database to 100 times (million size) the original size.

TABLE 3: Running time of basic operations on XOR-shared data.

Operations	Stage	Average
Equality test	Time (µs)	1.69
	Comm (byte)	262.23
Odk-PRF	Time (µs)	0.52
	Comm (byte)	20.48
Swap circuit	Time (µs)	10.56
	Comm (byte)	1.53

7.1 Asymptotic Analyses

In this section, we will analyze the performance of each basic operation on XOR-shared data in PriRanGe. As described in Sec. 5.5, $\mathcal{CS}_1$ and $\mathcal{CS}_2$ need to perform a functionality $\mathsf{EQ}(\cdot)$ in **Step 1**. Since each share of the query array and array in database needs to be computed and compared one by one, this complexity is $O(mn)$. Therefore, we test this functionality based on Yao's garbled circuit in ABY framework [50], and use SIMD operations [53] to speed up this process. Then, in **Step 2**, we evaluate the efficiency of the DIC protocol, which is mainly based on Odk-PRF. For each server, it runs an Odk-PRF protocol with another server, then it can obtain m PRF values with m keys since this protocol is based on OT extension. So its complexity is linear with the size of query array, i.e., $O(m)$. Besides, there is also a process of polynomial computation in our DIC protocol. The computation cost of computing a m point-value polynomial is $O(mlogm^2)$. Therefore, we can compute the computation cost of our DIC protocol, which is $O(mlogm^2)$. Compare to the proposed secure (Genome-wide Range Query) GRQ search protocol in [4], the computation complexity of GRQ is $O(mlgn)$, where m is the size of query array and n is the total short sequences in database. Finally, the servers perform SOSF protocol with each other to shuffle their database in **Step 3**. The SOSF protocol is mainly based on the *swap circuit*, and $\mathcal{CS}_1$ and $\mathcal{CS}_2$ will use three swap circuits. The complexity is thus $O(3n)$ and linear with the size of database. To facilitate description of each basic operation in our protocol, we further test the running time and communication cost of the basic operations on secrete shared data, the results for a single running instance are shown in Table 3. Clearly, **Step 1** (with complexity $O(mn)$) will occupy much more time and communication cost than that of **Step 2** (with complexity $O(m)$) and **Step 3** (with complexity $O(3n)$) due to the differences in cost of their basic operations.

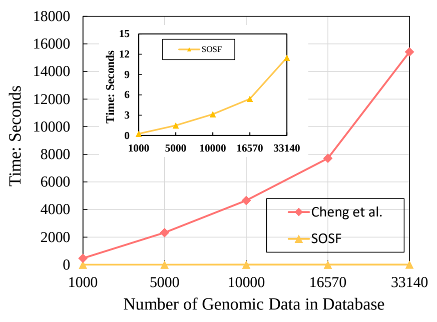
7.2 Experimental Performance

In this section, we analyze the performance of our PriRanGe protocol experimentally. The computation time and communication cost of **Step 1** and **Step 2** of PriRanGe are listed in Table 4, and the efficiency of **Step 3** is depicted in Fig. 5(a) and Fig. 5(b).

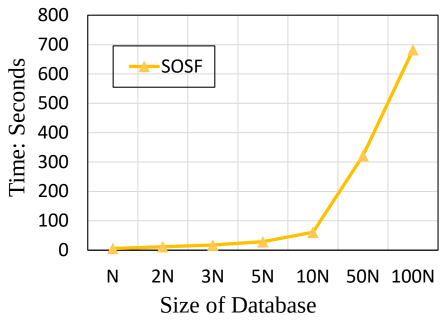
TABLE 4: The running time and communication cost of **Step 1** and **Step 2** (m is the size of queried array and n is the dataset size).

Number of Variants	Stage	**Step 1**	**Step 2**
m = 50, n = 16570	Time (s)	1.45	0.064
	Comm (MB)	207.21	0.041
m = 100, n = 16570	Time (s)	2.90	0.093
	Comm (MB)	414.42	0.061
m=200, n=16570	Time (s)	5.87	0.18
	Comm (MB)	828.85	0.083
m = 300, n = 16570	Time (s)	8.74	0.25
	Comm (MB)	1243.27	0.011
m = 500, n = 16570	Time (s)	14.57	0.42
	Comm (MB)	2072.11	0.15

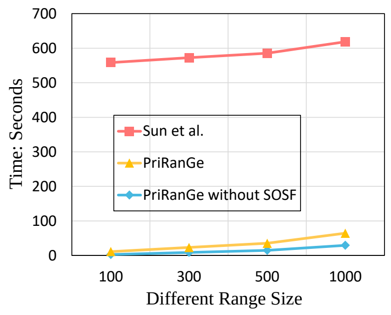
We consider different sizes of queried range of $\mathcal{C}$, which is $m = \{50, 100, 200, 300, 500\}$. In **Step 1**, $\mathcal{CS}_1$ and $\mathcal{CS}_2$ need



(a) The performance of oblivious shuffling protocols.



