electronics

MDPI

Article

A Secure and Efficient Dynamic Analysis Scheme for Genome Data within SGX-Assisted Servers

Bao Li, Fucai Zhou *, Qiang Wang and Da Feng

Software College, Northeastern University, Shenyang 110169, China; luckybao@stumail.neu.edu.cn (B.L.); wangqiang1@mail.neu.edu.cn (Q.W.); dafeng@stumail.neu.edu.cn (D.F.)
* Correspondence: fczhou@mail.neu.edu.cn; Tel.: +86-139-4041-3064

Abstract: With the rapid development of the Internet of Things (IoT), more and more user devices access the network and generate large amounts of genome data. These genome data possess significant medical value when researched. However, traditional genome analysis confronts security and efficiency challenges, including access pattern leakage, low efficiency, and single analysis methods. Thus, we propose a secure and efficient dynamic analysis scheme for genome data within a Software Guard Extension (SGX)-assisted server, called SEDASGX. Our approach involves designing a secure analysis framework based on SGXs and implementing various analysis methods within the enclave. The access pattern of genome data is always obfuscated during the analysis and update process, ensuring privacy and security. Furthermore, our scheme not only achieves higher analysis efficiency but also enables dynamic updating of genome data. Our results indicate that the SEDASGX analysis method is nearly 2.5 times more efficient than non-SGX methods, significantly enhancing the analysis speed of large-scale genome data.

Keywords: Intel SGX; security and privacy; data analysis; ORAM; IoT

Citation: Li, B.; Zhou, F.; Wang, Q.; Feng, D. A Secure and Efficient Dynamic Analysis Scheme for Genome Data within SGX-Assisted Servers. *Electronics* **2023**, *12*, 5004. https://doi.org/10.3390/electronics12245004

Academic Editor: Domenico Rosaci

Received: 14 November 2023
Revised: 11 December 2023
Accepted: 12 December 2023
Published: 14 December 2023

1. Introduction

Cloud computing platforms [1] offer elastic storage space and stronger computing power for gene data. With the rising development of e-healthcare technologies, the pool of genetic data collected from distributed healthcare devices and centers is growing explosively. Thus, the genetic data collected will be exposed and distributed among multiple healthcare devices or centers. However, genomes can range anywhere from 4000 bases to 670 Gb, and involve important personal privacy. For example, humans have two copies of their inherited genome of 3.2 Gb each. Genomes are stored in VCF file format. VCF is one of the important file formats in the biomedical domain because of its critical role in describing DNA and RNA variants. VCF can describe single- and multi-nucleotide polymorphisms (SNPs and MNPs), insertions and deletions (INDELs), and simple structural variants (SVs) against a reference genome [2]. The most common mutation in the human population is called single nucleotide polymorphism (SNP). It is the variation in a single nucleotide at a particular position of the genome. There are about 5 million SNPs observed per individual, and sensitive information about individuals (such as disease predispositions) are typically inferred by analyzing the SNPs. How to securely share genome data and efficiently analyze them in the IoT environment is needed to solve the problem of information islands [3]. Therefore, the designed scheme must not only ensure the privacy and security of genome data, but also ensure the security and efficiency of the genome data analysis process. The main reason is that personal genome data can carry sensitive information, including information that can reveal the identity of the owner [4] and even facial features [5]. For example, Claes et al. have developed a 3D model of human faces based on gender, ancestral genomes, and facial features [5], highlighting the potential risks of sharing sensitive genetic data.

Unfortunately, while traditional encryption algorithms, such as homomorphic encryption (HE) [6–9] and secure multi-party computing (SMC) [10–12] can ensure the confiden-

tiality of gene data, they cannot be applied to massive genome data scenarios due to high computational overheads and low computational efficiency. The emergence of trusted execution environments (TEEs), such as Intel SGX, has made it possible to operate a device with genome data in a trusted, isolated region called an enclave. Thus, the TEE brings neither high computational overheads nor restrictions based on software technology and makes it possible to securely and efficiently analyze massive genome data.

Nevertheless, existing schemes have various drawbacks. Most of the traditional genetic data analysis schemes [6–12] are based on homomorphic encryption and secure multi-party computation. These schemes suffer from the problem that the communication cost between the client and the server is too high. In addition, the practical application of homomorphic encryption and secure multi-party computing technology still has the problem of low efficiency in computing large-scale data. The emergence of trusted execution environments has brought about a turning point in the above problems. While the emergence of a trusted execution environment can to some extent alleviate the efficiency issues in large-scale genetic data analysis, many schemes [13–15] based on trusted execution environments still have shortcomings in aspects such as data access patterns, single-gene data analysis methods, dynamic updates, and multi-user access control. For example, Chen et al. first proposed a secure outsourcing genetic testing framework based on SGX in [13]. Mandal et al. built a practical, private data oblivious genome variants search using Intel SGX in [14]. Can et al. proposed a hardware–software hybrid approach SkSES to perform statistical tests on genomic data presented as VCF files from different countries in [15]. Previous schemes have not addressed a series of issues such as data access pattern leakage, single analysis methods, and dynamic data update during multi-user analysis in IoT scenarios. Thus, we propose a secure and efficient dynamic analysis scheme for genome data within an SGX-assisted server. The main contributions of SEDASGX are summarized as follows.

- SEDASGX provides a multi-party genome data analysis architecture for edge computing scenarios based on Intel SGX. This architecture uses the AES-GCM algorithm and the attributes of SGX to ensure the confidentiality and integrity of the code, genetic data, and analysis results. In addition, even if the terminal device does not require hardware support, it still meets the needs of users for uploading and analysis, reducing the hardware requirements of end users.
- SEDASGX is used to construct an oblivious data storage structure based on Path ORAM to avoid the leakage of access patterns of genome data in the analysis and update process, and encrypts them with a tamper-proof encryption algorithm (ASE-GCM) to guarantee the confidentiality and integrity of gene data, as well as the correctness of the analysis results.
- SEDASGX utilizes various genome analysis methods and dynamic updates. Additionally, the identity encryption technology based on the SGX security analysis architecture ensures that during the analysis process users cannot obtain each other's analysis results.

2. Related Work

Privacy of genomic data has recently become a very hot research topic. Several privacy-preserving schemes have been proposed for processing of genomic data in different secure aspects. In the following, we present the detailed classifications of state-of-the-art work.

- Homomorphic encryption-based schemes. Kim et al. [6] used homomorphic encryption technology to encrypt a DNA sequence and conduct a secure evaluation of χ^2 distribution over the encrypted data. Sarkar et al. [7] proposed a privacy-preserving genotype imputation using machine learning and a Paillier homomorphic encryption. Wang et al. [8] designed a homomorphic exact logistic regression model algorithm aiming at reducing the computational and storage costs. Blatt et al. [9] presented a privacy-preserving framework based on several advances in homomorphic encryption and demonstrated that it can perform an accurate GWAS analysis for a real dataset of

more than 25,000 individuals, keeping all individual's data encrypted and requiring no user interactions.

