Balancing Security and Privacy in Genomic Range Queries

SEOYEON HWANG, ERCAN OZTURK, and GENE TSUDIK, University of California Irvine, USA

Exciting recent advances in genome sequencing, coupled with greatly reduced storage and computation costs, make genomic testing increasingly accessible to individuals. Already today, one's digitized DNA can be easily obtained from a sequencing lab and later used to conduct numerous tests by engaging with a testing facility. Due to the inherent sensitivity of genetic material and the often-proprietary nature of genomic tests, privacy is a natural and crucial issue. While genomic privacy received a great deal of attention within and outside the research community, genomic security has not been sufficiently studied. This is surprising since the usage of fake or altered genomes can have grave consequences, such as erroneous drug prescriptions and genetic test outcomes.

Unfortunately, in the genomic domain, privacy and security (as often happens) are at odds with each other. In this article, we attempt to reconcile security with privacy in genomic testing by designing a novel technique for a secure and private genomic range query protocol between a genomic testing facility and an individual user. The proposed technique ensures *authenticity* and *completeness* of user-supplied genomic material while maintaining its *privacy* by releasing only the minimum thereof. To confirm its broad usability, we show how to apply the proposed technique to a previously proposed genomic private substring matching protocol. Experiments show that the proposed technique offers good performance and is quite practical. Furthermore, we generalize the genomic range query problem to sparse integer sets and discuss potential use cases.

CCS Concepts: • **Security and privacy** → **Privacy-preserving protocols;**

Additional Key Words and Phrases: Cryptographic protocols, genomic security, genomic privacy, range query, range completeness, private substring matching

ACM Reference format:
Seoyeon Hwang, Ercan Ozturk, and Gene Tsudik. 2023. Balancing Security and Privacy in Genomic Range Queries. *ACM Trans. Priv. Sec.* 26, 3, Article 23 (March 2023), 28 pages.
https://doi.org/10.1145/3575796

1 INTRODUCTION

Dramatic recent technical advances in DNA sequencing technology [42, 49, 63] and reduced sequencing costs paved the way for ubiquitous and affordable genomic testing. As a result, genomic tests, such as paternity/parentage and pre-symptomatic disease diagnosis, that were used in the past mainly by doctors and legal authorities are becoming available to the general public.

In a typical genomic testing scenario, a testing facility ("tester"), such as 23andMe [1] or CRI Genetics [2], requests genomic data from an individual ("Alice") regarding specific locations and/or

This is an extended version of [40] (WPES 2019).
This work was supported in part by funding from the UC Irvine CORCL award.
Authors' address: S. Hwang, E. Ozturk, and G. Tsudik, University of California Irvine, Irvine, CA 92697; emails: {seoyh1, ercano}@uci.edu, gts@ics.uci.edu.

2471-2566/2023/03-ART23 $15.00
https://doi.org/10.1145/3575796

ranges on the DNA of that individual. Alice naturally wants to reveal minimal genomic data, since DNA—besides one's own highly personal and sensitive material—includes significant information about her past, current, and future relatives. The tester also needs to keep specifics (such as queried locations) secret, due to the often-proprietary nature of these genomic tests. Consequently, genomic privacy has justifiably attracted lots of attention from the research community, and numerous privacy techniques have been proposed [14, 15, 17, 19, 36, 37, 52, 67, 75].

On the other hand, genomic security, even though at least as important, received considerably less attention than genomic privacy. In particular, the authenticity and integrity of genomic data are often ignored or over-simplified, though they are crucial to accurate outcomes of genomic tests. An erroneous (whether or not maliciously caused) genomic test result can have grave health consequences when used for medical diagnoses or treatment. It could also involve social risks when used for determining familial relationships or other non-health-related traits.

At the first glance, the genomic security problem seems simple and easily solvable with traditional cryptographic primitives, such as digital signatures. However, the main challenge stems from conflicting requirements among the triad of security, privacy, and efficiency. For example, a single signature on Alice's whole genome provides security—specifically, authenticity and integrity. However, it requires Alice to send the entire genome to the tester (for signature verification), which results in zero genomic privacy for Alice and incurs significant communication costs. Another intuitive approach is to individually sign each smallest genomic unit (called a "base") and provide the tester with only those signed bases that are needed for a given test. This would offer much better privacy for Alice and incur much lower communication overhead. However, it is expensive to compute (at sequencing time) and requires the tester to verify potentially many signatures.

As an alternative to the whole genome representation, other more compact DNA representations can be used. One such example called **Single-Nucleotide Polymorphisms (SNPs)**—one-base genomic mutations—account for only about 0.1% of the entire genome. Therefore, they are significantly more efficient to use to represent a genome. However, this increase in efficiency introduces additional security problems. If we sign each SNP individually, since SNP locations are not consecutive and their positions are unpredictable (sprinkled throughout the entire genome), Alice could cheat by omitting signed SNPs from the requested range.[1] There are similar tradeoffs for other candidate representations.

Inspired by these challenges, this work focused on reconciling genomic security, privacy, and efficiency. Genomic security requires authenticity, which comprises origin authentication, integrity, and completeness. It aims to counter potentially malicious owners and/or outsiders tampering with genomic data. Privacy (against malicious testers) demands flexibility and sufficiently fine granularity controlled by the genome owner so as to reveal minimal information. At the same time, efficiency motivates minimizing genomic data processing, which complicates both privacy and security. After carefully examining security, privacy, and efficiency needs, we propose techniques based on the combination of established cryptographic tools that achieve a good balance for genomic testing. Anticipated contributions are as follows:

- We construct a secure and private range query technique that serves as a building block for various protocols and genomic representations. Range queries allow us to perform efficient genomic tests on various regions[2] that control similar functions.

[1]Of course, she cannot introduce fake SNPs.
[2]*Note:* One example of such a region is the ***Major Histocompatibility Complex (MHC)***, a large locus on vertebrate DNA, which consists of a set of genes coding for proteins responsible for detecting foreign molecules at the cellular level. MHC ranges from `6p22.1` to `6p21.3` and consists of about four megabases; see Cytogenetic location [4]. Various SNPs in this

- To demonstrate applicability of the proposed technique, we use it to improve both security and performance of prior protocols for private genomic substring matching [37].
- We prototype proposed protocols and evaluate their performance. In the course of the evaluation, we investigate various optimizations and analyze the performance of two additively homomorphic encryption schemes (ElGamal [41] and Paillier [69]) and two range-proof schemes (signature based [28] and BulletProofs [27]).
- Finally, we generalize the problem setting to sparse integer sets and discuss applications of the proposed schemes beyond genomics.

Organization: Section 2 provides an overview of the background, followed by Section 3 providing our system and security models, as well as the range completeness problem. Our main contribution is introduced in Section 4, followed by its application to the **Size- and Position-Hiding Private Substring Matching (SPH-PSM)** [37] problem in Section 5. Sections 6 and 7 discuss the implementation and evaluation of the proposed construction as well as its application to SPH-PSM. Next, Section 8 presents a generalized version of the problem with sparse integer sets and comments on its security. Then, Section 9 reviews related work, followed by some limitations of this work and future work in Section 10. Lastly, Section 11 concludes the article. All notation and acronyms used in this article are summarized in Table 1.

2 PRELIMINARIES

This section provides an overview of background material on genomics and cryptographic tools used in this work.

2.1 Genomics

The human genome is composed of around 3.2 billion[3] base pairs packed into 23 chromosomes of different size [61]. Each base can be represented by four letters of the genetic code: [A]denine, [C]ytosine, [G]uanine, and [T]hymine, where [A] always bonds with [T] and [G] always bonds with [C]. According to the **Human Genome Project (HGP)**, only around 0.1% of base pairs differ between individuals [9]. Although it is not yet possible to determine or predict exactly where these differences occur, many types of genetic variations can be used to identify an individual and determine one's susceptibility to diseases and/or sensitivity to drugs. SNPs are the most common type of genetic variation (a.k.a. mutations) among people, which represents a difference in a single base, e.g., an [A] changes to a [C] or a [G]. A variation is classified as an SNP if over 1% of a population does not carry the same nucleotide at a specific position in the DNA sequence [8].

Normally, an SNP data sequenced by, e.g., 23andMe [1], contains two base letters, one per each chromosome. For example, let the genotype of an SNP among Alice's DNA sequenced result be "AG" at position 169. This means that "A" is on one strand of one chromosome and "G" is on one strand of the other chromosome—naturally the opposite strands have paired "T" and "C," respectively, at position 169—and another person can have a different genotype at position 169, e.g., "CC." To represent the genotypes efficiently, we use an 8-bit[4] integer $\in [0, 15]$, instead of representing all two-character combinations of {A,C,G,T}. Thus, we denote a base as (pos, l), where pos is a

region (e.g., `rs9264942, rs4418214, rs2395029, and rs3131018`) have been shown to protect against **Human Immunodeficiency Virus (HIV)** [74].

[3]There are around 6 billion base pairs in a diploid human genome, whereas the reference genome in the HGP contains only around 3.2 billion base pairs. This is because most human cells contain 23 chromosomes in pairs (i.e., 46 chromosomes in total), which are almost identical. Only one of each pair, 24 chromosomes in total—22 non-sex and 2 sex (X and Y)—can represent the whole human genome information [11].

[4]Although 4-bit is enough to represent all 16 combinations, we use the standard `uint8` type to contain any insertions/deletions in future work.

Table 1. Notation and Acronyms

Notation	Description
SL	A sequencing lab converting analog genomic sample into its digital representation
T	A tester that performs genomic tests on digitized genomes
Alice	An individual requesting her digitized genome from SL and later interacts with T
$Q = [a, b]$	T's range query, where a and b are integer boundaries, $0 < a \leq b < N(\approx 3.2 * 10^9)$
Bio_{Alice}	Alice's (physical) DNA sample
$G_{Alice} = \{g_1, g_2, \ldots, g_N\}$	Alice's whole genome represented as an ordered set of tuples of the form $g_i = (i, l_i), i = 1, \ldots, N$, where i is an integer position and l_i is an integer in [0,15] representing a combination of two base letters: $\{A, G, C, T\}$ (e.g., 0 for "AA," 1 for "AC," etc.)
$\mathcal{M} = \{m_1, m_2, \ldots, m_n\}$	SNP representation of Alice's genome, where n is the number of genetic variations. Denoted by a tuple $m_i = (pos_i, l_i), i = 1, \ldots, n$, where $\mathcal{M} \subset G_{Alice}$. Note that $pos_i < pos_{i+1}$ for $\forall i$
$\mathcal{M}_Q \subseteq \mathcal{M}$	The subset of $\mathcal{M}$ including all Alice's SNPs located in a queried range Q, i.e., $\mathcal{M}_Q = \{m_i \in \hat{\mathcal{M}} \mid a \leq pos_i \leq b\}$, say $\{m_{k+1}, \ldots, m_{k+j}\}$ for some k, j
$Sign(.), Verify(.)$	A secure digital signature scheme with signing and verification protocols
$Com(x)$	A secure commitment scheme allowing zero-knowledge range proofs over x. Equivalent to $Com(x; s)$, where s is a random bit string used as a salt
NIZK$(z : R(z))$	A non-interactive zero-knowledge proof of knowledge of z such that $R(z)$ holds. Denote proof and verification protocol as NIZK_Prove(.) and NIZK_Verify(.), respectively
$Enc(.), Dec(.)$	Encryption and decryption for an additively homomorphic encryption scheme, respectively
$H(.)$	A cryptographically secure hash function
pk_X, sk_X	A pair of public key and private key of an entity X, respectively

non-negative integer $< 3.2 * 10^9$ representing the position of the base and l is a non-negative integer $\in [0, 15]$ representing two base letters from each chromosome at that position.

Due to the relatively small volume of SNPs, a reference genome model that contains only the list of mutations can be used to reduce storage and computation costs. Using a compact reference form, such as the 1,000 Genomes Project variant call format,[5] the human genome can be represented using only about 120 megabytes, while the entire (raw) representation takes up to 200 gigabytes. In this article, we focus on these two representations: the *whole genome* and the compact *SNP* based. We defer other genomic representation formats, e.g., **Short Tandem Repeat (STR)** and **Restriction Fragment Length Polymorphism (RFLP)**, to future work.

2.2 Cryptographic Commitments

A commitment scheme is a cryptographic primitive that involves a prover and a verifier. The prover first commits to a chosen (secret) value and reveals it later to the verifier. A commitment scheme thus has two phases. In the first, *commit*, phase, the prover sends a message (*commitment*) to the verifier committing to a secret value. The commitment must reveal no information about the committed value; this is called the *hiding* property. In the second, *reveal*, phase, the prover sends to the verifier a message (*decommitment*) that reveals the previously committed value. The verifier validates the revealed value against the committed one. The former must uniquely match the latter; this is called the *binding* property. We denote a commitment using $Com(z; r)$, where Com is a commitment scheme, z is the committed value, and r is a random bit-string, used as a salt.

In this article, we use two types of commitments: one based on the **discrete logarithm problem (DLP)** and the other based on a strong cryptographic hash function. For DLP-based commitments, we use the well-known Pedersen [71] and Fujisaki-Okamoto [44] commitment schemes.

A Pedersen commitment is defined in a subgroup of $\mathbb{Z}_p^*$, where p is a large prime, albeit any group where the DLP is hard can be used. Let $\mathbb{G}$ be the group of order p, generated by element g, i.e., $\mathbb{G} = \langle g \rangle$ with $o(G) = p$. Two non-identity elements, g and h, in $\mathbb{G}$ are used as public key, where $log_g h$ is not known to either the prover or the verifier. To commit to a message $z \in \mathbb{Z}_p$, a

[5]See www.internationalgenome.org/wiki/Analysis/vcf4.0.

Pedersen commitment is computed as $Com = Com(z;r) = g^z h^r$, where r is a random number in $\mathbb{Z}_p$. To decommit, later a prover reveals z and r and a verifier checks if indeed $g^z h^r = Com$.