(b) The performance of SOSF protocol.



(c) The performance comparison.

Fig. 5: The performance evaluation of our protocols.

to constrain a queried range in the database, which requires mn times $\mathsf{EQ}(\cdot)$ operations. In **Step 2**, $\mathcal{CS}_1$ and $\mathcal{CS}_2$ execute the DIC protocol. The DIC protocol includes the evaluation of $2m$ times OPRF and the operations of a polynomial with order $m-1$, and the executing time and communication cost of DIC is irrelevant to the size of n. As a result, the DIC protocol is more efficient than previous range query schemes [4] because the running time of computing the intersection of a certain range is irrelevant to the database size.

To explain our protocol in more detail, we give a running time and communication cost in **Step 1** and **Step 2** as depicted in Table 4. It is clear that the cost of **Step 1** increases linearly with respect to mn and the cost of **Step 2** increases linearly with respect to m, which agrees with the asymptotic analyses in Section 7.1.

In **Step 3**, the servers perform the SOSF protocol to shuffle the database. This process can be executed offline, since $\mathcal{CS}_1$ and $\mathcal{CS}_2$ can prepare lots of shuffled databases before $\mathcal{C}$ launches a query. Then the two servers use a new shuffled database after they returned a queried result to $\mathcal{C}$. The performance of SOSF is shown in Fig. 5(a). We compare the time efficiency of our shuffling protocol with the shuffling protocol in [17], which employs Paillier homomorphic encryption scheme [54] to achieve the same functionality as our SOSF protocol. Our SOSF protocol uses $O(n \log n)$ symmetric-key operations and does not involve time-consuming operations, and the shuffling protocol in [17] involves totally $2n$ encryptions, $2n$ homomorphic and n multiplications over Paillier cryptosystem. As shown in Fig. 5(a), SOSF is faster than the method in [17] by a magnitude of 3. Besides, we evaluate our SOSF protocol on a million size database. As presented in Fig 5(b), n equals to the size of the database (16570). When the database has 16570×100 (1 million) items, the running time of shuffling process is around 11 minutes.

Furthermore, we present the overall running time of the PriRanGe protocol in Fig. 5(c) through comparing it to the protocol in [4]. In Fig. 5(c), we compare the range query protocol in [4] with our PriRanGe protocol and PriRanGe without the SOSF protocol. It is worth noting that the SOSF protocol is responsible for hiding data access patterns, Sun et al. did not consider access pattern protection. And the comparison to the method in [17] is not included since their shuffling protocol is too slow as tested above. Considering the different range sizes of both protocols, the most time-consuming process of the work in [4] is the index processing and loading. As shown in Fig. 5(c), for different size of query range, i.e., $m \in [100, 1000]$, and fixed dataset size $n = 16,570$, our design consistently outperforms the method in [4].

8 Discussion

In this section, we will present some discussion on the practical application and advantages of our protocols, as well as its potential to support different queries with scaleablity. In our paper, we have proposed two sub-protocols (DIC and SOSF) and use those protocols as building blocks to form our PriRanGe protocol. We can utilize those two sub-protocols for other scenarios and support other kinds of queries, since those protocols are of independent interest.

For the DIC protocol, it aims to achieve a range-constrained intersection query for genomic data in the outsourced setting. It can also be applied for generic range query. Consider a scenario where a client has a query array $\mathbf{x}$ and wants to launch a range query in an outsourced database $\mathbf{db}$ of a database provider, that is, to check whether each element of $\mathbf{x}$ is in the specific range of $\mathbf{db}$ or not. We can apply our DIC protocol to this scenario. Note that, the client and the database can also play the roles of $\mathcal{CS}_1$ and $\mathcal{CS}_2$ in our model. In this regard, the client and the database provider invoke our DIC protocol locally rather than in the outsourced setting. Meanwhile, different from the traditional secure range query method, the queried range is kept private in our protocol. For the SOSF protocol, it achieves the functionality that shuffles the original secret-shared array to newly secret-shared arrays between two parties. As stated in [17], the shuffling protocol can hide data access patterns, since the indices of items in the database are kept private to prevent inference attacks. Therefore, our SOSF protocol can be used as a building block and applied to other secret-shared two-cloud based protocols for hiding data access patterns. In conclusion, our sub-protocols are generic and can be transplanted to other types of privacy-preserving queries.