- Secure multi-party computation (SMC)-based schemes. Kamm et al. [10] proposed secretly sharing the sensitive data among several parties and computing GWAS over the distributed data. Dong et al. [11] proposed a secure and efficient GWAS scheme. Zhu et al. [12] proposed a privacy-preserving framework for conducting genome-wide association studies over outsourced patient data.
- Hardware-based schemes. Chen et al. [13] presented one of the first implementations of a Software Guard Extension (SGX)-based securely outsourced genetic testing framework, which leveraged multiple cryptographic protocols and a minimal perfect hash scheme to enable efficient and secure data storage and computation outsourcing. Mandal et al. [14] built a memory oblivious structure to search genome variants using Intel SGX. Kockan et al. [15] proposed SkSES, which employs sketch algorithm, data compression, and population stratification reduction methods to reduce the memory consumption.

3. Preliminaries

3.1. Intel SGX

Intel SGX [16–20] is a set of new instructions and modifications to the memory access architecture of Intel CPUs. Figure 1 summarizes the main features of SGX: memory isolation and remote attestation.

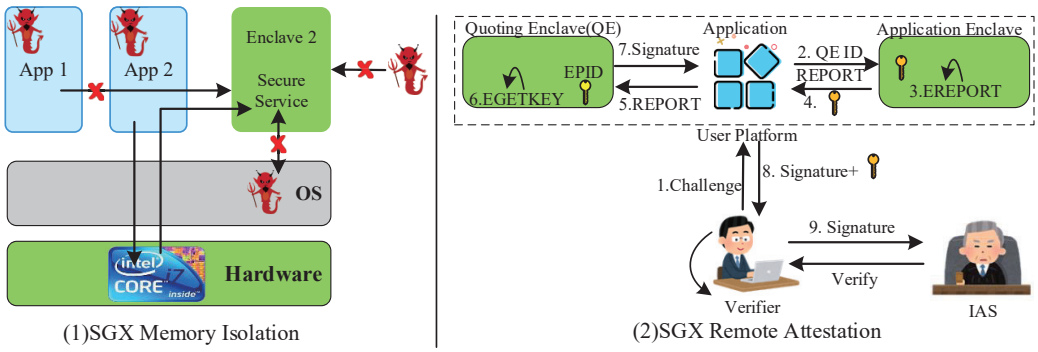

Figure 1. Intel SGX Features.

- Memory Isolation. When a program runs on the SGX-enabled platform, it is divided into two parts: an untrusted storage region and a trusted isolated region (enclave). An enclave is a separate block of physical RAM that cannot be accessed by other applications, privileged software, OS, hypervisor, or the firmware on the system in Figure 1(1). Meanwhile, the enclave is used to protect sensitive data and codes. When an SGX program is hung or closed, the untrusted storage region mainly is utilized to store the encrypted sensitive data separated from the enclave.
- Remote Attestation. SGX provides a cryptographic verification that an enclave is running securely on a remote server platform. When an enclave is created, an app enclave generates a set of claims (i.e., Key and REPORT), and an SGX component QE (quoting Enclave) generates an attestation signature of the report by using the EGETKEY instruction in Figure 1((2)-6). QE returns the signature to the App in Figure 1((2)-7). After that, the App sends the signature and key to the verifier in Figure 1((2)-8). Finally, the verifier checks the signature using the Intel attestation services (IASs). In particular, QE only accepts measurements from trusted hardware, and the hardware guarantees that only enclaves that have been correctly created can be measured. Furthermore, a secure channel between the enclave and the client can be established via ECDH [21] and ECDSA [22]. This trusted channel is used to share the secret between the enclave and the client.

3.2. Oblivious RAM

Oblivious RAM (ORAM) algorithms allow a user to hide the access pattern of data that are accessed on a remote server by continuously shuffling and re-encrypting them. An adversary can observe the physical locations of the data accessed, but the ORAM algorithms ensure that the adversary learns nothing about the true access pattern during frequent accesses between the enclave and the storage. Next, we give a definition of the access pattern, and more details are given in [23].

Definition 1 (Access Pattern)**.** *Let* $\vec{x}$*=(*q_1*,* q_2*,...,*q_n*) denote a data effective request sequence of length n, where* q_i*=(*op_i*,* id_i*,* $data_i$*),* op_i *denotes a read* id_i *or a write* $(id_i, data_i)$ *operation. Additionally,* id_i *represents the identifier of a data block, and* $data_i$ *represents gene data written into a data block with an identifier* id_i*.*

3.3. Genome Analysis Methods

- Chisquare statistics are mainly used to study the relation of gene mutations in genome data. Because human beings are diploid and have two copies of all (non-sex)chromosomes, each person will have either the genotypes aa, ab(ba), or bb for each locus. For example, we sample the genomes of N individuals for particular single nucleotide variants (SNV), of which some have a particular disease (cases), and the rest do not (controls). The genotype aa, ab(ba), and bb represent 0, 1, and 2, respectively. Thus, person genomes can be represented by a vector g$\in\{0,1,2\}^N$. The i-th entry corresponds to the number of transcripts the person i has of a allele at locus j. Let y$\in\{0,1\}^N$ be a vector that represents the gene mutation state of a person $y_i = 1$ if the i-th person has the disease, and $y_i = 0$ if he/she does not. More details are described in [24]. As we will see, these values in Table 1 are sufficient to compute the χ^2 statistic using the following Equation (1).

$$\chi^2 = \sum_{i\in\{0,1\}} \sum_{j\in\{0,1\}} \frac{(m_{ij} - c_j \times \frac{r_i}{2N})^2}{c_j \times \frac{r_i}{2N}} \tag{1}$$

Table 1. Genotype table.

Genotype	a	b	Sum
Cases (y_i = 1)	m_{00}	m_{01}	r_0
Controls (y_i = 0)	m_{10}	m_{11}	r_1
Sum	c_0	c_1	$2N$

- Fisher's exact test [25] is a statistical test used to determine whether there is a nonrandom association between two categorical variables. It is generally used to determine whether a gene locus is statistically associated with a factor, which is more accurate than the Chisquare test. As preparation for extending to R $\times$ C contingency tables, the cell counts in 2 $\times$ 2 tables are denoted by $\{m_{ij}\}$ for $i = 0, 1$ and $j = 0, 1$. Given the above Table 1, a more general formula is as follows (2). In simple terms, $i = 0, 1$ and $j = 0, 1$ are extended to $i = 0, 1,\ldots, R$ and $j = 0, 1,\ldots, C$, followed by the row margins $\{c_0, c_1,\ldots, c_R\}$ and the column margins $\{r_0, r_1,\ldots, r_C\}$. Meanwhile, the formula is extended to (3).

$$p = \frac{c_0!c_1!r_0!r_1!}{(2N)!m_{00}!m_{01}!m_{10}!m_{11}!} \tag{2}$$

$$p = \frac{\prod_i (c_i!) \prod_j (r_j!)}{(2N)! \prod_i \prod_j (m_{ij}!)} \tag{3}$$

- Logistic regression [26] methods are commonly used in statistical analysis. They are also applied to genetic association studies due to the detection demand of massive genetic marker predictor variables, e.g., case/control status. Given a dichotomous

phenotype vector Y of m observations, and a matrix of single nucleotide polymorphism (SNP) genotypes X, let $p = \mathrm{P}(Y = 1|X = x)$. The likelihood function is:

$$L = \prod_{Y=1} p \prod_{Y=0} (1 - p) \tag{4}$$

where

$$p = \frac{1}{1 + e^{-(\alpha+\beta X)}} \tag{5}$$

and β is the vector of coefficients.