A Fujisaki-Okamoto commitment is an extension of the Pedersen commitment to the RSA setting. Instead of Z_P^*, it uses Z_N^*, where $N = PQ$ and P, Q are distinct primes, such that $P = 2p+1$ and $Q = 2q + 1$, where p and q are also primes. Now, two generators g_p and g_q generate each group, $\mathbb{G}_p$ and $\mathbb{G}_q$, of prime orders p and q, respectively; i.e., $\mathbb{G}_p$ and $\mathbb{G}_q$ are subgroups of $\mathbb{Z}_P^*$ and $\mathbb{Z}_Q^*$, respectively. Generator g of the group $\mathbb{G}_{pq}$ is computed using the Chinese Remainder Theorem, such that $g = g_p \pmod{P}$ and $g = g_q \pmod{Q}$. Then, h is computed by $h = g^\alpha \pmod{N}$, where α is uniformly chosen from $\mathbb{Z}_{pq}^*$, and (g, h) is set as the public key. To commit a message $z \in \mathbb{Z}_N$, the Fujisaki-Okamoto commitment is computed by $Com(z;r) = g^z h^r \pmod{N}$, where r is a random number in $\mathbb{Z}_{\lambda N}^*$ with a security parameter λ. Both Pederson and Fujisaki-Pkamoto commitment schemes are statistically hiding and computationally binding.

Finally, the commitments based on secure cryptographic hash functions rely on the fact that modern hash functions practically reveal no information about the value they *hide* if they are used with a sufficiently long random salt. In addition, since secure cryptographic hash functions offer weak and/or strong collision resistance properties, the commitments based on such functions are *binding*. A commitment is of the form $H(z||r)$, where H is a cryptographically secure hash function (e.g., SHA2 [12]), z is the committed value, and r is a sufficiently long random bit-string.

2.3 Zero-Knowledge Range Proofs

A **Zero-knowledge Range Proof (ZKRP)** allows a prover to convince a verifier that a committed secret value is within a given range without revealing that value. The three standard **zero-knowledge proof (ZKP)** properties, *completeness*, *soundness*, and *zero-knowledge*, also hold for ZKRPs. That is, when the secret is in the given range, an honest prover can always convince an honest verifier of the fact, which yields *completeness*. Meanwhile, no dishonest prover can convince an honest verifier if the secret is not in the range, which yields *soundness*. Also, the verifier learns nothing from the execution of the protocol about the secret value other than that it lies in the range, which corresponds to *zero-knowledge*. ***Non-interactive Zero-knowledge Proofs (NIZKs)*** are a class of ZKP that requires no interaction between the prover and the verifier. It is well known that NIZK can be constructed from ZKP in a random oracle model using the Fiat-Shamir heuristic [43].

Boudot's range proof [24] is the first practical construction of ZKRP proposed in 2001. It includes two protocols: one for the extended range and the other for the exact range. In the first, for a requested range $[a, b]$, the prover shows that the secret integer v resides in $[a - \theta, b + \theta]$, where $\theta = 2^{t+l+1}\sqrt{b-a}$ and t and l are security parameters, i.e., with the expansion rate[6] $\delta = 2\theta$. In the second protocol, the prover shows that v resides in the exact requested range, $[a, b]$, i.e., with $\delta = 1$.

Following Boudot, there has been a long line of work on ZKRPs. As mentioned in survey papers [38, 65], ZKRP methods can be classified based on their main characteristics: (1) *square decomposition*, (2) *signature based*, (3) *multi-based decomposition*, and (4) *Bulletproofs*.

Square decomposition constructions, such as [24, 46, 57], use the fact that any non-negative integer can be represented by the sum of squares. *Signature-based constructions* [28] are generalized from a zero-knowledge set membership test, by showing that the prover knows a signature on the secret, among the entire set of signatures on integers in the given range. In *multi-based decomposition constructions* [29, 58], the secret is decomposed by a u-ary (usually binary) representation and the

[6]The expansion rate of a range proof allows for tolerance, defined as $\delta = \frac{B-A}{b-a}$, where $[a, b]$ is the requested range and $[A, B]$ is a larger range including $[a, b]$ wherein the prover shows the value resides. Note that if this rate is 1, the range proof convinces a verifier that the value is exactly in the requested range $[a, b]$.

prover shows that each coefficient is 0 or 1, which proves that the secret value is in a given range. Lastly, *Bulletproofs* [27] is a technique without a trusted setup phase, unlike other approaches. It uses the binary representation of the secret value and inner product proofs, which lend themselves to smaller proof sizes.

Any ZKRP protocol can be used in our construction. For example, we use Boudot's range proof [24] as the ZKRP of our main construction (Section 4) and Bulletproofs [27] in the extended version (Section 5). We briefly summarize these protocols below.

Assume that the prover wants to show that a secret value $v \in [a, b]$ is in $[a - \theta, b + \theta]$, where θ is defined as above. To do so, the prover shows (1) $v - a \geq -\theta$ and (2) $b - v \geq -\theta$. To demonstrate (1), the prover writes $(v - a)$ as the sum of the greatest square less than v and a non-negative number, i.e., $v - a = v_1^2 + p$, where $v_1 := \lfloor \sqrt{v - a} \rfloor$ and $p = (v - a) - v_1^2$. Next, the prover shows, in zero-knowledge, that the commitment to v_1^2 hides a square and the commitment to p hides a number with the absolute value less than θ, using the method in [30]. The same procedure is done for (2), but with r_1', r_2' such that $r_1' + r_2' = -r$. As a result, the prover shows that $v \in [a - \theta, b + \theta]$, where $\theta = 2^{t+l+1}\sqrt{b - a}$.

Showing that the secret value v is exactly in $[a, b]$ is similar. However, it is now shown that the expanded secret value lies in the expanded range of $[a, b]$, which implies that $v \in [a, b]$. More specifically, the prover first expands v by computing $v' = v \cdot 2^T$, where $T = 2(t + l + 1) + |b - a|$. Then it shows that $v' \in [2^T a - \theta', 2^T b + \theta']$, where $\theta' = 2^{t+l+T/2}\sqrt{b - a}$ using the previous protocol with expanded commitment $Com(z; r)^{2^T}$. Since $\theta' < 2^T$, the verifier is convinced that $v' \in]2^T a - 2^T, 2^T b + 2^T[\iff v \in]a - 1, b + 1[$ so that $v \in [a, b]$ as $v \in \mathbb{Z}$.

With Bulletproofs [27], the prover performs ZKRP twice: once for $v \in [a, a + 2^n]$ and then for $v \in [b - 2^n, b]$, in order to show that $v \in [a, b]$. To show that $v \in [0, 2^n - 1]$, the prover first vectorizes v to n-bit value, $v_L := (v_1, \ldots, v_n) \in \{0, 1\}^n$. Then, for $v_R := v_L - 1^n$, the Hadamard product of v_L and v_R is zero, i.e., $v_L \circ v_R = 0$. To show these properties, the prover needs to show that:

(1) the inner product $\langle v_L, 2^n \rangle = v$, (2) $\langle v_L, v_R \circ y \rangle = 0$, and (3) $\langle v_L - 1^n - v_R, y^n \rangle = 0$, for $\forall y \in \mathbb{Z}_p$.

These equalities can be combined into one, by the verifier choosing a random $z \in \mathbb{Z}_p$ and prover showing that

$$\langle v_L - z \cdot 1^n, y^n \circ (v_R + z \cdot 1^n) + z^2 \cdot 2^n \rangle = z^2 \cdot v + \delta(y, z), \tag{1}$$

where $\delta(y, z) = (z - z^2) \cdot \langle 1^n, y^n \rangle - z^3 \langle 1^n, 2^n \rangle \in \mathbb{Z}_p$. Also, to hide the information about v_L and v_R, additional blinding terms $s_L, s_R \in \mathbb{Z}_p^n$ are used, so that the prover sends l, r, and t instead, where:

$$\mathbf{l} := l(X) = v_L - z \cdot 1^n + s_L \cdot X, \mathbf{r} := r(X) = y^n \circ (v_R + z \cdot 1^n + s_R \cdot X) + z^2 \cdot 2^n \in \mathbb{Z}_p^n[X],$$
$$\text{and } t(X) := \langle l(X), r(X) \rangle \in \mathbb{Z}_p[X].$$

To convince the verifier that the constant term t_0 of $t(X) = t_0 + t_1 \cdot X + t_2 \cdot X^2$ becomes the right-hand side of Equation (1), i.e., $t_0 = v \cdot z^2 + \delta(y, z)$, the prover commits to the remaining coefficients, $t_1, t_2 \in \mathbb{Z}_p$; receives a random point $x \in \mathbb{Z}_p^*$ from the verifier; and replies the $t(x)$ value to the verifier. By checking all the commitments and values, the verifier is convinced that $v \in [0, 2^n - 1]$.

2.4 Homomorphic Encryption

Homomorphic encryption (HE) is a special type of encryption that allows users to perform certain arithmetic operations on encrypted data such that results are reflected in the plaintext. If it supports *both* unlimited addition and multiplication of ciphertexts, the HE scheme is called ***Fully Homomorphic Encryption (FHE)***. A scheme that supports a limited number of operations of either type is called ***Somewhat Homomorphic Encryption (SWHE)***. Finally, a scheme that supports only one operation type (e.g., addition or multiplication) is called ***Partially Homomorphic Encryption (PHE)***. We use PHE in this work. (Typically, in terms of computation costs, $FHE > SWHE > PHE$.)

There are many PHE schemes, e.g., [32–34, 41, 45, 53, 66, 68, 69, 73]. For example, the well-known ElGamal encryption scheme [41] is a PHE that supports multiplication, as for any $m_1, m_2 \in \langle g \rangle$ and random r_1, r_2,

$$Enc(m_1) * Enc(m_2) = (g^{r_1}, m_1 * h^{r_1}) * (g^{r_2}, m_2 * h^{r_2}) = (g^{r_1+r_2}, m_1 * m_2 * h^{r_1+r_2}) = Enc(m_1 m_2).$$

Another variant of ElGamal [33] is additively homomorphic. In it, g^m instead of m is used as the ciphertext. i.e.,

$$Enc(m_1) * Enc(m_2) = (g^{r_1}, g^{m_1} * h^{r_1}) * (g^{r_2}, g^{m_2} * h^{r_2}) = (g^{r_1+r_2}, g^{m_1+m_2} * h^{r_1+r_2}) = Enc(m_1 + m_2).$$

Another popular PHE is Paillier [69], which also supports addition. In it,

$$\begin{aligned} Enc(m_1) * Enc(m_2) &= (g^{m_1} r_1^n \pmod{n^2}) * (g^{m_2} r_2^n \pmod{n^2}) = g^{m_1+m_2}(r_1 + r_2)^n \pmod{n^2} \\ &= Enc(m_1 + m_2), \end{aligned}$$

where r_1, r_2 are randomly chosen elements in $\mathbb{Z}_n^*$, for any $m_1, m_2 \in \mathbb{Z}_n$. Although Paillier is widely used, we show in Section 7 that the **additively homomorphic ElGamal (AH-ElGamal)** scheme is more efficient in our context. (This is mainly because we only need to check if the ciphertext is the encryption of zero.)

3 SYSTEM AND SECURITY MODELS

3.1 System Model

The system model includes three types of entities: (1) individuals, (2) sequencing labs, and (3) testers, where each entity's role is as follows:

(1) Each individual provides his/her DNA sample to a sequencing lab.
(2) The sequencing lab is a service provider, certified by a trusted authority, that extracts and generates the digitized genomic data from the received DNA sample, e.g., hospitals and **Direct-to-Consumer (DTC)** service providers.
(3) The tester offers various genomic tests, e.g., paternity, pharmacogenetics, or cancer marker screenings, which entail one or more queries for genomic data on some specific locations required for a test. The exact locations for each test are certified by a trusted authority. Each query represents a range $Q = [a, b]$ with genomic (integer) positions a and b, which indicates that the tester needs all genomic data in Q to perform the test.

We assume a global pre-existing **Public Key Infrastructure (PKI)** for establishing trust among entities based on certified public keys. Although there would be a multitude of each entity type, we assume (for the sake of clarity) only one individual (*Alice*), one sequencing lab (*SL*), and one tester (*T*).

3.2 Security Model

We assume that *SL* and *T* are certified by a trusted authority and *SL* is fully trusted by both Alice and *T*. Sequencing and preparing digitized genomic data by *SL* are performed offline. We assume *T* is **Honest but Curious (HbC)**: although it faithfully follows the protocol, it aims to learn more information about Alice's genome than is required for the test. The ranges queried by *T* are considered to be pre-approved by a trusted authority. For instance, the trusted authority provides *T* a signed white-list of legitimate ranges or genome positions for all authorized genomic tests. For this reason, Alice is willing to reveal the required genomic data to *T* for the given test.

We assume that T does not trust Alice, since she might supply altered genomic data (whether by modification or omission) in order to influence the outcome of a test. For correct test results, T needs to ensure that all information Alice supplies is both *authentic* and *complete*. To this end, our goal is a **secure and private range query protocol, π,** between Alice and T. π takes T's range query Q and Alice's set of SNPs $\mathcal{M}$ as inputs and outputs the response $\mathcal{M}_Q$ to T, i.e., the set of all SNPs in $\mathcal{M}$ located in Q. Concrete security goals are:

(1) **Authenticity:** All SNPs reported by Alice must be authentic; i.e., Alice cannot modify any part of her digitized genome or introduce new parts without being detected.
(2) **Completeness:** All SNPs in Q should be reported; i.e., Alice should not omit any SNP in Q from her reply to T's query.
(3) **Alice's Privacy:** T should not learn any information about Alice's genome beyond SNPs in Q.

`Caveat:` Nucleotides (individual bases) at different positions can be correlated. Some such correlations are well known. Therefore, based on the specific mutations in a given range that Alice reveals to T as part of a legitimate test, T could infer genomic information from other regions of Alice's genome. We believe that such side-channel inference is unavoidable and consider it out of the scope of this work.

4 GENOMIC RANGE QUERY PROTOCOL

We now discuss several intuitive ways of supporting genomic range queries. Then, we describe the proposed technique and analyze its security, privacy, and efficiency.

4.1 Intuitive Approaches

One trivial approach is to use Alice's whole genome, G_{Alice}, and let SL sign all g_i's at sequencing time. When T asks for all mutations in Q, Alice provides all pairs within that range along with their signatures. T can easily detect if anything is missing since it receives all g_i's. Inclusion of fake bases is impossible, as Alice cannot forge SL's signatures. This approach provides authenticity and integrity, and leaks no information about bases outside the queried range. However, it has high computation and storage costs because every single base in Q needs to be signed, sent, and verified. To reduce costs, certain cryptographic methods, such as condensed and aggregated signatures, can be employed, albeit the final cost would be still far from optimal. We refer to [25] for a detailed discussion and comparison of such methods.