9 Conclusion

In this paper, we have proposed a privacy-preserving range-constrained intersection query scheme over outsourced genomic data. Besides query and database confidentiality, the extra feature of the proposed PriRanGe protocol is access pattern protection. Importantly, these strong security guarantees are mostly achieved by symmetric encryptions. We theoretically proved that PriRanGe is secure under the semi-honest model. Further experimental results demonstrate that PriRanGe is faster than constructions using asymmetric encryption by a magnitude of 3, and also faster than certain symmetric constructions which have complex indexing structures.

10 Acknowledgment

The authors wish to thank the anonymous reviewers for their valuable comments that contributed to the improved quality of this paper. Jian Weng was partially supported by National Key R&D Plan of China (Grant Nos. 2020YFB1005600), National Natural Science Foundation of China (Grant Nos. 61825203 and U1736203), Major Program of Guangdong Basic and Applied Research Project (Grant No. 2019B030302008), and Guangdong Provincial Science and Technology Project(Grant Nos. 2019B010137002 and 2017B010111005).

References

[1] S. J. Heerema and C. Dekker, "Graphene nanodevices for DNA sequencing," *Nature Nanotechnology*, vol. 11, no. 2, pp. 127–136, 2016.

[2] G. Danezis and E. De Cristofaro, "Fast and private genomic testing for disease susceptibility," in *Proceedings of the 13th ACM Workshop on Privacy in the Electronic Society*, 2014, pp. 31–34.

[3] A. Khan and A. Mathelier, "Intervene: A tool for intersection and visualization of multiple gene or genomic region sets," *BMC Bioinformatics*, vol. 18, no. 1, pp. 1–8, 2017.

[4] W. Sun, N. Zhang, W. Lou, and Y. T. Hou, "When gene meets cloud: Enabling scalable and efficient range query on encrypted genomic data," in *INFOCOM*. IEEE, 2017, pp. 1–9.

[5] X. Ding, E. Ozturk, and G. Tsudik, "Balancing security and privacy in genomic range queries," in *Proceedings of the 18th ACM Workshop on Privacy in the Electronic Society*, 2019, pp. 106–110.

[6] K. El Emam, E. Jonker, L. Arbuckle, and B. Malin, "A systematic review of re-identification attacks on health data," *PloS One*, vol. 6, no. 12, p. e28071, 2011.

[7] N. Homer, S. Szelinger, M. Redman, D. Duggan, W. Tembe, J. Muehling, J. V. Pearson, D. A. Stephan, S. F. Nelson, and D. W. Craig, "Resolving individuals contributing trace amounts of dna to highly complex mixtures using high-density snp genotyping microarrays," *PLoS genetics*, vol. 4, no. 8, p. e1000167, 2008.

[8] N. Von Thenen, E. Ayday, and A. E. Cicek, "Re-identification of individuals in genomic data-sharing beacons via allele inference," *Bioinformatics*, vol. 35, no. 3, pp. 365–371, 2019.

[9] R. Li, A. X. Liu, A. L. Wang, and B. Bruhadeshwar, "Fast and scalable range query processing with strong privacy protection for cloud computing," *IEEE/ACM Transactions On Networking*, vol. 24, no. 4, pp. 2305–2318, 2015.

[10] C. Zuo, S. Sun, J. K. Liu, J. Shao, J. Pieprzyk, and L. Xu, "Forward and backward private dsse for range queries," *IEEE Transactions on Dependable and Secure Computing*, 2020.

[11] J. Liang, Z. Qin, S. Xiao, J. Zhang, H. Yin, and K. Li, "Privacy-preserving range query over multi-source electronic health records in public clouds," *Journal of Parallel and Distributed Computing*, vol. 135, pp. 127–139, 2020.