3.4. Identity-Based Encryption (IBE)

The formal notion of an identity-based encryption scheme was developed in [27]. An IBE scheme Π contains four algorithms: Setup, KeyGen, Enc, and Dec.

- Setup$(1^\lambda) \rightarrow (pk, msk)$. This algorithm takes as input a security parameter λ. It outputs the public parameter pk and a private master key msk.
- KeyGen$(pk, msk, id) \rightarrow sk_{id}$. This algorithm takes as input the public key pk, the private master key msk, and an identity id. It outputs private key sk_{id} of id.
- Enc$(sk_{id}, m) \rightarrow C$. This algorithm takes as input a public key pk and a message m. It outputs the ciphertext C for an identity id.
- Dec$(sk_{id}, C) \rightarrow m$. This algorithm takes as input a private key sk_{id} and the ciphertext C. It outputs the message m.

4. SEDASGX Scheme

A secure and efficient dynamic analysis scheme for genome data within SGX-assisted is composed of seven polynomial time calculations, namely SEDASGX = (Setup, Enc, Preprocess, Init, Analysis, Dec, Update). In this section, we will show the system model, notations and definitions, and constructions.

4.1. System Model

As shown in Figure 2, SEDASGX consists of five entities (i.e., analysis users ($\mathcal{AU}$), patients ($\mathcal{P}$), edge server ($\mathcal{ES}$), cloud server ($\mathcal{CS}$), and authority ($\mathcal{AUT}$)), and their respective tasks are described as follows.

- The $\mathcal{AU}$ generates encrypted analysis queries and sends them to the $\mathcal{CS}$. Additionally, the $\mathcal{AU}$ decrypts the query results.
- The $\mathcal{P}$ encrypts genome data and uploads them to the $\mathcal{ES}$. Meanwhile, the $\mathcal{P}$ sends the update queries to the $\mathcal{ES}$.
- The $\mathcal{ES}$ is divided into two parts: enclave and storage region. The enclave preprocesses all genomic data within the jurisdiction. After that, the enclave encrypts the processed data and sends them to the $\mathcal{CS}$. The storage region mainly stores source data.
- The $\mathcal{CS}$ is divided into two parts: enclave and storage region. The enclave performs initialization operation, genome analysis operation, and update operation. The storage region mainly stores all encrypted data and oblivious storage structure.
- The $\mathcal{AUT}$ is primarily responsible for remotely verifying the trusted execution environment of all edge servers and cloud services. Furthermore, the $\mathcal{AUT}$ is also responsible for generating keys for each entity within the system and distributing them through secure channels.

As shown in Figure 2, the $\mathcal{AUT}$ executes a setup operation to generate system master key pairs and secret key for each entity, and builds the secure channel by performing remote attestation with the $\mathcal{CS}$ and each $\mathcal{ES}$ in step (1). In step (2), the $\mathcal{P}$ encrypts genome data and uploads them to the $\mathcal{ES}$. The $\mathcal{ES}$ receives all genome data within its jurisdiction and performs a preprocessing operation within the enclave in step (3). Then, each $\mathcal{ES}$ encrypts these processed genome data and sends them to the $\mathcal{CS}$. After that, in step (4), the enclave uses the processed genome data to construct some oblivious data structures, a position map table, and a stash using an initialization operation. Then, the $\mathcal{AU}$ generates

encrypted analysis queries and sends them to the enclave. The enclave decrypts these encrypted analysis (update) queries and performs genome analysis (update) via loading these oblivious data structures in step (5). The $\mathcal{AU}$ receives the encrypted analysis results returned by the $\mathcal{CS}$ and performs decryption operations in step (6). Finally, the $\mathcal{P}$ sends the update to the $\mathcal{ES}$ in step (7).

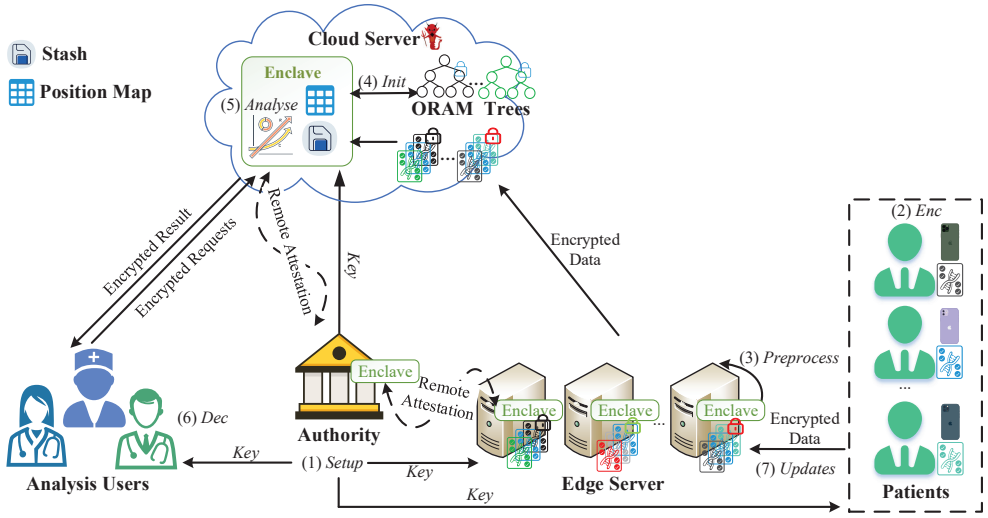

Figure 2. System Model.

4.2. Notations and Definitions

We summarize some notations used in SEDASGX in Table 2 and define two security definitions as follows.

Table 2. Notations.

Notations	Descriptions
λ, $\mathbb{G}$, g, H	Security parameter, group, the generator group $\mathbb{G}$, and hash function
ine pk, msk, sk_E, sk_i	Public key, master key, ORAM tree structure key, the data key of i-th $\mathcal{ES}$
ine sk_k, sk_{i_j}, m_{i_j}	The query key of k-th $\mathcal{AU}$, the data key of $\mathcal{P}$, the j-th data under the i-th $\mathcal{ES}$
ine iv_{i_j}, tag_{i_j}	The j-th initial vector under the i-th $\mathcal{ES}$, the j-th tag in the i-th $\mathcal{ES}$
ine add_{i_j}, $Size$	The j-th additional information under the i-th $\mathcal{ES}$, preset block size
ine $\mathcal{C}_{i_j}$, $\mathcal{M}_{i_{\eta_i}}$	The j-th ciphertext in the i-th $\mathcal{ES}$, the η_i-th data block under the i-th $\mathcal{ES}$
ine Z, pos, n	The node capacity of ORAM tree, genome locus, the i-th nation(i-th $\mathcal{ES}$)
ine N, M, pa, id	The number of $\mathcal{ES}$, the number of $\mathcal{P}$, the path of ORAM tree, block identifier
ine $\mathcal{T}_i$, $\mathcal{PM}_i$, $\mathcal{S}_i$	The i-th ORAM tree, the i-th position map table, the i-th stash
ine η_i, $\mathcal{N}_i$	The sum number of data blocks under i-th $\mathcal{ES}$, the size of $\mathcal{T}_i$
ine $\mathcal{L}_i$, $psize_i$, $ssize_i$	The high of ORAM tree, the size of $\mathcal{PM}_i$, the size of $\mathcal{S}_i$
ine $maxpa$, st	The maximum path of $\mathcal{T}_i$, the state of data block(True:1 and false:0)