Another intuitive approach is to use Alice's SNP-represented genome, $\mathcal{M}$, and let SL sign all m_i's at sequencing time. This would substantially reduce storage and computation costs due to the relatively small volume of SNPs. However, it introduces the completeness problem, since nothing prevents Alice from omitting some (or all) SNPs when she sends T the mutations and signatures on Q. To prevent any omissions and ensure correct ordering of mutations, SL can sign tuples consisting of two adjacent mutations sorted in ascending order, as suggested by [39]. However, this entails revealing two tuples that contain mutations in positions immediately beyond both lower- and upper-range boundaries, respectively. Due to the general sparsity of genomic mutations, this could leak a substantial amount of sensitive information to T.

4.2 Proposed Approach

Assume that Alice gives her physical DNA sample (Bio_{Alice}) to SL to obtain a digital representation of her genome in the form of SNPs. SL forms the SNP representation $\mathcal{M} = \{m_1, m_2, \ldots, m_n\}$ by comparing Alice's genome to a reference genome, as discussed in Section 2.1. Now, SL adds two special mutations, m_0 and m_{n+1}, to represent the lower and upper genome boundaries. At the end of the sequencing process, Alice receives the list:

$$\hat{\mathcal{M}} = \{m_0, m_1, m_2, \ldots, m_n, m_{n+1}\} = \mathcal{M} \cup \{m_0, m_{n+1}\}.$$

For the special sentinel mutations m_0 and m_{n+1}, positions pos_0 and pos_{n+1} are integers out of the normal human genomic position range, i.e., $pos_0 = -\infty < 0, pos_{n+1} = +\infty > N$, and the base letters l_0 and l_{n+1} are dummy letters of the same sizes.

Alice also receives the lists of signatures, $\Gamma = \{\gamma_0, \gamma_1, \ldots, \gamma_n\}$, and salts, $Salt$, used to generate these signatures. To show the ordering of mutations without revealing additional information on boundary ones, SL signs *the tuple of commitments* for each adjacent mutation, instead of signing each mutation, i.e., $\gamma_i = Sign(sk_{SL}, Tup_i)$, for $\forall i$. Each tuple consists of four commitments for the adjacent mutations and their positions:

$$Tup_i = (Com(pos_i; s_{i,1}), Com(m_i; s_{i,2}), Com(pos_{i+1}; s_{i+1,1}), Com(m_{i+1}; s_{i+1,2}))$$

for some commitment scheme Com and salts $s_{i,j}$ for the ith SNP and $j = 1, 2$. The latter two commitments for pos_{i+1} and m_{i+1} are reused in the next tuple, Tup_{i+1}. This can be done in the offline phase between SL and Alice.

In the online phase, Alice returns the mutations within the queried range Q, along with the corresponding signatures and salts. She also generates two range proofs for the positions of the first mutations outside the queried range: one for the mutation with the highest position below the lower bound of Q and the other for the mutation with the lowest position above the upper bound of Q. Alice sends these proofs and corresponding commitments of those positions. Denoting all SNPs in Q as $\mathcal{M}_Q = \{m_{k+1}, \ldots, m_{k+j}\}$, Alice sends $(\mathcal{M}_Q, Salt_Q, \Gamma_Q, C_k, C_{k+j+1}, l, h)$ to T, where

$$\begin{aligned}
Salt_Q &= \{s_{k+1}, \ldots, s_{k+j}\} = \{(s_{k+1,1}, s_{k+1,2}), \ldots, (s_{k+j,1}, s_{k+j,2})\},\\
\Gamma_Q &= \{\gamma_k, \gamma_{k+1}, \ldots, \gamma_{k+j}\},\\
C_k &= (Com(pos_k, s_{k,1}), Com(m_k, s_{k,2})),\\
C_{k+j+1} &= (Com(pos_{k+j+1}, s_{k+j+1,1}), Com(m_{k+j+1}, s_{k+j+1,2})),\\
l &\leftarrow \texttt{NIZKRP\_Prove}\{(pos_k, s_{k,1}) \mid C_{k,1} = Com(pos_k, s_{k,1}) \wedge pos_k < a\}\\
h &\leftarrow \texttt{NIZKRP\_Prove}\{(pos_{k+j+1}, s_{k+j+1,1}) \mid C_{k+j+1,1} = Com(pos_{k+j+1}, s_{k+j+1,1}) \wedge pos_{k+j+1} > b\},
\end{aligned}$$

where l and h are the proofs for the **non-interactive zero-knowledge proof of range proof (NIZKRP)**. T reconstructs intermediate tuples using received mutations and salts, and boundary tuples using received C_k and C_{k+j+1}, as follows:

$$\begin{aligned}
Tup_i &:= (Com(pos_i; s_{i,1}), Com(m_i; s_{i,2}), Com(pos_{i+1}; s_{i+1,1}), Com(m_{i+1}; s_{i+1,2}))\\
&\quad \text{for } i = k+1, \ldots, k+j-1,\\
Tup_k &= (C_k, Com(pos_{k+1}; s_{k+1,1}), Com(m_{k+1}; s_{k+1,2})),\\
Tup_{k+j} &= (Com(pos_{k+j}; s_{k+j,1}), Com(m_{k+j}; s_{k+j,2}), C_{k+j+1}).
\end{aligned}$$

Then, T verifies the signatures using SL's public key and NIZKRP proofs l, h with the boundaries of Q, a, and b; i.e., T sees if (1) $Verify(pk_{SL}, Tup_i, \gamma_i) = 1$ for all $i = k, \ldots, k+j$; (2) $\texttt{NIZKRP\_Verify}(C_{k,1}, l) = 1$; and (3) $\texttt{NIZKRP\_Verify}(C_{k+j+1,1}, h) = 1$, and it aborts if any of those fails. Otherwise, it proceeds with the test using received $\mathcal{M}_Q$. Figures 1 and 2 show offline and online phases, respectively. Communication and computation costs are reflected in Table 2.

4.3 Security Analysis

Assume a non-empty $\mathcal{M}_Q = \{m_{k+1}, \ldots, m_{k+j}\}$ for some non-negative integers k and j. T checks the authenticity of $m_i \in \mathcal{M}_Q$ by verifying the received signatures $\Gamma_Q = \{\gamma_i\}_{i=k}^{k+j}$ using the

Offline Phase of Secure & Private Genomic Range Query Protocol

SL **Alice**

$\mathcal{M} := \{m_1, ..., m_n\} \leftarrow Seq(Bio_{Alice}),$ where $m_i = (pos_i, l_i)$ $\xleftarrow{\quad Bio_{Alice} \quad}$

$\hat{\mathcal{M}} := \mathcal{M} \cup \{m_0, m_{n+1}\},$ where $m_0 = (-inf, B), m_{n+1} = (+inf,\ B)$

with $-\ inf < 0, +inf > N,$ and a bogus letter B

$Salt := \{s_0, ..., s_{n+1}\},$ where $s_i = (s_{i,1}, s_{i,2}) \leftarrow \$$

$\Gamma := \{\gamma_0, ..., \gamma_n\},$ where $\gamma_i = Sig(sk_{SL}, Tup_i),$

$Tup_i := (Com(pos_i; s_{i,1}), Com(m_i; s_{i,2}), Com(pos_{i+1}; s_{i+1,1}), Com(m_{i+1}; s_{i+1,2}))$

$\xrightarrow{\quad \hat{\mathcal{M}}, Salt, \Gamma \quad}$

Fig. 1. (Offline phase) digitizing Alice's SNPs.

Online Phase of Secure & Private Genomic Range Query Protocol

Alice ***T***

$\hat{\mathcal{M}} = \{m_0, ..., m_{n+1}\}, Salt = \{s_0, ..., s_{n+1}\}, \Gamma = \{\gamma_0, ..., \gamma_n\},$ $Q = [a, b]$

$\xleftarrow{\quad Q \quad}$

Choose $\mathcal{M}_Q, Salt_Q, \Gamma_Q,$ say $\mathcal{M}_Q = \{m_{k+1}, ...,\ m_{k+j}\}$

$Salt_Q := \{s_{k+1}, ..., s_{k+j}\},$ and $\Gamma_Q = \{\gamma_k, ...,\ \gamma_{k+j}\}$

Compute $C_k, C_{k+j+1},$ where $C_i = (C_{i,1}, C_{i,2}),$

$C_{i,1} = Com(pos_i; s_{i,1}), C_{i,2} = Com(m_i; s_{i,2})$

$l \leftarrow$ `NIZKRP_Prove`$\{(pos_k, s_{k,1}) \mid C_k = Com(pos_k; s_{k,1}) \wedge pos_k < a\}$

$h \leftarrow$ `NIZKRP_Prove`$\{(pos_{k+j+1}, s_{k+j+1,1}) \mid C_{k+j+1} = Com(pos_{k+j+1}; s_{k+j+1,1}) \wedge pos_{k+j+1} > b\}$

$\xrightarrow{\quad \mathcal{M}_Q, Salt_Q, \Gamma_Q, (C_k, C_{k+j+1}), l, h \quad}$

Compute $\{Tup_i\}_{i=k+1}^{k+j-1}$, using $\mathcal{M}_Q$ and $Salt_Q$, and

$Tup_k = (C_k, Com(pos_{k+1}; s_{k+1,1}), Com(m_{k+1}; s_{k+1,2}))$

$Tup_{k+j} = (Com(pos_{k+j}; s_{k+j,1}), Com(m_{k+j}; s_{k+j,2}), C_{k+j+1})$

Check if :

1.$Verify(pk_{SL}, Tup_i, \gamma_i) = 1$ for all $i = k, ..., k+j$

2.`NIZKRP_Verify`$(C_k, l) = 1$

3.`NIZKRP_Verify`$(C_{k+j+1}, h) = 1$

Abort, if not. (Otherwise, perform testing)

Fig. 2. (Online phase) genomic range query between Alice and Tester (T).

reconstructed tuples $\{Tup_i\}_{i=k}^{k+j}$. The links between two adjacent tuples prevent Alice from excluding any mutations. Also, T ensures completeness by verifying the two NIZKRP proofs, which shows that the commitments for the boundary positions hide the integers beyond the queried range. This allows Alice to maintain the privacy of all mutations outside Q.

Now, suppose that $\mathcal{M}_Q$ is empty; i.e., Alice has no SNPs within Q. Then, Alice sends one tuple Tup_l for some l, of the form $(Com(pos_l), Com(m_l), Com(pos_{l+1}), Com(m_{l+1}))$ such that $pos_l < a$ and $pos_{l+1} > b$ and its signature γ_l to T. T verifies γ_l, which satisfies authenticity, and checks the ZKPs that committed values are outside Q, ensuring completeness and Alice's privacy.

One special case occurs when there are no mutations before position a and/or after position b. The required range is large enough and it reveals all SNPs to conduct the test with $m_k = m_0$ and/or $m_{k+j+1} = m_{n+1}$. Since sentinel commitments are indistinguishable from other commitments, our security goals are also achieved in this case. In summary:

Goal 1. Authenticity is based on the security of the underlying digital signature scheme used by SL to sign tuples.

Table 2. Communication and Computation Costs, Where n = (Number of Alice's SNPs) and j = (Number of Mutations in Q)

(From → To)	**Communication Cost**
$SL \rightarrow$ Alice	$(n+2)$ mutations, $(n+1)$ signatures, and $2(n+1)$ salts
$T \rightarrow$ Alice	two integers, a and b, denoting the range $Q = [a, b]$
Alice $\rightarrow T$	j mutations, $2j$ salts, $(j+1)$ signatures, 4 commitments, 2 range proofs

Entity	**Computation Cost**
SL (Offline)	$2(n+2)$ commitments, $(n+1)$ signatures
T	$4j$ commitments, $(j+1)$ signature verifications, 2 range proof verifications
Alice	4 commitments, 2 range proof generations

Table 3. Comparisons of SPH-PSM Variants

	Genomic Representation	**Security (for T)**	**Privacy**	
			Alice	T
SPH-PSM [37]	Whole	✗	✓	✓
S-SPH-PSM (§5.2)	Whole	✓	✓	✓
ES-SPH-PSM (§5.3)	SNP	✓	✓	✓
FES-SPH-PSM (§5.4)	SNP	✓	✓	▲

✓ and ✗ denotes supported and unsupported, respectively, and ▲ denotes degree of support can vary.

Goal 2. Completeness is achieved by sequential linking of elements, allowing T to detect any omissions.

Goal 3. Alice's Privacy holds due to the use of ZKP and the *hiding* property of commitments, which reveals no information about either mutations' positions or mutations outside Q.

5 OTHER APPLICATIONS

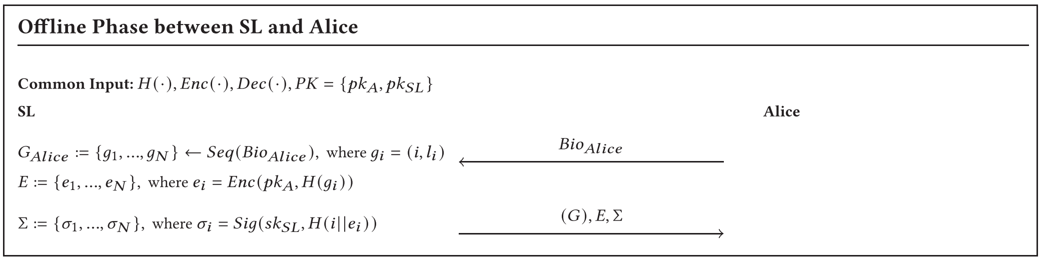
In this section, we show how to improve a private substring matching protocol for genomic data using our technique. SPH-PSM [37] operates on encrypted bases and allows the genome owner (Alice) to learn whether a test pattern (a list of contiguous bases on some specific locations required for a given test) held by a tester (T) exists in her genome, while not revealing anything about their inputs to each other. Though this protocol provides privacy for Alice's genome and T's pattern, it guarantees neither authenticity nor integrity of Alice's genome. Furthermore, SPH-PSM incurs high communication and computational costs, since it requires Alice to encrypt her whole genome and send the entire encrypted genome to T.

We denote our proposed protocols variants with the following acronyms: ***secure SPH-PSM (S-SPH-PSM)***, ***efficient and secure SPH-PSM (ES-SPH-PSM)***, and ***flexible, efficient, and secure SPH-PSM (FES-SPH-PSM)***, respectively. Table 3 provides a high-level comparison of these variants over SPH-PSM.