[12] E. De Cristofaro, S. Faber, and G. Tsudik, "Secure genomic testing with size-and position-hiding private substring matching," in *Proceedings of the 12th ACM Workshop on Privacy in the Electronic Society*, 2013, pp. 107–118.

[13] J. S. Sousa, C. Lefebvre, Z. Huang, J. L. Raisaro, C. Aguilar-Melchor, M.-O. Killijian, and J.-P. Hubaux, "Efficient and secure outsourcing of genomic data storage," *BMC Medical Genomics*, vol. 10, no. 2, pp. 15–28, 2017.

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

[15] M. M. Al Aziz, M. Z. Hasan, N. Mohammed, and D. Alhadidi, "Secure and efficient multiparty computation on genomic data," in *Proceedings of the 20th International Database Engineering & Applications Symposium*, 2016, pp. 278–283.

[16] M. Nassar, Q. Malluhi, M. Atallah, and A. Shikfa, "Securing aggregate queries for DNA databases," *IEEE Transactions on Cloud Computing*, vol. 7, no. 03, pp. 827–837, 2019.

[17] K. Cheng, Y. Hou, and L. Wang, "Secure similar sequence query on outsourced genomic data," in *Proceedings of the 2018 on Asia Conference on Computer and Communications Security*, 2018, pp. 237–251.

[18] T. Schneider and O. Tkachenko, "EPISODE: Efficient privacy-preserving similar sequence queries on outsourced genomic databases," in *Proceedings of the 2019 ACM Asia Conference on Computer and Communications Security*, 2019, pp. 315–327.

[19] Y. Zheng, H. Cui, C. Wang, and J. Zhou, "Privacy-preserving image denoising from external cloud databases," *IEEE Transactions on Information Forensics and Security*, vol. 12, no. 6, pp. 1285–1298, 2017.

[20] Z. Shan, K. Ren, M. Blanton, and C. Wang, "Practical secure computation outsourcing: A survey," *ACM Computing Surveys (CSUR)*, vol. 51, no. 2, pp. 1–40, 2018.

[21] Z. Zhang, K. Wang, W. Lin, A. W.-C. Fu, and R. C.-W. Wong, "Practical access pattern privacy by combining PIR and oblivious shuffle," in *Proceedings of the 28th ACM International Conference on Information and Knowledge Management*, 2019, pp. 1331–1340.

[22] J. Yao, Y. Zheng, Y. Guo, and C. Wang, "Sok: A systematic study of attacks in efficient encrypted cloud data search," in *Proceedings of the 8th International Workshop*

on Security in Blockchain and Cloud Computing, 2020, pp. 14–20.

[23] S. Wu, Q. Li, G. Li, D. Yuan, X. Yuan, and C. Wang, "Servedb: Secure, verifiable, and efficient range queries on outsourced database," in *2019 IEEE 35th International Conference on Data Engineering (ICDE)*. IEEE, 2019, pp. 626–637.

[24] A. Abadi, S. Terzis, R. Metere, and C. Dong, "Efficient delegated private set intersection on outsourced private datasets," *IEEE Transactions on Dependable and Secure Computing*, vol. 16, no. 4, pp. 608–624, 2019.

[25] T. Duong, D. H. Phan, and N. Trieu, "Catalic: Delegated PSI cardinality with applications to contact tracing," in *International Conference on the Theory and Application of Cryptology and Information Security*. Springer, 2020, pp. 870–899.

[26] X. Yi, M. G. Kaosar, R. Paulet, and E. Bertino, "Single-database private information retrieval from fully homomorphic encryption," *IEEE Transactions on Knowledge and Data Engineering*, vol. 25, no. 5, pp. 1125–1134, 2012.

[27] S. Devadas, M. van Dijk, C. W. Fletcher, L. Ren, E. Shi, and D. Wichs, "Onion ORAM: A constant bandwidth blowup oblivious RAM," in *Theory of Cryptography Conference*. Springer, 2016, pp. 145–174.

[28] V. Kolesnikov and R. Kumaresan, "Improved OT extension for transferring short secrets," in *Annual Cryptology Conference*. Springer, 2013, pp. 54–70.

[29] M. J. Atallah and J. Li, "Secure outsourcing of sequence comparisons," *International Journal of Information Security*, vol. 4, no. 4, pp. 277–287, 2005.

[30] X. Li, Y. Zhu, J. Wang, and J. Zhang, "Efficient and secure multi-dimensional geometric range query over encrypted data in cloud," *Journal of Parallel and Distributed Computing*, vol. 131, pp. 44–54, 2019.