Definition 2 (Correctness). *The SEDASGX scheme is correct if the following holds: First, $\mathcal{AUT}$ runs the Setup algorithm to generate public key pk and master key msk. For the genome data $\mathcal{M}_{i_j}$,$1\leq i\leq N$,$1\leq j\leq M$, $\mathcal{P}$ executes the Enc algorithm to generate encrypted genome data $\mathcal{C}_{i_j}$ and sends them to $\mathcal{ES}$. Then, the enclave of the $\mathcal{ES}$ performs the Preprocess algorithm to generate fixed-size data blocks $\mathcal{M}_{i_{\eta_i}}$ and encrypts $\mathcal{M}_{i_{\eta_i}}$ to $\mathcal{C}_{i_{\eta_i}}$. After that, the enclave of the $\mathcal{CS}$ calls the Init algorithm to generate encrypted ORAM trees $\mathcal{T}_i$, position map tables $\mathcal{PM}_i$, and stashes $\mathcal{S}_i$ by using $\mathcal{C}_{i_{\eta_i}}$. Given an analysis request $\mathcal{C}_q$, the ciphertext analysis result $\mathcal{C}_R$ can always decrypt into the corresponding plaintext analysis result and can be successfully verified by the Dec algorithm (AES-GCM).*

Definition 3 (Query Unlinkability). *Let $\overrightarrow{q} = (q_1, q_2, \ldots, q_n)$ denote a set of analysis sequences with the same key and length. If any two analysis queries q_i and q_j are computationally indistinguishable, the query pattern of the SEDASGX is secure.*

4.3. Constructions

We now give the detailed construction of each algorithm in the SEDASGX.

Setup(1^λ)$\rightarrow$(pk, msk, sk_{i_j}, sk_i, sk_k, sk_E). Taking a security parameter λ as input, the enclave of the $\mathcal{AUT}$ generates a group $\mathbb{G}$ with order p, where g is a random generator of $\mathbb{G}$. The enclave picks up a $\alpha \in_R \mathbb{Z}_p$, and selects a collision-resistant hash function H:$\{0,1\}^* \rightarrow \mathbb{G}$. Meanwhile, the $\mathcal{AUT}$ generates unique identities id_{i_j}, id_k, id_i, and id_E for $\mathcal{P}$, $\mathcal{AU}$, $\mathcal{ES}$, and $\mathcal{CS}$, and generates private keys $sk_{i_j} = H(id_{i_j})^\alpha$, $sk_k = H(id_k)^\alpha$, $sk_i = H(id_i)^\alpha$, and $sk_E = H(id_E)^\alpha$ based on the these unique identities respectively, $1 \leq i \leq N$, $1 \leq j \leq M$, and $1 \leq k \leq \varsigma$. Finally, the $\mathcal{AUT}$ publishes the public key $pk = (\mathbb{G}, p, g, H, id_{i_j}, id_i, id_k, id_E)$ of the system, and keeps the master key $msk = \alpha$ secret. Meanwhile, the $\mathcal{AUT}$ sends the private keys sk_{i_j}, sk_i, sk_k, and sk_E to the $\mathcal{P}$, the $\mathcal{ES}$, the $\mathcal{AU}$, and the $\mathcal{CS}$ by the secure channel respectively.

Enc(m_{i_j}, iv_{i_j}, aad_{i_j}, sk_{i_j})$\rightarrow$($\mathcal{C}_{i_j}$, tag_{i_j}). This algorithm takes as inputs data m_{i_j}, the initial vertor iv_{i_j}, the additional authentication data add_{i_j}, and the data key sk_{i_j} as input. It outputs ciphertext $\mathcal{C}_{i_j}$ and the encrypted tag tag_{i_j}. Then, each $\mathcal{P}$ uploads the $\mathcal{C}_{i_j}$ and the tag_{i_j} to the $\mathcal{ES}$. Note that the ***Enc*** is an AES-GCM encryption algorithm.

Preprocess(m_{i_j}, $Size$)$\rightarrow \mathcal{M}_{i_{\eta_i}}$. This algorithm takes genome data m_{i_j} and a predetermined size $Size$ as input. It outputs a set of fixed-size blocks of data $\mathcal{M}_{i_{\eta_i}}$, where i denotes the i-th edge server and η_i represents the total number of data blocks preprocessed by the i-th the $\mathcal{ES}$, $1 \leq j \leq M$, $1 \leq i \leq N$. In particular, $\eta_i = \Sigma_{j=1}^{M} \eta_{i_j}$, j represents the j-th patient under the i-th $\mathcal{ES}$, and η_{i_j} represents the number of data blocks after the m_{i_j} is split. Firstly, the enclave of each $\mathcal{ES}$ decrypts $\mathcal{C}_{i_j}$ and divides all genome data m_{i_j} into fixed-size data block $\mathcal{M}_{i_{\eta_i}}$, recursively. If the last remaining data point is not sufficient to meet the predetermined size, it needs to be randomly filled to reach the required size. Finally, the enclave encrypts $\mathcal{M}_{i_{\eta_i}}$ to $\mathcal{C}_{i_{\eta_i}}$ and sends them to the $\mathcal{CS}$.

Init($\mathcal{C}_{i_{\eta_i}}$, sk_E, Z, sk_i)$\rightarrow$($\mathcal{T}_i$, $\mathcal{PM}_i$, $\mathcal{S}_i$). This algorithm takes all encrypted data blocks $\mathcal{C}_{i_{\eta_i}}$, the ORAM tree structure key sk_E, the node capacity Z, and the data key sk_i of the $\mathcal{ES}$ as input. It outputs position map tables $\mathcal{PM}_i$, stashes $\mathcal{S}_i$, and encrypted ORAM trees $\mathcal{T}_i$, $1 \leq i \leq N$, $1 \leq j \leq M$, according to Algorithm 1.

- First, the enclave of the $\mathcal{CS}$ calculates the sum η_i of all data blocks in each $\mathcal{ES}$. Then, enclave computes $\mathcal{N}_i$, $\mathcal{L}_i$, $psize_i$, and $sszie_i$ using the Equations (6) and (7), where Pow is a function that finds the power of 2 closest to η_i.

$$\eta_i = \Sigma_{j=1}^{M} \eta_{i_j}; \mathcal{N}_i = Pow(\eta_i); \mathcal{L}_i = log_2(\mathcal{N}_i + 1) - 1 \tag{6}$$

$$psize_i = \mathcal{N}_i * Z; sszie_i = (\mathcal{L}_i + 1) * Z \tag{7}$$

- Second, the enclave creates the position map tables $\mathcal{PM}_i$ and stashes $\mathcal{S}_i$ according to $psize_i$ and $ssize_i$. Meanwhile, the enclave randomly generates $ssize_i$ dummy genome data blocks and encrypts them according to sk_E. Then, the enclave writes them to the ORAM tree $\mathcal{T}_i$ according to $\mathcal{P}_i$.
- Finally, the enclave computes sk_i by the α and utilizes them to decrypt $\mathcal{C}_{i_{\eta_i}}$ to obtain $\mathcal{M}_{i_{\eta_i}}$. The enclave re-encrypts $\mathcal{M}_{i_{\eta_i}}$ to $\mathcal{C}^*_{i_{\eta_i}}$ in the $\mathcal{S}_i$ and writes them to $\mathcal{T}_i$ according to updated $\mathcal{P}_i$.