5.1 Size- and Position-hiding Private Substring Matching Protocol (SPH-PSM) [37]

First, Alice generates a public-private key-pair for an AHE scheme and encrypts each base of her genome using the public key. T computes the additive inverse of each base in its specific test pattern and encrypts each inverse using Alice's public key.

In the online phase, Alice sends the entire encrypted genome to T. For each position where the pattern locates, T homomorphically adds its encrypted pattern to Alice's encrypted base. Then, T adds the resulting values and returns the final (encrypted) sum to Alice. Alice decrypts received



Offline Phase between SL and Alice

Common Input: $H(\cdot), Enc(\cdot), Dec(\cdot), PK = \{pk_A, pk_{SL}\}$

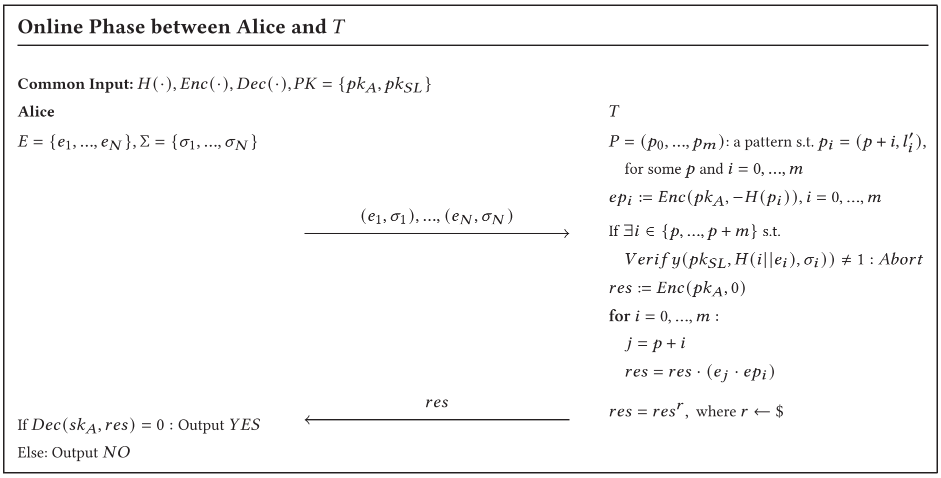
SL | **Alice**

$G_{Alice} := \{g_1, ..., g_N\} \leftarrow Seq(Bio_{Alice})$, where $g_i = (i, l_i)$ $\quad \xleftarrow{Bio_{Alice}}$

$E := \{e_1, ..., e_N\}$, where $e_i = Enc(pk_A, H(g_i))$

$\Sigma := \{\sigma_1, ..., \sigma_N\}$, where $\sigma_i = Sig(sk_{SL}, H(i||e_i))$ $\quad \xrightarrow{(G), E, \Sigma}$

Fig. 3. Offline phase of secure SPH-PSM (S-PSH-PSM).



Online Phase between Alice and T

Common Input: $H(\cdot), Enc(\cdot), Dec(\cdot), PK = \{pk_A, pk_{SL}\}$

Alice | T

Alice: $E = \{e_1, ..., e_N\}, \Sigma = \{\sigma_1, ..., \sigma_N\}$

T: $P = (p_0, ..., p_m)$: a pattern s.t. $p_i = (p + i, l_i')$, for some p and $i = 0, ..., m$

$ep_i := Enc(pk_A, -H(p_i)), i = 0, ..., m$

$\xrightarrow{(e_1, \sigma_1), ..., (e_N, \sigma_N)}$

```
If $\exists i \in \{p, ..., p+m\}$ s.t.
    $Verify(pk_{SL}, H(i||e_i), \sigma_i)) \neq 1 : Abort$
$res := Enc(pk_A, 0)$
for $i = 0, ..., m$ :
    $j = p + i$
    $res = res \cdot (e_j \cdot ep_i)$
$res = res^r$, where $r \leftarrow \$$
```

$\xleftarrow{res}$

Alice: If $Dec(sk_A, res) = 0$: Output YES

Else: Output NO

Fig. 4. Online phase of secure SPH-PSM (S-SPH-PSM).

ciphertext using her private key and learns the test result. The decrypted result is 0 if Alice's genome matches the test pattern; it is a random value otherwise. During the whole process, T does not learn any information on Alice's genome or the test result.

5.2 Secure SPH-PSM (S-SPH-PSM)

In SPH-PSM, Alice can modify her digitized genome and influence the test result, since SPH-PSM does not guarantee authenticity or integrity of Alice's genome. To prevent this, we design *secure SPH-PSM (S-SPH-PSM)* and let SL act as a certification authority for the genomic data.

When SL sequences Alice's DNA sample (Bio_{Alice}), it encrypts the hash of each base using an AHE scheme, under Alice's public key.[7] Then, it signs the hash of each ciphertext—along with its position—using its private key and sends the list of ciphertexts and corresponding signatures to Alice.

The online phase is similar to SPH-PSM, except that T verifies the signatures and checks authenticity and integrity of ciphertexts on the positions required for the test. Figures 3 and 4 show the offline and online phases of S-SPH-PSM, respectively.

[7]The choice of this public key depends on who will learn the test result at the end of the protocol. For instance, for court-mandated tests, the court's public key can be used instead.

Offline Phase between SL and Alice

Common Input: $H(\cdot), E(\cdot), D(\cdot), PK = \{pk_A, pk_{SL}\}$

SL		**Alice**
$\mathcal{M} := \{m_1, ..., m_n\} \leftarrow Seq(Bio_{Alice})$, where $m_i = (pos_i, l_i)$	$\xleftarrow{Bio_{Alice}}$	
$\hat{\mathcal{M}} := \mathcal{M} \cup \{m_0, m_{n+1}\}$, where $pos_0 < 0, pos_{n+1} > N$		
$E := \{e_0, ..., e_{n+1}\}$, where $e_i = Enc(pk_A, H(m_i))$		
$Salt := \{s_0, ..., s_{n+1}\}$, where $s_i \leftarrow \$$		
$\Gamma := \{\gamma_0, ..., \gamma_n\}$, where $\gamma_i = Sign(sk_{SL}, H(Tup_i))$,		
$Tup_i := (Com(pos_i; s_i), e_i, Com(pos_{i+1}; s_{i+1}), e_{i+1})$	$\xrightarrow{(\hat{\mathcal{M}}), E, Salt, \Gamma}$	$\mathcal{M}' := \{m'_0, ..., m'_{n+1}\}$,
		where $m'_i := Dec(sk_A, e_i) = (pos'_i, l'_i)$
		$C := \{c_0, ..., c_{n+1}\}$,
		where $c_i := Com(pos'_i; s_i), s_i \in Salt$

Fig. 5. Offline phase of efficient and secure SPH-PSM (ES-SPH-PSM) and flexible, efficient, and secure SPH-PSM (FES-SPH-PSM).

S-SPH-PSM also ensures authenticity and completeness via signing of both ciphertext and its position. T can detect omissions or rearrangements, as it verifies signatures for all consecutive positions in the range. Also, to prevent accidental matches, we hash both the position and the base letter when encrypting the bases, as in the AH-ElGamal-based protocol [37].

5.3 Efficient and Secure SPH-PSM (ES-SPH-PSM)

We design *efficient and secure SPH-PSM protocol (ES-SPH-PSM)* to improve efficiency. We use SNPs instead of the whole genome representation and our techniques proposed in Section 4. Offline and online phases of this protocol are given in Figures 5 and 6, respectively. In the former, we add special sentinel SNPs and sign the tuples of adjacent bases, as in Section 4, whereas the tuple consists of commitments of position and the ciphertext of each SNP instead. SL sends to Alice the encrypted genomic data and signatures along with the salts used for the commitments. In the online phase, T verifies "all" received tuples and computes multiple results for all possible starting positions. The computational complexity of this approach is $O(nm)$, where n is the size of Alice's genome and m is the number of bases in the pattern that T holds. This is because SNP positions are hidden, unlike in the whole genome representation. (See unoptimized commented out pseudo-code in Figure 6.)

However, this computational cost can be significantly improved and reduced to $O(n)$ with some optimizations. First, T performs an initial calculation as before by matching the first m (encrypted) SNPs with its m (encrypted) inverses of the pattern. T then keeps the sum of this operation in a temporary result. Since most encrypted inverses and SNPs are reused in the next computation and they are aggregated, using the temporary result helps reduce the computational cost. That is, the next result is computed by subtracting the first encrypted SNP and adding the next encrypted SNP to the previous result. For each computation, the temporary result is stored in the result list with randomization. Whenever the pattern and Alice's SNPs match, the aggregated result will be an encrypted zero. When Alice receives randomly permuted results, she sees if any of the decrypted results are zero (see Figure 6).

To maintain the size-hiding property, we add m additional results, encryption of random values, so that the number of results is always n, independent of the pattern size. The pattern's position-hiding property still holds by randomly shuffling multiple results before they are sent to Alice. For completeness, Alice can just reveal the boundary positions, pos_0 and pos_{n+1}. In ES-SPH-PSM, we achieve orders of magnitude efficiency improvement with the same security and privacy guarantees.

Online Phase between Alice and T

Common Input: $H(\cdot), Enc(\cdot), Dec(\cdot), PK = \{pk_A, pk_{SL}\}$

Alice

$\mathcal{M}' := \{m'_0, ..., m'_{n+1}\}, C := \{c_0, ..., c_{n+1}\}$,
$E = \{e_0, ..., e_{n+1}\}, \Gamma = \{\gamma_0, ..., \gamma_n\}$

T

$P = (p_0, ..., p_m)$: a pattern
where $p_i = (pos''_i, l''_i)$, for $i = 0, ..., m$
$ep_i := Enc(pk_A, -H(p_i)), i = 0, ..., m$

Alice $\xrightarrow{E, C, \Gamma, pos_0, pos_{n+1}, s_0, s_{n+1}}$ T

```
If $Verify(pk_{SL}, (c_i, e_i, c_{i+1}, e_{i+1}), \gamma_i)) \neq 1$
  for any $i$ : $Abort$
See if 1. $Com(pos_0; s_0) = c_0 \wedge pos_0 < 0$
and 2. $Com(pos_{n+1}; s_{n+1}) = c_{n+1} \wedge pos_{n+1} > N$

/* Start of Subtring Matching */
// Unoptimized
// for $i = 1, ..., n - m$ :
//   $res_i := Enc(pk_A, 0)$
//   for $j = 0, ..., m$ :
//     $r_j \leftarrow \$, k := i + j$
//     $res_i = res_i \cdot (e_k \cdot ep_j)^{r_j}$

// Optimized
$tmp\_res := Enc(pk_A, 0)$
for $j = 0, ..., m$ :
  $tmp\_res = tmp\_res \cdot (e_{j+1} \cdot ep_j)$
$r_1 \leftarrow \$, res_1 = (tmp\_res)^{r_1}$
for $k = m + 1, ..., n - 1$ :
  $tmp\_res = tmp\_res \cdot e_{k+1} \cdot (e_{k-m})^{-1}$
  $r_{k-m+1} \leftarrow \$, res_{k-m+1} = (tmp\_res)^{r_{k-m+1}}$
$res_{n-m+1}, ..., res_n \leftarrow \$$
$res := \pi(res_1, ..., res_n)$, where $\pi$ : a random permutation
```

Alice $\xleftarrow{res}$ T

If $\exists i$ s.t. $Dec(sk_A, res_i) = 0$: Output YES
Else: Output NO

Fig. 6. Online phase of efficient and secure SPH-PSM (ES-SPH-PSM).

5.4 Flexible, Efficient, and Secure SPH-PSM (FES-SPH-PSM)

Genomic tests whose nature is known (e.g., immunity) often operate on a small range of the genome and may be public. This allows efficiency gains by querying only the mutations located in that small range. Our proposed techniques in Section 4 can be applied as in ES-SPH-PSM to preserve security goals for Alice's genome with adjustable privacy for the pattern. To keep the pattern's privacy (size- and position-hiding properties), a wider range including the required range can be used. The offline phase of the FES-SPH-PSM protocol is the same as the one of ES-SPH-PSM, so presented in Figure 5, and the online phase is in Figure 7. The two main differences are (1) T now queries a range, and (2) Alice provides encrypted mutations only in that range with NIZKRP proofs for boundary values.

Tables 4 and 5 show the computation and communication costs of each SPH-PSM variant.

5.5 Discussion

Different Test Result Learner. One may question the need for security in SPH-PSM since Alice learns the test result. From a legal point of view, genomic data, as well as the result, have to be authentic. To support such cases, SPH-PSM can be updated as follows: Alice's role in the protocol

Online Phase between Alice and T

Common Input: $H(\cdot), Enc(\cdot), Dec(\cdot), PK = \{pk_A, pk_{SL}\}$

Alice

$\mathcal{M}' := \{m'_0, ..., m'_{n+1}\}, C := \{c_0, ..., c_{n+1}\}$,
$E = \{e_0, ..., e_{n+1}\}, \Gamma = \{\gamma_0, ..., \gamma_n\}$

T

$P = (p_0, ..., p_m)$: a pattern, where
$p_i = (pos''_i, l''_i)$ and $a \le pos''_i \le b$
$ep_i := Enc(pk_A, -p_i), i = 0, ..., m$

$$\xleftarrow{Q = [a,b] (\text{or } [A,B] \supseteq [a,b])}$$

Choose all SNPs in Q, say $\mathcal{M}_Q = \{m'_{k+1}, ..., m'_{k+j}\}$
i.e., $pos'_k < a \le pos'_{k+1}$, and $pos'_{k+j} \le b < pos'_{k+j+1}$
and choose corresponding $e'_i s, c'_i s$, and $\gamma'_i s$
$l \leftarrow$ `NIZKRP_Prove`$\{(pos_k, s_k) \mid c_k = Com(pos_k; s_k) \wedge pos_k < 0\}$
$h \leftarrow$ `NIZKRP_Prove`$\{(pos_{k+j+1}, s_{k+j+1}) \mid c_{k+j+1} = Com(pos_{k+j+1}; s_{k+j+1}) \wedge pos_{k+j+1} > N\}$

$$\xrightarrow[(\gamma_k, ..., \gamma_{k+j}), l, h]{(e_k, ..., e_{k+j+1}), (c_k, ..., c_{k+j+1}),}$$

Check the followings and $Abort$, if not
1.$Verify(pk_{SL}, (c_i, e_i, c_{i+1}, e_{i+1}), \gamma_i)) = 1$,
for $i = k, ..., k+j$
2.`NIZK_Verify`$(c_k, l) = 1$
3.`NIZK_Verify`$(c_{k+j+1}, h) = 1$

/* Start of Substring Matching in ES-SPH-PSM * /

Fig. 7. (Online phase) flexible, efficient and secure SPH-PSM (FES-SPH-PSM).