[31] X. Teng, J. Yang, J.-S. Kim, G. Trajcevski, A. Züfle, and M. A. Nascimento, "Fine-grained diversification of proximity constrained queries on road networks," in *Proceedings of the 16th International Symposium on Spatial and Temporal Databases*, 2019, pp. 51–60.

[32] Y. Zheng, R. Lu, S. Zhang, Y. Guan, J. Shao, F. Wang, and H. Zhu, "Pmrq: Achieving efficient and privacy-preserving multi-dimensional range query in ehealthcare," *IEEE Internet of Things Journal*, 2022, doi=10.1109/JIOT.2022.3158321.

[33] Y. Zheng, R. Lu, Y. Guan, J. Shao, and H. Zhu, "Towards practical and privacy-preserving multi-dimensional range query over cloud," *IEEE Trans. Dependable Secur. Comput.*, 2021.

[34] W. Li, J. Guan, and S. Zhou, "Efficiently evaluating range-constrained spatial keyword query on road networks," in *International Conference on Database Systems for Advanced Applications*. Springer, 2014, pp. 283–295.

[35] V. Kolesnikov, M. Rosulek, and N. Trieu, "SWiM: Secure wildcard pattern matching from OT extension," in *International Conference on Financial Cryptography and Data Security*. Springer, 2018, pp. 222–240.

[36] Y. Huang, D. Evans, and J. Katz, "Private set intersection: Are garbled circuits better than custom protocols?" in *NDSS*, 2012.

[37] M. Chase, E. Ghosh, and O. Poburinnaya, "Secret-shared shuffle," in *International Conference on the Theory and Application of Cryptology and Information Security*. Springer, 2020, pp. 342–372.

[38] K. Cheng, L. Wang, Y. Shen, Y. Liu, Y. Wang, and L. Zheng, "A lightweight auction framework for spectrum allocation with strong security guarantees," in *INFOCOM*. IEEE, 2020, pp. 1708–1717.

[39] T. Schneider and O. Tkachenko, "Towards efficient privacy-preserving similar sequence queries on outsourced genomic databases," in *Proceedings of the 2018 Workshop on Privacy in the Electronic Society*, 2018, pp. 71–75.

[40] D. Wang and P. Wang, "Two birds with one stone: Two-factor authentication with security beyond conventional bound," *IEEE transactions on dependable and secure computing*, vol. 15, no. 4, pp. 708–722, 2016.

[41] Q. Wang, D. Wang, C. Cheng, and D. He, "Quantum2fa: efficient quantum-resistant two-factor authentication scheme for mobile devices," *IEEE Transactions on Dependable and Secure Computing*, 2021.

[42] Z. Zhang, Y. Wang, and K. Yang, "Strong authentication without temper-resistant hardware and application to federated identities." in *NDSS*, 2020.

[43] L. Chen, K. Huang, M. Manulis, and V. Sekar, "Password-authenticated searchable encryption," *International Journal of Information Security*, vol. 20, no. 5, pp. 675–693, 2021.

[44] M. Ball, T. Malkin, and M. Rosulek, "Garbling gadgets for boolean and arithmetic circuits," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, 2016, pp. 565–577.

[45] M. J. Freedman, C. Hazay, K. Nissim, and B. Pinkas, "Efficient set intersection with simulation-based security," *Journal of Cryptology*, vol. 29, no. 1, pp. 115–155, 2016.

[46] Y. Lindell, "How to simulate it–a tutorial on the simulation proof technique," *Tutorials on the Foundations of Cryptography*, pp. 277–346, 2017.

[47] J. Yu, K. Ren, and C. Wang, "Enabling cloud storage auditing with verifiable outsourcing of key updates," *IEEE transactions on information forensics and security*, vol. 11, no. 6, pp. 1362–1375, 2016.

[48] Z. Li, D. Wang, and E. Morais, "Quantum-safe round-optimal password authentication for mobile devices," *IEEE Transactions on Dependable and Secure Computing*, 2020.

[49] Y. Lindell and B. Pinkas, "A proof of security of Yao's protocol for two-party computation," *Journal of Cryptology*, vol. 22, no. 2, pp. 161–188, 2009.

[50] D. Demmler, T. Schneider, and M. Zohner, "ABY-A framework for efficient mixed-protocol secure two-party computation." in *NDSS*, 2015.