Query(sk_k, q)$\rightarrow \mathcal{C}_q$. This algorithm takes as inputs a query key sk_k of k-th analysis user and an analysis query q. It outputs the encrypted query $\mathcal{C}_q$.

- To generate a tailored analysis query $q = H(\mu||\nu||pos||fisher)$, the k-th $\mathcal{AU}$ first selects the data from various $\mathcal{ES}$ (μ and ν, $1 \leq \mu, \nu \leq N$), a certain gene locus pos, and a certain analysis method $fisher$, *Chi-square*, or LR based on their analysis requirements.
- Subsequently, the $\mathcal{AU}$ employs their query key sk_k to generate an encrypted analysis query $\mathcal{C}_q$ using the Enc algorithm, and sends $\mathcal{C}_q$ to the $\mathcal{CS}$.

Algorithm 1: Init Algorithm

Input: ORAM structure key sk_E, encrypted data blocks $\mathcal{C}_{i_{\eta_i}}$, data key sk_i of $\mathcal{ES}$, node capacity Z.

Output: The position map $\mathcal{PM}_i$, stashes $\mathcal{S}_i$, and encrypted ORAM trees $\mathcal{T}_i$, $1 \leq i \leq N$.

Enclave:

Decrypt $\mathcal{C}_{i_{\eta_i}}$ to obtain $\mathcal{M}_{i_{\eta_i}}$, $\eta_i = \Sigma_{j=1}^{M} \eta_{i_j}$, $1 \leq i \leq N$, $1 \leq j \leq M$;

For $1 \leq i \leq N$ **do**

  Compute $\mathcal{N}_i = \mathbf{Pow}(\eta_i)$, $\mathcal{L}_i = log_2(\mathcal{N}_i + 1) - 1$, $sszie_i = (\mathcal{L}_i + 1) * Z$, $psize_i = \mathcal{N}_i * Z$, $maxpa = 2^{\mathcal{L}_i}$;

  Initialize an $\mathcal{S}_i = \{c_i, id_i, pos_i, pa_i, n_i, st_i\}$ of size $ssize_i$ and $\mathcal{PM}_i = \{id_i, pos_i, pa_i, n_i\}$ of size $psize_i$;

  For $1 \leq t \leq \beta_i = \lceil psize_i / sszie_i \rceil * ssize_i$ **do**

    Generate a dummy block m'_{i_t}, $id_{i_t} = t$, $pa_{i_t} = Random(2^{\mathcal{L}_i})$, $pos_{i_t} = random$, and $n_{i_t} = i$;

    Copy m'_{i_t}, id_{i_t}, pa_{i_t}, pos_{i_t}, and n_{i_t} to $\mathcal{S}_i.c_{i_t}$, $\mathcal{S}_i.id_{i_t}$, $\mathcal{S}_i.pa_{i_t}$, $\mathcal{S}_i.pos_{i_t}$, $\mathcal{S}_i.n_{i_t}$ and set $\mathcal{S}_i.st_{i_t} = 0$;

    IF $t \mod ssize_i = 0$

      Encrypt blocks in $\mathcal{S}_i$ and write them to $\mathcal{T}_i$ by $\mathcal{PM}_i$ and empty $\mathcal{S}_i$;

For $1 \leq i \leq N$ **do**

  Set $\beta_i = \lceil \eta_i / ssize_i \rceil * ssize_i$ and generate $\beta_i - \eta_i$ dummy blocks m'_{i_ι}, $\eta_i < \iota \leq \beta$

  For $1 \leq j \leq \beta_i$ **do**

    Copy j, $\mathcal{M}_{i_{\eta_i}}.pos$, $\mathcal{M}_{i_{\eta_i}}.n$, $\mathcal{M}_{i_{\eta_i}}.c$ to $\mathcal{S}_i.id_{i_j}$, $\mathcal{S}_i.pos_{i_j}$, $\mathcal{S}_i.n_{i_j}$, $\mathcal{S}_i.c_{i_j}$, and set $\mathcal{S}_i.st_{i_j} = 1$

      $\mathcal{S}_i.pa_{i_j} = \mathbf{Random}(maxpa)$;

    $\mathcal{PM}_i.id_{i_j} = j$, $\mathcal{PM}_i.pa_{i_j} = \mathcal{S}_i.pa_{i_j}$, $\mathcal{PM}_i.n_{i_j} = \mathcal{S}_i.n_{i_j}$, $\mathcal{PM}_i.pos_{i_j} = \mathcal{S}_i.pos_{i_j}$;

    IF $j \mod ssize_i = 0$

      Encrypt blocks in the $\mathcal{S}_i$ and write them to $\mathcal{T}_i$ by $\mathcal{PM}_i$, and empty $\mathcal{S}_i$;

Return ORAM tree $\mathcal{T}_i$, position map $\mathcal{PM}_i$, and stashes $\mathcal{S}_i$;

Analysis(sk_E, $\mathcal{T}_i$, $\mathcal{PM}_i$, $\mathcal{S}_i$, sk_k, $\mathcal{C}_q$)$\rightarrow\mathcal{C}_R$. This algorithm takes the ORAM tree structure key sk_E, the encrypted ORAM tree $\mathcal{T}_i$, the position map table $\mathcal{PM}_i$, the stash $\mathcal{S}_i$, the query key sk_k of k-th $\mathcal{AU}$, and an encrypted query $\mathcal{C}_q$ as inputs. It outputs the encrypted analysis result $\mathcal{C}_R$ according to the following Algorithm 2.

- First, the enclave decrypts the $\mathcal{C}_q$ to obtain μ, ν, pos, and $fisher$.
- Second, the enclave acquires its corresponding ORAM tree $\mathcal{T}_\mu$ and $\mathcal{T}_\nu$ based on the values of μ and ν, and subsequently read the data from the ORAM tree $\mathcal{T}_\mu$ and $\mathcal{T}_\nu$ to the stash $\mathcal{S}_\mu$ and $\mathcal{S}_\nu$ by utilizing the position map table $\mathcal{PM}_\mu$ and $\mathcal{PM}_\nu$, respectively.
- Finally, the enclave extracts the relevant information required for data analysis from the data blocks that have been read, and calls the corresponding analysis algorithm (e.g., $fisher$) to analyze the genome data according to the query request and obtain the corresponding analysis results R. After that, the enclave encrypts the R by using the corresponding query key sk_k and sends the $\mathcal{C}_R$ to the k-th $\mathcal{AU}$.