Table 4. Operation Complexity of Each SPH-PSM Variant

	Offline			Online	
	SL	Alice	T	Alice	T
SPH-PSM	–	N Enc	m EncInv	1 IsZero	1 Enc, MultConstant $2m$ MultCiphers
S-SPH-PSM	N Enc, Hash, Sign	–	m EncInv	1 IsZero	m SigVerify, $2m$ MultCiphers, 1 MultConstant, 1 Enc
ES-SPH-PSM	$n+2$ Enc, $n+2$ Comm, $n+1$ Hash, Sign	$n+2$ Comm	m EncInv	$n/2$ IsZero (on average)	2 BoundaryCheck, CommCheck, $n+1$ Hash, SigVerify, n Enc, $n-m+1$ MultConstant **No opt:** $2(n-m+1)*m$ MultCiphers **Optimized:** $3n-m$ MultCiphers
FES-SPH-PSM	Same as ES-SHP-PSM			ComputeBoundaryIndex, 2 RangeProofGen $k/2$ IsZero (on average)	2 RangeProofVerify, $k+1$ Hash, SigVerify, k Enc, $(k-m+1)$ MultConstant **No opt.:** $2(k-m+1)*m$ MultCiphers **Optimized:** $3k-m$ MultCiphers

Enc: encryption of a AHE, MultConstant: $\text{Enc}(m)^r = \text{Enc}(m * r)$, MultCiphers: $\text{Enc}(m_1) * \text{Enc}(m_1) = \text{Enc}(m_1 + m_2)$, EncInv: $\text{Enc}(m^{(-1)}) = \text{Enc}(-m)$, IsZero: check if the input ciphertext is encryption of zero or not, Hash: computation of a cryptographic hash function, (Sign, SigVerify): a digital signature scheme, (Comm, CommCheck): a commitment scheme, (RangeProofGen, RangeProofVerify): a NIZKRP, and (ComputeBoundaryIndex, BoundaryCheck): integer comparison.

can be divided into two: genome owner (Alice) and test requester (e.g., court). Now, when SL encrypts the mutations, it uses the public key of the latter. The rest of the protocol remains the same, but T sends the encrypted result to the court, not to Alice. As a result, the court gets only the (correct) Boolean result for the test, and also Alice keeps her privacy by not revealing the whole genomic data either to T or to the court.

Genomic Similarity Testing. In genomic tests, sometimes not an exact match but a similarity score needs to be calculated (e.g., some paternity tests). S-SPH-PSM in Section 5.2 can be modified

Table 5. Data Transfer Complexity of Each SPH-PSM Variant

	Offline	**Online**	
	SL → Alice	**Alice → Tester**	**Tester → Alice**
SPH-PSM	-	N ciphertexts	1 ciphertext
S-SPH-PSM	N ciphertexts, signatures	N ciphertexts, signatures	Same as SPH-PSM
ES-SPH-PSM	$n+2$ ciphertexts, salts $n+1$ signatures	$n+2$ ciphertext, commitments $n+1$ signatures 2 positions (integers), salts	n ciphertexts
FES-SPH-PSM	Same as ES-SPH-PSM	$k+2$ ciphertexts, commitments $k+1$ signatures 2 range proofs	k ciphertexts

to support such tests by simply not aggregating the result in one ciphertext. Specifically, consider two input genomes for the similarity test and one of the parties (Alice) learns the result. In this case, both parties encrypt their genomes under Alice's public key and send it to T, which homomorphically adds received ciphertexts for each position, shuffles the order, and sends the shuffled list to Alice. Alice checks for similarity by decrypting each entry and counting the number of decrypted zeros.

6 IMPLEMENTATION

This section describes implementation details.

6.1 Genomic Range Query Protocol

For SL's signatures, we use the **Elliptic Curve Digital Signature Algorithm (ECDSA)** with elliptic curve `secp256r1`, as its signatures are relatively short (512-bit), compared to other schemes with the same security level (256 bits). For commitments, we use the **Fujisaki-Okamoto (FO)** commitment scheme [44] that allows us to create Boudot's range proofs [24]. Parameters used for these commitments are $s = 552$, and, for range proofs, $t = 128$, $l = 40$. For mutation commitments, we use a secure cryptographic hash function, `SHA2-256` [12], with 128-bit salts. A tuple is computed as

$$[\ FO(pos_i, s_{i,1}), SHA2(m_i, s_{i,2}), FO(pos_{i+1}, s_{i+1,1}), SHA2(m_{i+1}, s_{i+1,2}))\ ] \tag{2}$$

with randomly generated salts $s_{i,1}$ in $Z^*_{\mathcal{N}}$, where $\mathcal{N}$ is a composite number with two 512-bit prime factors, and 128-bit salts $s_{i,2}$ for $SHA2$.

To implement the commitments and range proofs, we used the code from [10]. We also used the Bouncy Castle [5] crypto library for cryptographic primitives, e.g., hash functions and signatures, and Java's SecureRandom class [7] for generating salts. The code for offline and online phases was written in Java and was evaluated on a PC with an Intel(R) Core(TM) i7-3770 CPU @ 3.40GHz chip and 16GB of RAM.

6.2 SPH-PSM Variants

For the SPH-PSM variants, we use Pedersen commitments [71], the additively homomorphic variant of ElGamal [41] (AH-ElGamal), and Bulletproofs [27] for the NIZKRP in FES-SPH-PSM. The structures of a tuple for ES-SPH-PSM and FES-SPH-PSM are similar to the tuple (2) in Section 6.1 of the secure range query protocol, except FO is replaced with Pedersen commitments, while $SHA2$ commitments are replaced with encrypted bases. The same ECDSA setting is used for SL's signatures as in the previous section.

Our codebase is in Golang [3], evaluated on a server with 64 Intel(R) Xeon(R) CPUs E5-4610 v2 @ 2.30GHz and 128GB of RAM. We modified the official Golang implementation for ElGamal to implement an AH-ElGamal. As in the previous section, we use the official Golang crypto library

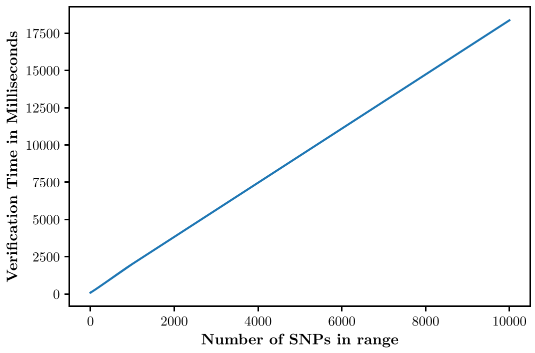


Fig. 8. T's verification time given the number of SNPs in the queried range.

for SHA256 and ECDSA with the curve secp256r1 for hashing and digital signatures, respectively; we also use the range proof code from [10].

7 EVALUATION

This section evaluates the proposed construction. For the sake of reproducibility, all our implementation and evaluation results are publicly available at [13].

7.1 Synthetic Genomic Data Generation

For the experiments, we generated synthetic genomic data with $3 \cdot 10^9$ bases for the whole genome and $3 \cdot 10^6$ bases for SNP representations. Each base pair is structured with a 32-bit integer to represent its position and an 8-bit integer for a combination of two base letters. Eight-bit integers are defined with alphabetical order of combinations, i.e., 0 = "AA," 1 = "AC,"..., 15 = "TT." For the evaluation of Section 7.2, we used a single type of SNP ("AA") since our measurements do not depend on the specific SNP type. For SPH-PSM variants, we generated necessary genomic files before each experiment and let each experiment read the required files during the evaluation. To generate a synthetic genomic file, we input the total size of the data `n`; starting and ending positions, `s` and `e`, of the pattern; and a Boolean variable `isSNP` to check if it is for SNP representation or the whole genome representation. Note that which base letters are used does not affect the evaluation result. For simplicity, we generate bases with "AA" for all other positions and bases with "TT" for the positions in [`s`,`e`]. For T's pattern (marked with `n` = 0), we generate bases with "TT" for the positions in the range [`s`,`e`]. If `isSNP` is `True`, we generate bases for every 1,000 positions, whereas we generate bases for all integers if it is `False`.

7.2 Secure Genomic Range Queries

We used $3 \cdot 10^6$ SNPs for the experiment. The entire offline phase, including commitments and signature generations, took 4.2 hours on the aforementioned platform. We note that this process can be easily parallelized. Times for individual operations are:

- Fujisaki-Okamoto commitment: **3.5 ms**
- Boudot's range proof: **47.7 ms** for proof generation and **37 ms** for validation
- SHA2 commitment: **0.3 ms** (including salt creation) and **0.1 ms** for validation

Salts can be alternatively (re-)generated using a seed and a **Pseudorandom number generator (PRNG)**, or HKDF [54], a simple key derivation function based on HMAC [20], to reduce the storage cost. Verification cost incurred by T scales linearly with the number of SNPs in Q, while verification cost of the range proof is dominated by signature verification, as shown in Figure 8.

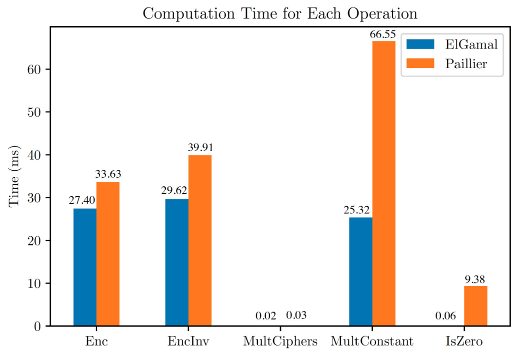


Fig. 9. Computation time comparison between AH-ElGamal and Paillier for operations in SPH-PSM variants. Enc is encryption and EncInv is modular inversion of encryption result. MultCiphers and MultConstant are multiplication of a ciphertext with another ciphertext and a constant, respectively. IsZero is used to check whether a given ciphertext is an encryption of zero.

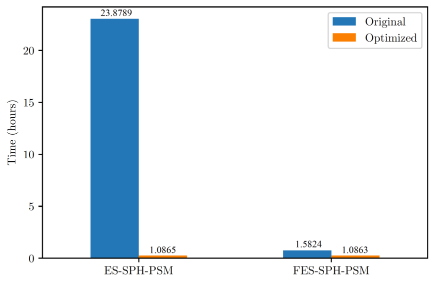


Fig. 10. Total runtime comparison between optimized vs. original ES-SPH-PSM and FES-SPH-PSM.

7.3 SPH-PSM Variants

To assess the computational efficiency of AH-ElGamal, we first compared it to Paillier [69], a well-known additive PHE scheme, using its popular implementation [6]. Figure 9 shows the comparison of AH-ElGamal and Paillier for each operation. Since we do not use the full decryption operation and only check if a ciphertext is encryption of zero, results show that AH-ElGamal is significantly more efficient for all operations used in SPH-PSM variants. Thus, we use AH-ElGamal for a homomorphic encryption scheme in the rest of the evaluation.

For individual operations, computation times averaged over 10 executions were as follows:

- Salt Generation: **2.7** μs
- Hash Computations: **4.8** μs for $H(pos_i, l_i)$, **3.9** μs for $h2 := H(pos_i, e_i)$, and **10.4** μs for $h3 := H(Tup_i)$
- Digital Signature Generation: **91.1** μs for $sig1 := Sign(h2)$ and **70.2** μs for $sig2 := Sign(h3)$
- Digital Signature Verification: **171.4** μs for $Verify(sig1)$ and **157.6** μs for $Verify(sig2)$
- Commitment Generation: **511.5** μs
- Range Proof (for FES-SPH-PSM):
  Bulletproofs [27]: **262.2 ms** for proof generation and **168.0 ms** for validation
  Signature-based [28]: **1,324.6 ms** for proof generation and **1,299.4 ms** for validation

We use the most efficient range proof scheme, Bulletproofs, for NIZKRP in the rest of the evaluation.

Next, we measured the effects of the optimization described in Section 5.3 for both ES-PSH-PSM and FES-SPH-PSM. We used 10^8 bases for Alice's genome and 10^7 bases for T's pattern. Figure 10 shows that it lowers test completion times drastically, which confirms that even complex tests can be performed fast using this optimization. It also shows that FES-SPH-PSM is much more efficient than ES-SPH-PSM, i.e., when the location of the test pattern is public.

To measure the overhead of adding security to SPH-PSM, we compared computation costs of SPH-PSM and S-SPH-PSM. We implemented the algorithm from [37] in GoLang and compared the cost for offline phases. We compared only the offline phases because the online phases are almost the same and the only difference—signature verifications—depends on the pattern size, which is

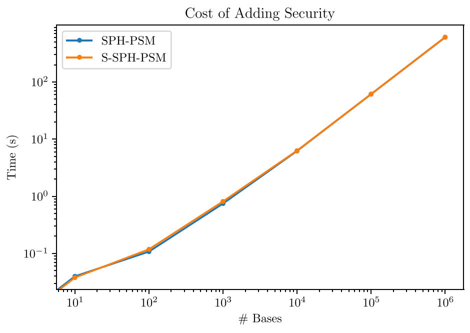


Fig. 11. Offline computation cost comparison between SPH-PSM and S-SPH-PSM.

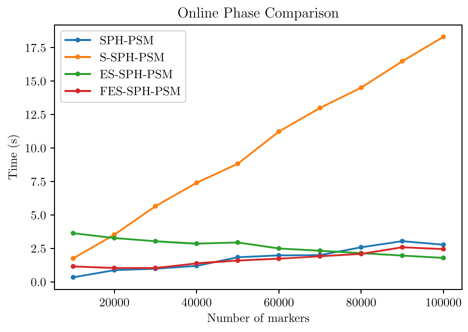


Fig. 12. Online phase comparison of SPH-PSM variants.

relatively small to the whole genome size. Since these signing operations are independent, we use multi-threading to parallelize them and reduce delay. Results in Figure 11 show that the overhead of adding security is negligible compared to the cost of homomorphic encryption operations for the whole genome.