[51] "libOTe," https://github.com/osu-crypto/libOTe.

[52] "Homo sapiens mitochondrion, complete genome," https://www.ncbi.nlm.nih.gov/nuccore/251831106.

[53] "ABY Framework," https://github.com/encryptogroup/ABY.

[54] C. Hazay, G. L. Mikkelsen, T. Rabin, T. Toft, and A. A. Nicolosi, "Efficient rsa key generation and threshold paillier in the two-party setting," *Journal of Cryptology*, vol. 32, no. 2, pp. 265–323, 2019.

Yaxi Yang received the B.E. degree from the School of Electronics and Information Engineering, Jinan University, Zhuhai, China, in 2017. She is currently working toward the Ph.D. degree with the School of Information Science and Technology, Jinan University, Guangzhou, and her research interests include applied cryptography and secure multi-party computation.

Yao Tong is an adjunct professor in the South China University of Technology, GuangZhou. Currently, she is the CEO of Guangzhou Fongwell Data Limited Company. Her research interests include big data applications.

Jian Weng received the Ph.D. degree in computer science and engineering from Shanghai Jiao Tong University, in 2008. From 2008 to 2010, he held a post-doctoral position with the School of Information Systems, Singapore Management University. He is currently a Professor and the Dean with the College of Information Science and Technology, Jinan University. His research interests include public key cryptography, cloud security, blockchain, etc. He has published over 100 papers in cryptography and security conferences and journals, such as CRYPTO, EUROCRYPT, ASIACRYPT, TCC, PKC, TPAMI, TIFS, and TDSC. He served as a PC co-chairs or PC member for more than 30 international conferences. He also serves as associate editor of IEEE Transactions on Vehicular Technology.

Yufeng Yi received the B.E. degree from the School of Medical Information Engineering, Guangdong Pharmaceutical University, Guangzhou, China, in 2018. He is currently working toward the M.E. degree with the School of Cyber Security, Jinan University, Guangzhou, and his research interests include applied cryptography and secure multi-party computation.

Yandong Zheng received her M.S. degree from the Department of Computer Science, Beihang University, China, in 2017 and she is currently pursuing her Ph.D. degree in the Faculty of Computer Science, University of New Brunswick, Canada. Her research interest includes cloud computing security, big data privacy and applied privacy.

Leo Yu Zhang (M'17) is currently a Lecturer with the School of Information Technology, Deakin University, VIC, Australia. He received the bachelor's and master's degrees in computational mathematics from Xiangtan University, Xiangtan, China, in 2009 and 2012, respectively, and the Ph.D. degree from the City University of Hong Kong, Hong Kong, in 2016. Prior to joining Deakin, he held various research positions with the City University of Hong Kong, the University of Macau, Macau, China, the University of Ferrara, Ferrara, Italy, and the University of Bologna, Bologna, Italy. His current research interests include applied cryptography and AI-related security, and he has published more than 60 refereed journal and conference articles in these fields.

Rongxing Lu (S'09-M'11-SM'15-F'21) is a University Research Scholar, an associate professor at the Faculty of Computer Science (FCS), University of New Brunswick (UNB), Canada. Before that, he worked as an assistant professor at the School of Electrical and Electronic Engineering, Nanyang Technological University (NTU), Singapore from April 2013 to August 2016. Rongxing Lu worked as a Postdoctoral Fellow at the University of Waterloo from May 2012 to April 2013. He was awarded the most prestigious "Governor General's Gold Medal", when he received his PhD degree from the Department of Electrical & Computer Engineering, University of Waterloo, Canada, in 2012; and won the 8th IEEE Communications Society (ComSoc) Asia Pacific (AP) Outstanding Young Researcher Award, in 2013. Dr. Lu is an IEEE Fellow. His research interests include applied cryptography, privacy enhancing technologies, and IoT-Big Data security and privacy. He has published extensively in his areas of expertise (with H-index 78 from Google Scholar as of Jan 2022), and was the recipient of 9 best (student) paper awards from some reputable journals and conferences. Currently, Dr. Lu serves as the Chair of IEEE ComSoc CIS-TC (Communications and Information Security Technical Committee), and the founding Co-chair of IEEE TEMS Blockchain and Distributed Ledgers Technologies Technical Committee (BDLT-TC). Dr. Lu is the Winner of 2016-17 Excellence in Teaching Award, FCS, UNB.