Algorithm 2: Analysis Algorithm

Input: Encrypted ORAM tree $\mathcal{T}_i$, structure key sk_E, query key sk_κ, encrypted analysis query $\mathcal{C}_q$.
Output: Encrypted analysis result $\mathcal{C}_R$.
$\underline{\mathcal{AU}}$:
Encrypt the analysis query q to $\mathcal{C}_q$ with query key sk_k;
$\underline{Enclave}$:
Decrypt $\mathcal{C}_q$ to $q = \mu||\nu||pos||fisher$ with sk_k, and read $\mathcal{PM}_\mu$ and $\mathcal{PM}_\nu$ to enclave;
For $1 \leq i \leq N$ **do**
  For $1 \leq j \leq \beta_i$ **do**
    Find $\mathcal{PM}_\mu.id_{\mu_j}$ and $\mathcal{PM}_\mu.pa_{\mu_j}$ corresponding to $\mathcal{PM}_\mu.pos_{\mu_j} = pos$;
    Read all blocks on $\mathcal{PM}_\mu.pa_{\mu_j}$ in $\mathcal{T}_\mu$ and decrypt them to $\mathcal{S}_\mu$;
  For $1 \leq \omega \leq ssize_i$ **do**
    IF $\mathcal{PM}_\mu.id_{\mu_j} = \mathcal{S}_\mu.id_{\mu_\omega}$
      Get $\mathcal{S}_\mu.c_{\mu_\omega}$, update $\mathcal{S}_\mu.pa_{\mu_\omega}$, and $\mathcal{PM}_\mu.pa_{\mu_j}$, and re-encrypt $\mathcal{S}_\mu.c_{\mu_\omega}$;
Repeat steps 3–10 to obtain $\mathcal{S}_\nu.c_{\nu_\omega}$ of ν;
Compute analysis result R according to $\mathcal{S}_\mu.c_{\mu_\omega}$ of μ and $\mathcal{S}_\nu.c_{\nu_\omega}$ of ν via Fisher's exact algorithm;
Encrypt analysis result R to $\mathcal{C}_R$ with sk_k and send it to the k-th $\mathcal{AU}$;
$\underline{\mathcal{AU}}$:
Decrypt encrypted analysis result $\mathcal{C}_R$ and verify correctness;

Dec($\mathcal{C}_R$, sk_k, tag_R)$\rightarrow R$. This algorithm takes as inputs an encrypted analysis result $\mathcal{C}_R$, a query key sk_k, and an encrypted tag tag_R. It outputs an analysis result R. In other words, the $\mathcal{AU}$ verifies and decrypts this analysis result $\mathcal{C}_R$ via the AES-GCM algorithm.

Update($\mathcal{T}_i$, $\mathcal{PM}_i$, $\mathcal{S}_i$, sk_E, $\mathcal{C}'_{i_t}$, sk_i)$\rightarrow$($\mathcal{T}'_i$, $\mathcal{PM}'_i$, $\mathcal{S}'_i$). This algorithm takes as inputs the ORAM tree $\mathcal{T}_i$, position map $\mathcal{PM}_i$, stash $\mathcal{S}_i$, ORAM tree structure key sk_E, updated encrypted data $\mathcal{C}'_{i_t}$, and data key sk_i. We are assuming that the updated data blocks have been preprocessed by the $\mathcal{ES}$ and already reside within the $\mathcal{CS}$. Furthermore, given the vast volume of genomic data, we will only address scenarios involving the modification of individual data blocks and the addition of a specific number of data blocks. See Algorithm 3 for details.

- Case 1: *modify a single data block*. First, the enclave decrypts the $\mathcal{C}'_{i_1}$ to obtain updated data, e.g., μ, pos, and $\mathcal{M}'_{i_1}$. Then, the enclave finds pos in the $\mathcal{PM}_\mu$ to obtain $\mathcal{PM}_\mu.pa_{\mu_j}$ and $\mathcal{PM}_\mu.id_{\mu_j}$. After that, the enclave loads all data blocks on the $\mathcal{PM}_\mu.pa_{\mu_j}$ in the $\mathcal{T}_\mu$ and decrypts them into the $\mathcal{S}_\mu$. Next, the enclave searches the $\mathcal{PM}_\mu.id_{\mu_j}$ on the $\mathcal{S}_\mu$ and replaces $\mathcal{S}_\mu.c_{\mu_j}$ with the updated content $\mathcal{M}'_{i_1}$. Meanwhile, it updates $\mathcal{S}_\mu.pa_{\mu_j}$ and copies it to $\mathcal{PM}_\mu.pa_{\mu_j}$. Finally, the enclave re-encrypts all data blocks in the $\mathcal{S}_\mu$ and re-writes them back to the $\mathcal{T}_\mu$ according to the updated $\mathcal{S}_\mu.pa_{\mu_j}$.
- Case 2: *add small data blocks*. Upon receipt of the updated data blocks $\mathcal{C}'_{i_t}$, $\eta_i \leq t \leq psize_i$ uploaded by the $\mathcal{ES}$, the enclave uses the sk_i to decrypt block by block. Then, the enclave generates the updated ORAM tree $\mathcal{T}'_i$.
- Case 3: *add massive data blocks*. This is performed after receiving the encrypted data blocks $\mathcal{C}'_{i_t}$, $psize_i - \eta_i \leq i \leq t$ uploaded by $\mathcal{ES}$. Because the actual number of genome data blocks in the storage region has exceeded the storage limit of the original ORAM trees, the enclave needs to call the Algorithm 1 to regenerate the new ORAM tree $\mathcal{T}'_i$.

Algorithm 3: Update Algorithm

Input: Encrypted ORAM tree $\mathcal{T}_i$, ORAM tree key sk_E, data key sk_i, encrypted update data blocks $\mathcal{C}'_{i_t}$,
the position map $\mathcal{PM}_i$, the stash $\mathcal{S}_i$.
Output: Updated ORAM trees $\mathcal{T}_i'$, updated position map tables $\mathcal{PM}'_i$, updated stashes $\mathcal{S}'_i$.
$\underline{\mathcal{ES}}$:
Preprocess update data to $\mathcal{C}'_{i_t}$, and upload them to the $\mathcal{CS}$;
$\underline{Enclave}$:
Decrypt $\mathcal{C}'_{i_t}$ to $\mathcal{M}'_{i_t}$, μ and pos_{i_t} with sk_i, and find the $\mathcal{PM}_\mu$;
For $1 \leq i \leq N$ **do**
  IF $t = 1$
    For $1 \leq j \leq \beta_i$
      Find $\mathcal{PM}_\mu.id_{\mu_j}$ and $\mathcal{P}_\mu.pa_{\mu_j}$ corresponding to $\mathcal{PM}_\mu.pos_{\mu_j} = pos_{i_t}$;
      Read all blocks on $\mathcal{PM}_\mu.pa_{\mu_j}$ in $\mathcal{T}_\mu$ and decrypt them to $\mathcal{S}_\mu$;
    For $1 \leq w \leq ssize_i$ **do**
      IF $\mathcal{PM}_\mu.id_{\mu_j} = \mathcal{S}_\mu.id_{\mu_\omega}$
        Update $\mathcal{S}_\mu.pa_{\mu_\omega}$, $\mathcal{S}_\mu.c_{\mu_\omega}$, and $\mathcal{PM}_\mu$;
  ELSE IF $1 < t \leq psize_i - \eta_i$
    For $1 \leq l \leq t$ **do**
      Update $\mathcal{PM}_\mu.id_{i_{\eta_i+l}} = \eta_i + l$, $\mathcal{PM}_\mu.pa_{i_{\eta_i+l}} = Ransom(2^{\mathcal{L}_i})$, $\mathcal{PM}_\mu.pos_{i_{\eta_i+l}} = pos_{i_t}$;
    For $1 \leq \xi \leq \lceil t/sszie_i \rceil$ **do**
      For $1 \leq \omega \leq ssize_i$ **do**
        Copy $\mathcal{PM}_\mu.id_{i_{\eta_i+l}}$,$\mathcal{PM}_\mu.pa_{i_{\eta_i+l}}$, M_{i_l} to $\mathcal{S}_\mu.id_{i_\omega}$, $\mathcal{S}_\mu.pa_{i_\omega}$, $\mathcal{S}_\mu.c_{i_\omega}$ and set $\mathcal{S}_\mu.st_{i_\omega}$=1;
      Encrypt $\mathcal{S}_\mu$ and write them to ORAM $\mathcal{T}_\mu$ by $\mathcal{S}_\mu.pa_{i_\omega}$, and empty $\mathcal{S}_\mu$;
  ELSE $t > psize_i - \eta_i$
    Regenerate new $\mathcal{T}'_\mu$, $\mathcal{PM}'_\mu$, and $\mathcal{S}'_\mu$ with the $\mathcal{C}_{\mu_{\eta_i}}$ and updated data blocks $\mathcal{C}'_{i_t}$;
Return $\mathcal{T}_i'$, $\mathcal{P}'_i$, and $\mathcal{S}'_i$