For the online phase, we vary the number of markers and use the whole genome. Figure 12 shows the results: test completion time for S-SPH-PSM grows linearly with the number of markers in Alice's genome compared to SPH-PSM. This is because T verifies signatures within the range of its markers. ES-SPH-PSM and FES-SPH-PSM benefit from the optimization and their runtimes are comparable to that of SPH-PSM, even though they add security and reduce data transfer by three orders of magnitude.

To show that proposed constructs can be parallelized, we implemented multi-threading for offline and online phases of SPH-PSM variants. We believe that this is reasonable as a consumer-facing sequencing lab would be equipped with high-end storage and computing devices capable of handling massive amounts of genomic data. Figure 13 shows a comparison between single-threading and multi-threading with the increasing number of bases for S-SPH-PSM and ES-SPH-PSM. Since each signature generation and verification are independent, parallelization greatly lowers the computation time. Specifically, S-SPH-PSM offline and online phases using multi-threading are 32 and 12 times faster, respectively, compared to using a single thread. Similarly, the offline phase of ES-SPH-PSM is 32 times faster using multi-threading with over 10^7 bases. In contrast, multi-threading only slightly improves the online phase of ES-SPH-PSM, since signature verification is the only operation that can be parallelized.

8 GENERALIZATION TO SPARSE INTEGERS

Although our primary application domain is genomics, the proposed technique applies to other settings. For example, suppose that a forensics team, after obtaining a court order, wants to examine an **Internet Service Provider's (ISP)** log regarding a particular account's activities for a time period relevant to a cyberspace attack. This setting resembles genomic testing in terms of security and privacy requirements as well as the sparsity of sensitive data. From the security perspective, the forensics team needs to ensure that the log entries supplied by the ISP for the period and account of interest are genuine (authentic) and complete. From the privacy perspective, the ISP does not want to reveal any non-relevant log entries in order to protect its customers' privacy and/or its own interests, whereas the relevant log entries (i.e., log entries corresponding to the period and account of interest) are likely to be sparsely distributed within a very large dataset.

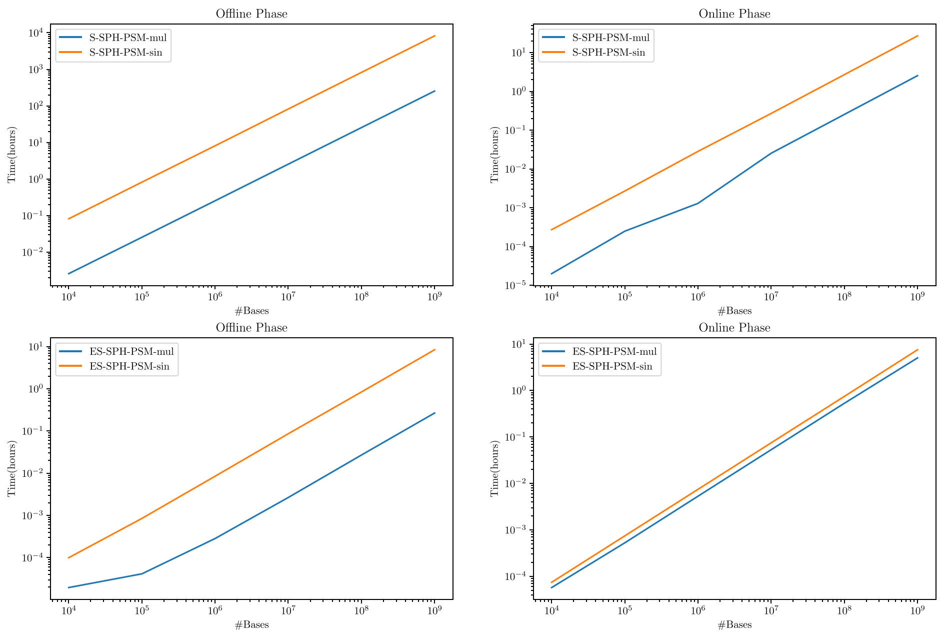


Fig. 13. Comparison between multi-threaded and single-threaded results.

Similar situations might arise in the context of range queries over firewall logs or ticket transaction databases. The proposed technique can be adapted to address their security and privacy challenges. In this section, we define a generalized version of the proposed scheme geared for secure and private range queries over sparse integers.

8.1 Secure and Private Range Queries over Sparse Integers

Let $\mathcal{D}$ be a domain set of consecutive integers from A to B, i.e., $\mathcal{D} = [A, B] \subseteq \mathbb{Z}$, for some $A, B \in \mathbb{Z}$. Though there are many ways to mathematically define a *sparse set* [22, 59, 80], informally, we call a subset $\mathcal{X}$ of a set $\mathcal{D}$ *sparse* if $\frac{|\mathcal{X}|}{|\mathcal{D}|}$ is small. Note that our approach also works for the improper subset $\mathcal{X}$, i.e., when $\mathcal{X} = \mathcal{D}$; however, it is more meaningful in terms of privacy if the subset $\mathcal{X}$ is sparse in $\mathcal{D}$.

We assume Alice and Bob act as a replier and a querier, respectively. Alice has a sparse subset $\mathcal{X}$ of $\mathcal{D} = [A, B]$, with n integers, and Bob has a range query $Q = [a, b]$ such that $A \leq a \leq b \leq B$ to receive all integers in $\mathcal{X} \cap Q$; i.e., if Alice's sparse set is $\mathcal{X} = \{x_1, x_2, \ldots, x_n\}$, then Bob wants to receive all x_is such that $a \leq x_i \leq b$.

Definition 8.1. Let $\mathcal{D} = [A, B] \subseteq \mathbb{Z}$ be a domain set of consecutive integers, for some $A, B \in \mathbb{Z}$. The ideal functionality $\mathcal{F}$ of the protocol between Alice and Bob, on input $\mathcal{X}$ and Q, is

$$\mathcal{F} : (\mathcal{X}, Q) \rightarrow (Q, \mathcal{X} \cap Q),$$

where $\mathcal{X}$ is a sparse subset of $\mathcal{D}$ and Q is a consecutive subset of $\mathcal{D}$.

Security goals are the same as in Section 3: authenticity, completeness, and privacy:

(1) **Authenticity.** Alice cannot modify $\mathcal{X}$ after it is chosen and Bob can check if the integers returned by Alice are the ones that Alice chose, i.e., authentic.

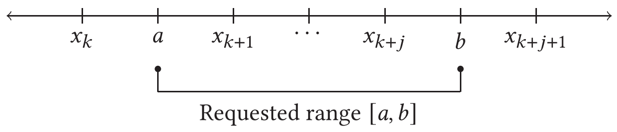


Fig. 14. Proving $x_k < a$ states x_w, for $\forall w \leq k$ is out of range. Similarly, proving $x_{k+j+1} > b$ states x_w, for $\forall w \geq (k + j + 1)$ is out of range.

(2) **Completeness.** Bob should be convinced that the integers returned by Alice are complete, i.e., are within Q without any omissions.
(3) **Alice's Privacy.** Bob learns no information about the integers in $\mathcal{X}$ that are outside Q.

8.2 Construction and Its Security

We now describe a concrete construction that meets our stated goals. We assume a trusted offline authority called *Auth* (as *SL* in Section 4) that facilitates trust among the protocol participants. The offline phase is for *Auth* to authorize Alices's sparse integer set $\mathcal{X}$ and the online phase where Alice and Bob interact is for Bob to get integers in $\mathcal{X} \cap Q$. Similar to Section 4, *Auth* first chooses two additional integers outside of $\mathcal{D}$ to indicate the boundaries and random salts for commitments. It then commits each element in $\mathcal{X}$, makes adjacent commitments in a tuple, and signs those tuples. Since only one commitment is required for each element, the overall size of salts being sent is halved. In the online phase, Alice chooses the elements in Q and computes two commitments for the sentinel values outside of Q and the two range proofs for them. Bob's computation cost is also reduced since the number of commitments is halved. As shown in Figure 14, NIZKRP steps for checking $x_k < a$ and $x_{k+j+1} > b$ show that any integers (other than the returned ones) are out of range, as integers are sorted before they are committed. Detailed offline and online phases are described in Figures 15 and 16, respectively.

We now give a proof sketch of security and privacy for the proposed construction:

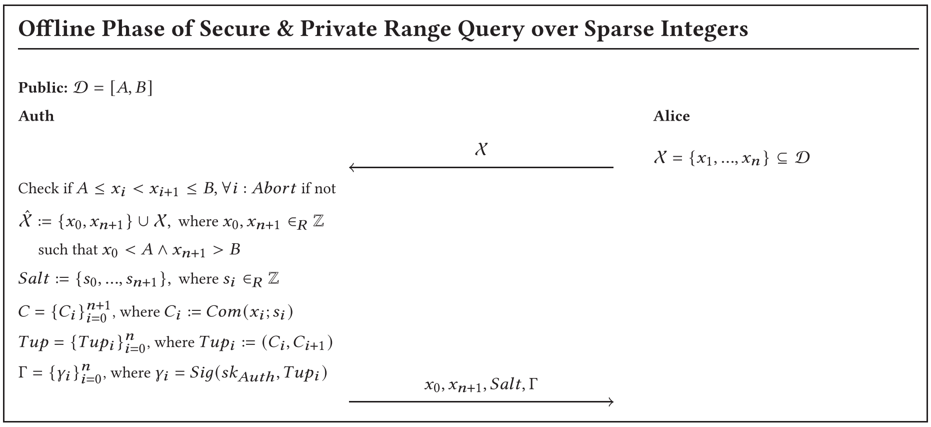
(1) Goal 1 is satisfied by the binding property of the commitment scheme, along with signatures on the tuples containing two commitments of neighboring integers. Signatures on commitments ensure that no element of $\mathcal{X}$ can be changed.
(2) Signatures on commitment tuples ensure that any inclusion or omission of elements is detectable.
(3) Consider tuples outside $Q = [a, b]$, i.e., $\{Com(x_k), Com(x_{k+1})\}$ and $\{Com(x_{k+j}), Com(x_{k+j+1})\}$ in Figure 14. Due to the hiding property of the commitment scheme, Bob learns no information about either x_k or x_{k+j+1} from $Com(x_k)$ and $Com(x_{k+j+1})$, respectively. Also, NIZKRP guarantees that no information about x_k and x_{k+j+1} is revealed other than the fact $x_k < a$ and $x_{k+j+1} > b$, respectively.

9 RELATED WORK

Privacy of genetic material and tests has attracted much attention, and numerous methods have been proposed that are based on various cryptographic techniques. Genomic security, on the other hand, remained in the background due to a (mistakenly) perceived lack of challenges. In this section, we briefly go over the cryptographic privacy building blocks in the genomics domain and then discuss related techniques for achieving data authenticity and integrity in the context of range queries.

9.1 Genomic Privacy

Several recent survey papers [18, 23, 64, 76] overview recent advances in genomic privacy. In this section, we list the techniques used in this domain and their related work. Secure **multi-party**



Offline Phase of Secure & Private Range Query over Sparse Integers

Public: $\mathcal{D} = [A, B]$

Auth | **Alice**

Alice: $\mathcal{X} = \{x_1, ..., x_n\} \subseteq \mathcal{D}$

$\xleftarrow{\mathcal{X}}$

Check if $A \leq x_i < x_{i+1} \leq B, \forall i : Abort$ if not

$\hat{\mathcal{X}} := \{x_0, x_{n+1}\} \cup \mathcal{X}$, where $x_0, x_{n+1} \in_R \mathbb{Z}$ such that $x_0 < A \wedge x_{n+1} > B$

$Salt := \{s_0, ..., s_{n+1}\}$, where $s_i \in_R \mathbb{Z}$

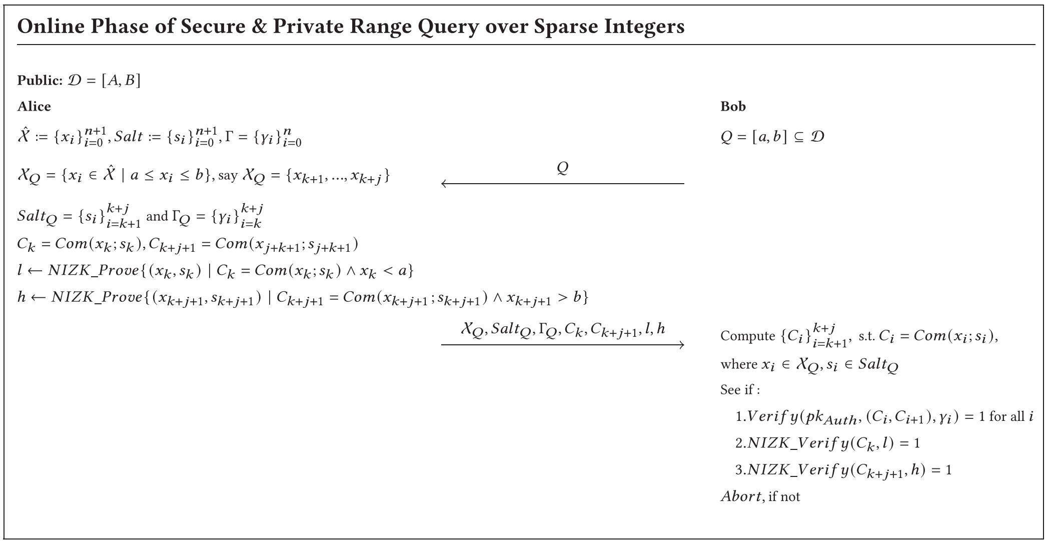
$C = \{C_i\}_{i=0}^{n+1}$, where $C_i := Com(x_i; s_i)$

$Tup = \{Tup_i\}_{i=0}^{n}$, where $Tup_i := (C_i, C_{i+1})$

$\Gamma = \{\gamma_i\}_{i=0}^{n}$, where $\gamma_i = Sig(sk_{Auth}, Tup_i)$

$\xrightarrow{x_0, x_{n+1}, Salt, \Gamma}$

Fig. 15. (Offline phase) authorizing Alice's sparse integer set $\mathcal{X}$ from *Auth*.