5. Security Analysis

Combining the threat model assumed in Figure 1(1) by Intel SGX itself, only the CPU can securely access data in the enclave. Thus, the adversary can impersonate cloud server administrators (or OS), other entities in the system, and external attackers.

5.1. Correctness

In SEDASGX, the $\mathcal{AU}$, $\mathcal{P}$, $\mathcal{AUT}$, and enclave are trusted, and they can execute all algorithms correctly. The genome data separated from the enclave are encrypted by the AES-GCM tamper-proof encryption algorithm and stored in the untrusted memory. The correctness of the SEDASGX relies on the integrity of the data stored in the untrusted memory. Thus, the correctness of the SEDASGX depends on the AES-GCM tamper-proof encryption algorithm. Fortunately, the correctness of the AES-GCM algorithm has been proved in [28].

5.2. Query Unlinkability

When the adversary is an external attacker, the adversary cannot obtain the key due to the memory isolation feature of Intel SGX. The adversary cannot obtain any plaintext information about the query without the key. Therefore, it is only necessary to prove that the adversary cannot distinguish any two queries.

Theorem 1. *SEDASGX can guarantee that the adversary cannot distinguish any two analysis queries that are generated from the same analysis content.*

Proof. Assuming that with the two analysis queries q_i and q_j, the user randomly selects different initial vectors iv_i, iv_j. Then, the user adopts the AES-GCM algorithm to encrypt q_i and q_j with

$$\begin{aligned} C_{q_i} &= Enc(S_C, iv_i, q_i, aad_i), \\ C_{q_j} &= Enc(S_C, iv_j, q_j, aad_j) \end{aligned} \tag{8}$$

The security of the AES-GCM encryption algorithm is based on a cryptographic conjecture that the block cipher is a secure pseudo-random permutation. Even the same analysis content will be encrypted into different ciphertext due to the randomness of the initial vector. Thus, C_{q_i} and C_{q_j} are computationally indistinguishable. □

5.3. Access Pattern

When the adversary is an administrator, SEDASGX uses the ORAM mechanism to avoid access pattern leakage caused by frequent access to memory due to genome analysis.

Theorem 2. *Let AP($\overrightarrow{x}$) represent the access pattern of the storage sequence for a given analysis query. An ORAM is secure if (1) for any two analysis queries of the same length, their access patterns AP($\overrightarrow{x}$) and AP($\overrightarrow{y}$) are computationally indistinguishable except user and enclave, and (2) the ORAM is correct in the case that returns on input $\overrightarrow{x}$ data that is consistent with $\overrightarrow{x}$ probability $\geq 1 - negl(|\overrightarrow{x}|)$, i.e., the ORAM may fail with probability $negl(|\overrightarrow{x}|)$.*

Proof. When a user sends an analysis request, the enclave loads all genome data blocks based on a certain path $\mathcal{PM}_i.pa_{i_{\eta_i}}$ in the ORAM tree $\mathcal{T}_i$ each time. To prove the security of ORAM, we assume that $Q = \{id_1, id_2, \ldots, id_{psize_i}\}$ is an block identifier sequence with size $psize_i$. Thus, the access pattern p observed by the adversary is as follows:

$$p = \{pos_{psize_i}[id_{psize_i}], ...; pos_1[id_1]\} \tag{9}$$

where $pos_\kappa[id_\kappa]$ is the position of κ-th genome data block on a certain path. Every data block is encrypted with the AES-GCM algorithm. Thus, any two access pattern sequences are computationally indistinguishable due to initial vector iv generated randomly. Moreover, $\mathcal{PM}_i$ is accompanied by an update during the enclave data loads every time, e.g., any two positions $pos_{\kappa_1}[id_{\kappa_1}]$ and $pos_{\kappa_2}[id_{\kappa_2}]$ are statistically independent of each other under $\kappa_1 < \kappa_2$ and $id_{\kappa_1} = id_{\kappa_2}$. Likewise, $pos_{\kappa_1}[id_{\kappa_1}]$ and $pos_{\kappa_2}[id_{\kappa_2}]$ are statistically independent of each other under $\kappa_1 < \kappa_2$ and $id_{\kappa_1} \neq id_{\kappa_2}$. Thus, we obtain the Equation (10) (by using Bayes rule).

$$\Pr(p) = \prod_{\kappa=1}^{psize_i} \Pr(pos_\kappa[id_\kappa]) = (\frac{1}{2^{\mathcal{L}_i}})^{psize_i} \tag{10}$$

This proves that AP($\overrightarrow{x}$) is computationally indistinguishable from a random sequence of bit strings. The correctness of the ORAM was proven in detail in [23]. □

5.4. CCA Security

Theorem 3. *If Π_E is a CPA secure encryption scheme, and Π_M is a message authentication code with a unique tag, then SEDASGX is a CCA secure encryption scheme.*

Proof. In the AES-GCM algorithm, plaintext data are encrypted using the AES-CTR mode, and then an authentication tag (MAC) is generated through GHASH, and finally, the ciphertext is obtained by XOR operation. Among them, the AES-CTR has been proved in [29] to satisfy CPA security.

Now suppose there exists an adversary, denoted as $\mathcal{A}$, who can distinguish the ciphertext. The $\mathcal{A}$ can choose two plaintexts m_0 and m_1 of the same length, $|m_0| = |m_1|$, and receives an encrypted ciphertext $c_b = Enc(sk, m_b)$, where sk is a symmetric key and b is 0 or 1, indicating the plaintext chosen by the $\mathcal{A}$. Therefore, $\mathcal{A}$'s goal is to infer b from c_b.

To achieve this goal, $\mathcal{A}$ can construct two valid authentication tags (MACs) with different GHASH values, and select one of these tags to attempt to match the encrypted ciphertext. However, since the GHASH function is collision-resistant, $\mathcal{A}$ cannot construct two valid tags with the same GHASH value. Therefore, we prove that the SEDASGX scheme for AES-GCM algorithm encryption with keys generated by the IBE scheme is CCA secure. □

6. Experiment Analysis

We show the experimental results from experimental analysis of SEDASGX and comparison of SEDASGX with a non-SGX server.