Online Phase of Secure & Private Range Query over Sparse Integers

Public: $\mathcal{D} = [A, B]$

Alice | **Bob**

Alice: $\hat{\mathcal{X}} := \{x_i\}_{i=0}^{n+1}, Salt := \{s_i\}_{i=0}^{n+1}, \Gamma = \{\gamma_i\}_{i=0}^{n}$

Bob: $Q = [a, b] \subseteq \mathcal{D}$

$\xleftarrow{Q}$

$\mathcal{X}_Q = \{x_i \in \hat{\mathcal{X}} \mid a \leq x_i \leq b\}$, say $\mathcal{X}_Q = \{x_{k+1}, ..., x_{k+j}\}$

$Salt_Q = \{s_i\}_{i=k+1}^{k+j}$ and $\Gamma_Q = \{\gamma_i\}_{i=k}^{k+j}$

$C_k = Com(x_k; s_k), C_{k+j+1} = Com(x_{j+k+1}; s_{j+k+1})$

$l \leftarrow NIZK\_Prove\{(x_k, s_k) \mid C_k = Com(x_k; s_k) \wedge x_k < a\}$

$h \leftarrow NIZK\_Prove\{(x_{k+j+1}, s_{k+j+1}) \mid C_{k+j+1} = Com(x_{k+j+1}; s_{k+j+1}) \wedge x_{k+j+1} > b\}$

$\xrightarrow{\mathcal{X}_Q, Salt_Q, \Gamma_Q, C_k, C_{k+j+1}, l, h}$

Bob: Compute $\{C_i\}_{i=k+1}^{k+j}$, s.t. $C_i = Com(x_i; s_i)$, where $x_i \in \mathcal{X}_Q, s_i \in Salt_Q$

See if :

1.$Verify(pk_{Auth}, (C_i, C_{i+1}), \gamma_i) = 1$ for all i

2.$NIZK\_Verify(C_k, l) = 1$

3.$NIZK\_Verify(C_{k+j+1}, h) = 1$

$Abort$, if not

Fig. 16. (Online phase) range query-response between Alice and Bob.

computation (MPC) allows two or more parties to jointly compute a function on their private inputs without revealing any information, other than the output of the function. For example, [51] utilizes **oblivious transfer (OT)** and oblivious circuit evaluation to compute the edit distance and Smith-Waterman similarity score to measure the similarity between two DNA sequences. [19] uses a variant of **Private Set Intersection (PSI)**, PSI Cardinality, which reveals only the size of the intersection over two input sets, for paternity tests by observing the sizes of DNA fragments cut by restricting enzymes. It also shows how to use Authorized PSI for personalized medicine, where an authority authorizes the markers to be checked in the DNA. [36] uses an Android smartphone to store encrypted genomic data and PSI techniques to provide results of personal medicine, paternity, and ancestry tests using the smartphone as the end-user computation device. [81] utilizes secret-sharing-based MPC techniques to compute **minor allele frequencies (MAFs)** and chi-squared statistics in the context of Genome-Wide Association Study computation and the Hamming distance between two genomic datasets. To improve the efficiency of edit distance computation, [78] approximates the edit distance by compressing genome sequences to sets and privately computing the set difference size (or the threshold of the set difference size) using garbled circuit and OT techniques.

Homomorphic Encryption (HE) is another popular tool for privacy in genomic tests. For example, [14] uses so-called Partial HE (see Section 2.4) and dynamic programming techniques to compute the edit distance of two DNA sequences without revealing their sequences to each other, which is later improved in [15]. Subsequently, [31] suggests a way to compute the edit distance on encrypted genomic data using Somewhat HE (see Section 2.4). [52] constructs a secure genomic data query architecture with third parties, where one party stores and computes encrypted genomic data—e.g., **storage and processing unit (SPU)**—while the other party manages the keys used to encrypt and decrypt genomic data. It uses the additively homomorphic property of the Paillier [69] cryptosystem to secure count query. Similarly, [16] suggests an architecture for disease susceptibility tests, with a third-party SPU. This technique utilizes Paillier Partial HE and proxy re-encryption [21] so that only the two parties who receive the partial secret keys from the patient can participate in the test; i.e., SPU homomorphically computes the test on the encrypted DNA and partially decrypts the result, such that the medical center can obtain the final result by partial decryption of the message received from SPU. These ideas are further refined in [17] and [35], which showed that [17] can be more efficiently implemented by using additively homomorphic Elliptic Curve-based ElGamal and simpler encoding method of genomic data. [72] presents a method to search encrypted biomedical data stored in the server using Bloom filters and HE, so that the user can perform the search query to the server. [55] uses a ring-based FHE scheme [60] to encrypt genomic data and demonstrates common genomic computations over encrypted data, e.g., Pearson Goodness-of-Fit test, the D' and r^2 measures of linkage disequilibrium, the **Estimation Maximization (EM)** algorithm for haplotyping, and the Cochran-Armitage Test for Trend. [82] and [77] also assume encrypted genomic data with an FHE [47] on an untrusted public cloud. [82] focuses on the chi-square statistics and proposes two protocols for secure division operation, while [77] suggests a framework for estimating the P-value of exact logistic regression parameters over encrypted data. [48] shows how to construct an index tree of the genomic data and send the index tree to the cloud server after encryption with Paillier; the server then traverses the encrypted tree according to the encrypted query from a query initiator using secure function evaluation via an interactive protocol using Yao's garbled circuits [79].

9.2 Range Query Security

Range query completeness was explored in the context of outsourced databases where the data owner assigns query handling to a cloud-based data publisher. The latter, if malicious, can reply to a range query with incomplete and/or fake results. To counter such misbehavior, [50] focused on minimizing privacy leakages on data attributes using data partitioning algorithms that are aware of the distribution of query ranges. [70] developed methods based on continual linking of elements and collision-resistant hash functions to prevent malicious actions. [56] used Merkle hash and B$^+$ trees, as well as aggregated signatures, to provide authenticity and integrity (with less strict privacy requirements than [70]) and improve efficiency in the dynamic database case.

Other cryptographic range query techniques incorporated so-called range proofs. Early examples of range proofs include [26, 30, 62]. [62] uses the bit-length of the committed value to prove that the number is in the range $[0, 2^k - 1]$, where k is the number of bits in that committed value. [26] only proves that the committed value lies in a wider range: $[-a, 2a]$, instead of $[0, a]$. [30] convinces a verifier that the committed value lies in a range with the expansion rate of 2^{t+l+1}, where t and l are security parameters. [24] proposes two efficient protocols for proving that the committed value that is in $[a, b]$ lies in $[a - \theta, b + \theta]$, where $\theta = 2^{t+l+1}\sqrt{b-a}$ and t and l are security parameters. The second protocol (that uses Fujisaki-Okamoto commitments [44]) is the one we used in this article; it has the expansion rate of 1. [28] proposes range proofs based on set membership protocols by extending the u-ary notation and proving that the secret $z \in [0, u^l - 1]$.

10 LIMITATIONS AND FUTURE WORK

In this work, we focused on genomic tests that reveal the genomic data in some queried range and showed our proposed technique can be used for private substring matching-type tests where genomic data can be encrypted under an additively homomorphic encryption scheme. However, different types of genomic tests may require more operations other than addition, subtraction, and constant multiplication that additively homomorphic encryption cannot support. We cannot avoid using some SWHE or FHE in such cases to keep the privacy, sacrificing the performance, as some previous work [31, 55, 77, 82] suggested.

We avoided going into discussions regarding low-level side channels in our work. For instance, Alice can infer the size of the queried range from execution time when applying our technique to SPH-PSM as in Section 5. However, this can be easily prevented in practice by letting T send the replies after a fixed period of time instead of sending them right after the computation is finished.

Lastly, we only consider the SNP for the efficient genomic representation; however, there are multiple genomic representation formats such as STR and RFLP. Also, for the base letters, we only consider well-sequenced genomic data without any gene duplications, insertions, deletions, or lateral gene transfers that can commonly occur in genomic materials. Therefore, there is still room for improving our proposed techniques and we consider these to be opportunities for future work.

11 CONCLUSIONS

Genomic privacy has understandably attracted lots of attention due to the dire consequences of possible leaks. Meanwhile, genomic security (authentication and integrity of genomic data) has remained relatively obscure. This article motivated and constructed a technique for secure and private genomic range queries. Its key properties are *Authenticity* of genomic material, *Completeness* of mutations within a given range, and total *Privacy* as far as any genomic data outside a given range.

To achieve these properties, we used techniques based on zero-knowledge range proofs to show that a committed value (the position of a genomic mutation) is outside the queried range, digital signatures to prevent any alterations on genomic material, and linkage among two consecutive mutations to preclude any omissions. To improve protocol efficiency, we carefully chose the cryptographic primitives while keeping in mind both security and privacy requirements. We also abstracted away from genomics and defined a more general problem of secure and private range queries over sparse integers.

Although we focused on revealing plaintext SNPs within the queried range to a tester, the proposed technique is equally applicable to encrypted genomes. Previous work that utilizes homomorphic encryption to offer SNP matches or weighted averaging without revealing the plaintext SNPs (e.g., [16, 35, 52]) can directly apply our approach to support authenticity and completeness of mutations within the queried range, while allowing the privacy of mutations. We showed, as an example, that a private substring matching algorithm [37] can benefit from the proposed technique in terms of better security and performance.

ACKNOWLEDGMENTS

We sincerely thank anonymous reviewers for their valuable comments and suggestions.

REFERENCES

[1] 23andMe. Retrieved January 24, 2022, from https://www.23andme.com/.
[2] CRI Genetics. Retrieved January 24, 2022, from https://www.crigenetics.com/.
[3] GO. Retrieved January 24, 2022, from https://golang.org/.
[4] How do geneticists indicate the location of a gene? Retrieved January 24, 2022, from https://ghr.nlm.nih.gov/primer/howgeneswork/genelocation.
[5] The Legion of the Bouncy Castle. Retrieved January 24, 2022, from https://www.bouncycastle.org/.
[6] Paillier. Retrieved January 24, 2022, from https://github.com/didiercrunch/paillier.
[7] SecureRandom (Java Platform SE 8). Retrieved January 24, 2022, from https://docs.oracle.com/javase/8/docs/api/java/security/SecureRandom.html.
[8] SNP. Retrieved January 24, 2022, from https://www.nature.com/scitable/definition/snp-295/#:~:text=If%20more%20than%201%25%20of,having%20more%20than%20one%20allele.
[9] Whole Genome Association Studies. Retrieved January 24, 2022, from https://www.genome.gov/17516714/2006-release-about-whole-genome-association-studies.
[10] Zero-Knowledge Proofs. Retrieved January 24, 2022, from https://github.com/ing-bank/zkproofs.
[11] National Research Council (US) Committee on Mapping and Sequencing the Human Genome. 1988. Mapping and Sequencing the Human Genome. Washington, DC: National Academies Press (US), 1988. 2, Introduction. Retrieved January 31, 2022, from https://www.ncbi.nlm.nih.gov/books/NBK218247/.
[12] 2012. Secure Hash Standard. FIPS PUB 180-4, Information Technology Laboratory, National Institute of Standards and Technology, Gaithersburg, MD, 2012.
[13] S. Hwang, E. Ozturk, and G. Tsudik. 2022. Source code for evaluation. https://github.com/sprout-uci/genomic-security-journal-code.
[14] Mikhail J. Atallah, Florian Kerschbaum, and Wenliang Du. 2003. Secure and private sequence comparisons. In *Proceedings of the 2003 ACM Workshop on Privacy in the Electronic Society (WPES'03)*, Association for Computing Machinery, New York, NY, 39–44. https://doi.org/10.1145/1005140.1005147
[15] Mikhail J. Atallah and Jiangtao Li. 2005. Secure outsourcing of sequence comparisons. *International Journal of Information Security* 4, 4 (2005), 277–287. https://doi.org/10.1007/s10207-005-0070-3
[16] Erman Ayday, Jean Louis Raisaro, and Jean-Pierre Hubaux. 2013. Privacy-enhancing technologies for medical tests using genomic data. In *Proceeding of the Network and Distributed System Security Symposium (NDSS'13)*.
[17] Erman Ayday, Jean Louis Raisaro, Jean-Pierre Hubaux, and Jacques Rougemont. 2013. Protecting and evaluating genomic privacy inmedical tests and personalized medicine. In *Proceedings of the 12th ACM Workshop on Workshop on Privacy in the Electronic Society (WPES'13)*, Association for Computing Machinery, New York, NY, 95–106. https://doi.org/10.1145/2517840.2517843
[18] Abinaya B. and Santhi S. 2021. A survey on genomic data by privacy-preserving techniques perspective. *Computational Biology and Chemistry* 93 (2021), 107538.
[19] Pierre Baldi, Roberta Baronio, Emiliano De Cristofaro, Paolo Gasti, and Gene Tsudik. 2011. Countering GATTACA: Efficient and secure testing of fully-sequenced human genomes. In *Proceedings of the 18th ACM Conference on Computer and Communications Security (CCS'11)*, Association for Computing Machinery, New York, NY, 691–702. https://doi.org/10.1145/2046707.2046785
[20] Mihir Bellare, Ran Canetti, and Hugo Krawczyk. 1996. Keying hash functions for message authentication. In *Advances in Cryptology (CRYPTO'96)*, Neal Koblitz (Ed.). Springer, Berlin, 1–15.
[21] Matt Blaze, Gerrit Bleumer, and Martin Strauss. 1998. Divertible protocols and atomic proxy cryptography. In *Advances in Cryptology (EUROCRYPT'98)*, Kaisa Nyberg (Ed.). Springer, Berlin, 127–144.
[22] Henry Blumberg. 1939. Exceptional sets. In *Fundamenta Mathematicae.* 3–32.
[23] Luca Bonomi, Yingxiang Huang, and Lucila Ohno-Machado. 2020. Privacy challenges and research opportunities for genomic data sharing. *Nature Genetics* 52, 7 (July 2020), 646–654.
[24] Fabrice Boudot. 2000. Efficient proofs that a committed number lies in an interval. In *International Conference on the Theory and Applications of Cryptographic Techniques.* Springer, 431–444.
[25] Tatiana Bradley, Xuhua Ding, and Gene Tsudik. 2017. Genomic security (lest we forget). *IEEE Security & Privacy* 15, 5 (2017), 38–46.
[26] Ernest F. Brickell, David Chaum, Ivan B. Damgård, and Jeroen van de Graaf. 1987. Gradual and verifiable release of a secret. In *Conference on the Theory and Application of Cryptographic Techniques.* Springer, 156–166.

[27] Benedikt Bünz, Jonathan Bootle, Dan Boneh, Andrew Poelstra, Pieter Wuille, and Greg Maxwell. 2018. Bulletproofs: Short proofs for confidential transactions and more. In *2018 IEEE Symposium on Security and Privacy (SP'18)*. IEEE, 315–334.

[28] Jan Camenisch, Rafik Chaabouni, and Abhi Shelat. 2008. Efficient protocols for set membership and range proofs. In *Proceeding of the Advances in Cryptology-(ASIACRYPT'08)*, Josef Pieprzyk (Ed.). Springer, Berlin Heidelberg, 234–252.

[29] Sébastien Canard, Iwen Coisel, Amandine Jambert, and Jacques Traoré. 2014. New results for the practical use of range proofs. In *Public Key Infrastructures, Services and Applications*, Sokratis Katsikas and Isaac Agudo (Eds.). Springer, Berlin, 47–64.

[30] Agnes Hui Chan, Yair Frankel, and Yiannis Tsiounis. 1998. Easy come-easy go divisible cash. In *Proceeding of the Advances in Cryptology (EUROCRYPT'98, International Conference on the Theory and Application of Cryptographic Techniques, Espoo, Finland, May 31-June 4, 1998, Proceeding)*, Lecture Notes in Computer Science, Springer, Vol. 1403, 561–575. DOI: 10.1007/BFb0054154

[31] Jung Hee Cheon, Miran Kim, and Kristin Lauter. 2015. Homomorphic computation of edit distance. In *Financial Cryptography and Data Security*, Michael Brenner, Nicolas Christin, Benjamin Johnson, and Kurt Rohloff (Eds.). Springer, Berlin, 194–212.

[32] Josh Benaloh Clarkson. 1994. Dense probabilistic encryption. In *Proceedings of the Workshop on Selected Areas of Cryptography*. 120–128.

[33] Ronald Cramer, Rosario Gennaro, and Berry Schoenmakers. 1997. A secure and optimally efficient multi-authority election scheme. In *Advances in Cryptology (EUROCRYPT'97)*, Walter Fumy (Ed.). Springer, Berlin, 103–118.

[34] Ivan Damgård and Mads Jurik. 2001. A generalisation, a simplification and some applications of Paillier's probabilistic public-key system. In *Public Key Cryptography*, Kwangjo Kim (Ed.). Springer, Berlin, 119–136.

[35] George Danezis and Emiliano De Cristofaro. 2014. Fast and private genomic testing for disease susceptibility. In *Proceedings of the 13th Workshop on Privacy in the Electronic Society*. ACM, 31–34.

[36] Emiliano De Cristofaro, Sky Faber, Paolo Gasti, and Gene Tsudik. 2012. Genodroid: Are privacy-preserving genomic tests ready for prime time? In *Proceedings of the 2012 ACM Workshop on Privacy in the Electronic Society*. ACM, 97–108.

[37] Emiliano De Cristofaro, Sky Faber, and Gene Tsudik. 2013. Secure genomic testing with size-and position-hiding private substring matching. In *Proceedings of the 12th ACM Workshop on Workshop on Privacy in the Electronic Society*. 107–118.

[38] Chunhua Deng, Jia Fan, Zhen Wang, Yili Luo, Yue Zheng, Yixin Li, and Jianwei Ding. 2019. A survey on range proof and its applications on blockchain. In *2019 International Conference on Cyber-Enabled Distributed Computing and Knowledge Discovery (CyberC'19)*. 1–8. https://doi.org/10.1109/CyberC.2019.00011

[39] Premkumar Devanbu, Michael Gertz, Charles Martel, and Stuart G. Stubblebine. 2003. Authentic data publication over the Internet. *Journal of Computer Security* 11, 3 (2003), 291–314.

[40] Xuhua Ding, Ercan Ozturk, and Gene Tsudik. 2019. Balancing security and privacy in genomic range queries. In *Proceedings of the 18th ACM Workshop on Privacy in the Electronic Society (WPES'19)*. Association for Computing Machinery, New York, NY, 106–110. https://doi.org/10.1145/3338498.3358652

[41] Taher ElGamal. 1985. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE Transactions on Information Theory* 31, 4 (1985), 469–472.

[42] Yanxiao Feng, Yuechuan Zhang, Cuifeng Ying, Deqiang Wang, and Chunlei Du. 2015. Nanopore-based fourth-generation DNA sequencing technology. *Genomics, Proteomics & Bioinformatics* 13, 1 (2015), 4–16.

[43] Amos Fiat and Adi Shamir. 1987. How to prove yourself: Practical solutions to identification and signature problems. In *Proceedings on Advances in Cryptology (CRYPTO'86)*. Springer-Verlag, Berlin, 186–194.

[44] Eiichiro Fujisaki and Tatsuaki Okamoto. 1997. Statistical zero knowledge protocols to prove modular polynomial relations. In *Annual International Cryptology Conference*. Springer, 16–30.

[45] Shafi Goldwasser and Silvio Micali. 1982. Probabilistic encryption & how to play mental poker keeping secret all partial information. In *Proceedings of the 14th Annual ACM Symposium on Theory of Computing (STOC'82)*. Association for Computing Machinery, New York, NY, 365–377. https://doi.org/10.1145/800070.802212

[46] Jens Groth. 2005. Non-interactive zero-knowledge arguments for voting. In *Applied Cryptography and Network Security*, John Ioannidis, Angelos Keromytis, and Moti Yung (Eds.). Springer, Berlin, 467–482.

[47] Shai Halevi and Victor Shoup. An Implementation of homomorphic encryption. Retrieved January 31, 2022, from https://github.com/shaih/HElib(2013).

[48] Mohammad Zahidul Hasan, Md Safiur Rahman Mahdi, Md Nazmus Sadat, and Noman Mohammed. 2018. Secure count query on encrypted genomic data. *Journal of Biomedical Informatics* 81 (2018), 41–52.

[49] Stephanie J. Heerema and Cees Dekker. 2016. Graphene nanodevices for DNA sequencing. *Nature Nanotechnology* 11, 2 (2016), 127.

[50] Bijit Hore, Sharad Mehrotra, and Gene Tsudik. 2004. A privacy-preserving index for range queries. In *Proceedings of the 30th International Conference on Very Large Data Bases-Volume 30*. VLDB Endowment, 720–731.

[51] Somesh Jha, Louis Kruger, and Vitaly Shmatikov. 2008. Towards practical privacy for genomic computation. In *2008 IEEE Symposium on Security and Privacy (SP'08)*. 216–230. https://doi.org/10.1109/SP.2008.34

[52] Murat Kantarcioglu, Wei Jiang, Ying Liu, and Bradley Malin. 2008. A cryptographic approach to securely share and query genomic sequences. *IEEE Transactions on Information Technology in Biomedicine* 12, 5 (2008), 606–617.

[53] Akinori Kawachi, Keisuke Tanaka, and Keita Xagawa. 2007. Multi-bit cryptosystems based on lattice problems. In *International Workshop on Public Key Cryptography*, 315–329. https://doi.org/10.1007/978-3-540-71677-8_21

[54] Hugo Krawczyk. Cryptographic Extraction and Key Derivation: The HKDF Scheme. Cryptology ePrint Archive, Report 2010/264, https://ia.cr/2010/264.

[55] Kristin Lauter, Adriana López-Alt, and Michael Naehrig. 2015. Private computation on encrypted genomic data. In *Progress in Cryptology (LATINCRYPT'14)*, Diego F. Aranha and Alfred Menezes (Eds.). Springer International Publishing, Cham, 3–27.

[56] Feifei Li, Marios Hadjieleftheriou, George Kollios, and Leonid Reyzin. 2006. Dynamic authenticated index structures for outsourced databases. In *Proceedings of the 2006 ACM SIGMOD International Conference on Management of Data*. ACM, 121–132.

[57] Helger Lipmaa. 2003. On diophantine complexity and statistical zero-knowledge arguments. In *Advances in Cryptology (ASIACRYPT'03)*, Chi-Sung Laih (Ed.). Springer, Berlin, 398–415.

[58] Helger Lipmaa, N. Asokan, and Valtteri Niemi. 2001. Secure Vickrey Auctions without Threshold Trust. (2001). http://eprint.iacr.org/2001/095. Published in Financial Cryptography 2002. helger@tcs.hut.fi 11810 received 13 Nov 2001, last revised 3 May 2002.

[59] Ie Lutsenko and I. V. Protasov. 2009. Sparse, thin and other subsets of groups. *International Journal of Algebra and Computation* 19, 4 (2009), 491–510.

[60] Adriana López-Alt, Eran Tromer, and Vinod Vaikuntanathan. 2012. On-the-fly multiparty computation on the cloud via multikey fully homomorphic encryption In. *Proceedings of the Annual ACM Symposium on Theory of Computing*. https://doi.org/10.1145/2213977.2214086

[61] Wojciech Makalowski. 2001. The human genome structure and organization. *Acta Biochimica Polonica* 48 (2001), 587–98.

[62] Wenbo Mao. 1998. Guaranteed correct sharing of integer factorization with off-line shareholders. In *International Workshop on Public Key Cryptography*. Springer, 60–71.

[63] Elaine R. Mardis. 2011. A decade's perspective on DNA sequencing technology. *Nature* 470, 7333 (2011), 198.

[64] Abukari Mohammed Yakubu and Yi-Ping Phoebe Chen. 2019. Ensuring privacy and security of genomic data and functionalities. *Briefings in Bioinformatics* 21, 2 (2019), 511–526. https://doi.org/10.1093/bib/bbz013 arXiv:https://academic.oup.com/bib/article-pdf/21/2/511/33585094/bbz013.pdf.

[65] Eduardo Morais, Tommy Koens, Cees Wijk, and Aleksei Koren. 2019. A Survey on Zero Knowledge Range Proofs and Applications.

[66] David Naccache and Jacques Stern. 1998. A new public key cryptosystem based on higher residues. In *Proceedings of the 5th ACM Conference on Computer and Communications Security (CCS'98)*. Association for Computing Machinery, New York, NY, 59–66. https://doi.org/10.1145/288090.288106

[67] Muhammad Naveed, Erman Ayday, Ellen W. Clayton, Jacques Fellay, Carl A. Gunter, Jean-Pierre Hubaux, Bradley A. Malin, and XiaoFeng Wang. 2015. Privacy in the genomic era. *ACM Computing Surveys (CSUR)* 48, 1 (2015), 6.

[68] Tatsuaki Okamoto and Shigenori Uchiyama. 1998. A new public-key cryptosystem as secure as factoring. In *Advances in Cryptology (EUROCRYPT'98)*, Kaisa Nyberg (Ed.). Springer, Berlin, 308–318.

[69] Pascal Paillier. 1999. Public-key cryptosystems based on composite degree residuosity classes. In *International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 223–238.

[70] HweeHwa Pang, Arpit Jain, Krithi Ramamritham, and Kian-Lee Tan. 2005. Verifying completeness of relational query results in data publishing. In *Proceedings of the 2005 ACM SIGMOD International Conference on Management of Data*. ACM, 407–418.

[71] Torben Pryds Pedersen. 1991. Non-interactive and information-theoretic secure verifiable secret sharing. In *Annual International Cryptology Conference*. Springer, 129–140.

[72] H. Perl, Y. Mohammed, M. Brenner, and M. Smith. 2012. Fast confidential search for bio-medical data using Bloom filters and Homomorphic Cryptography. In *2012 IEEE 8th International Conference on E-Science (e-Science'12)*. IEEE Computer Society, Los Alamitos, CA, 1–8. https://doi.org/10.1109/eScience.2012.6404484

[73] R. L. Rivest, A. Shamir, and L. Adleman. 1978. A method for obtaining digital signatures and public-key cryptosystems. *Commun. ACM* 21, 2 (Feb. 1978), 120–126. https://doi.org/10.1145/359340.359342

[74] International HIV Controllers Study and others. 2010. The major genetic determinants of HIV-1 control affect HLA class I peptide presentation. *Science (New York, NY)* 330, 6010 (2010), 1551.

[75] Juan Ramón Troncoso-Pastoriza, Stefan Katzenbeisser, and Mehmet Celik. 2007. Privacy preserving error resilient DNA searching through oblivious automata. In *Proceedings of the 14th ACM Conference on Computer and Communications Security*. ACM, 519–528.

[76] Zhiyu Wan, James W. Hazel, Ellen Wright Clayton, Yevgeniy Vorobeychik, Murat Kantarcioglu, and Bradley A. Malin. 2022. Sociotechnical safeguards for genomic data privacy. *Nature Reviews Genetics* 23, 7 (July 2022), 429–445.

[77] Shuang Wang, Yuchen Zhang, Wenrui Dai, Kristin Lauter, Miran Kim, Yuzhe Tang, Hongkai Xiong, and Xiaoqian Jiang. 2015. HEALER: Homomorphic computation of ExAct logistic rEgRession for secure rare disease variants analysis in GWAS. *Bioinformatics* 32, 2 (2015), 211–218. https://doi.org/10.1093/bioinformatics/btv563

[78] Xiao Shaun Wang, Yan Huang, Yongan Zhao, Haixu Tang, XiaoFeng Wang, and Diyue Bu. 2015. Efficient genome-wide, privacy-preserving similar patient query based on private edit distance. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security (CCS'15)*. Association for Computing Machinery, New York, NY, 492–503. https://doi.org/10.1145/2810103.2813725

[79] Andrew Chi-Chih Yao. 1986. How to generate and exchange secrets. In *27th Annual Symposium on Foundations of Computer Science (SFCS'86)*. IEEE, 162–167.

[80] Luděk Zajíček. 1983. Differentiability of the distance function and points of multi-valuedness of the metric projection in Banach space. *Czechoslovak Mathematical Journal* 33, 2 (1983), 292–308. http://eudml.org/doc/13383.

[81] Yihua Zhang, Marina Blanton, and Ghada Almashaqbeh. 2015. Secure distributed genome analysis for GWAS and sequence comparison computation. *BMC Medical Informatics and Decision Making* 15 (2015), S4. https://doi.org/10.1186/1472-6947-15-S5-S4

[82] Yuchen Zhang, Wenrui Dai, Xiaoqian Jiang, Hongkai Xiong, and Shuang Wang. 2015. FORESEE: Fully outsourced secuRe gEnome study basEd on homomorphic encryption. *BMC Medical Informatics and Decision Making* 15 (2015), S5. https://doi.org/10.1186/1472-6947-15-S5-S5

Received 20 October 2021; revised 7 November 2022; accepted 28 November 2022