6.1. Implementation

We realize a series of experimental evaluations using a real-world genome dataset [30] to evaluate SEDASGX in terms of preprocessing, update operation, and analysis efficiency. The dataset we used is from the third phase of the 1000 Genomes Project in the UCSC Genome Browser and represents 2504 samples on GRCh37. The 1000 Genomes Project utilizes advanced DNA sequencing technologies to analyze the genomes of a diverse set of individuals from various ethnic backgrounds. The 1000 Genomes Project dataset includes the following features: sample diversity, whole genome sequencing, data accessibility, data quality control, and clinical and population genetics applications.

We implemented SEDASGX in C/C++ and Python codes on real SGX hardware, and used Intel SGX SDK 2.11 version library and SGX-OpenSSL 1.1.1 version library for encryption and Setup, respectively. We used Python 3.11.5 version in The Python Community to implement genetic data preprocessing inside SGX, and we evaluated the performance of the algorithms in the SGX hardware debug mode. The experimental environment was deployed on a PC with an Intel ® Core ™ i7-10510U CPU (1.8 GHz∗8), 32G memory, and Ubuntu 20.04.3LTS operating system.

Table 3 shows the size of the genome data blocks of each country under different edge servers and the size of the corresponding ORAM tree built on the cloud server.

Table 3. Sample data size.

Edge Server	Japan	Gambia	Britain	American	China
Genome Blocks	383	414	4486	4715	9680
ORAM Tree Size	3066	3066	49,146	49,146	98,298

6.2. Function Analysis

Table 4 presents the function comparison between SEDASGX and existing research solutions, including security, analysis method, and dynamic update. Here, reference [13] mainly proposes a secure genetic testing framework based on SGX, which can defend against malicious attacks. However, multi-analysis methods, dynamic update, and IBE are not considered. Reference [14] adopts an oblivious RAM mechanism to avoid the access pattern leakage of the interaction between the user and the server with SGX. Nonetheless, it only supports the *Chisquare* analysis method and dynamic update is not considered. Similarly, reference [15] also does not consider the diversity of analysis methods, IBE, and the application of actual scenarios.

Table 4. Contrast of functions.

Scheme	*ChiSquare*	*Fisher*	*LR*	*Update*	*Obliviousness*	*SGX*	*IBE*
[13]	✗	✗	✗	✗	✗	✓	✗
[14]	✗	✗	✗	✗	✓	✓	✗
[15]	✓	✗	✗	✗	✗	✓	✗
SEDASGX	✓	✓	✓	✓	✓	✓	✓

The ✗represents not have this function and The ✓represents having this function in the Table 4.

6.3. Performance Analysis

Figure 3 shows the performance of each algorithm in the SEDASGX. In the preprocessing phase, the overhead of preprocessing increases with the increase in datasets under different edge servers. Among them, the overhead of enclave initialization is roughly the same for the same level of data volume. In the analysis process, the larger the ORAM tree $\mathcal{T}_i$ created, the more genetic data on the read path, and the greater the analysis overhead.

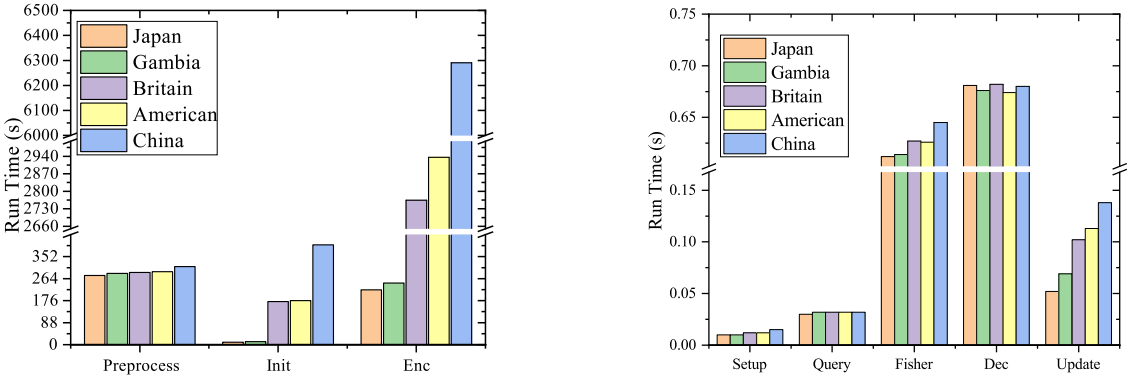

Figure 3. Performance over each algorithm of SEDASGX.

Figure 4 illustrates the performance of data preprocessing by different edge servers. It can be seen intuitively that as the amount of data held by the edge server increases, the overhead of preprocessing operations will also increase, and there is a linear relationship between the two, but the increase is not very drastic.

Figure 5 shows the comparison of the update time of the two edge servers with the largest and smallest data volumes. The update cost of edge servers owned by China is higher than the update cost of edge servers owned by The Gambia. Through comparison, the update efficiency is not only related to the amount of original data but also related to the number of update gene loci. Notice: The content presented in Figure 5 is the update cost for a small number of loci on a chromosome. We only want to reflect the relationship between the update cost and the number of updated gene loci according to Figure 5. Our scheme is built based on real genetic datasets, so the framework of the scheme is easily scalable to handle massive genetic loci.

Figure 6 shows the computational overhead of testing the three analysis algorithms of *Chisquare*, *Fisher*, and *Logistic Regression* under the two cases of hardware SGX-assisted cloud servers and traditional cloud servers. Experimental results show that the three genetic data analysis methods under the SGX-assisted cloud server are significantly faster than the three genetic data analysis methods under traditional cloud servers, and each analysis algorithm is approximately 2.5 times faster. This is because the genetic data analysis on the corresponding plaintext is performed after the ciphertext of a certain path on the ORAM tree is decrypted in the enclave. Meanwhile, SEDASGX not only reduces the communication cost between the client (User/Patient) and the cloud server but also makes the client lightweight, so that the client does not need to preprocess their genomic data. In summary, SEDASGX has high analysis efficiency.

Furthermore, in terms of security, SEDASGX not only inherits the data confidentiality and integrity of the non-SGX traditional scheme, but also has a trusted hardware environment that the non-SGX scheme does not have to ensure the confidentiality and integrity of the code. Therefore, the SEDAGX scheme can provide more secure genetic data analysis. In terms of calculation speed, while ensuring the same security strength, the plaintext calculation rate of the SEDASGX scheme within the SGX hardware is much higher than the ciphertext calculation rate of the non-SGX traditional scheme.

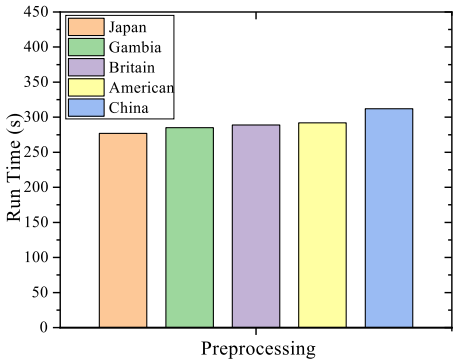

Figure 4. The performance of the preprocessing operation.

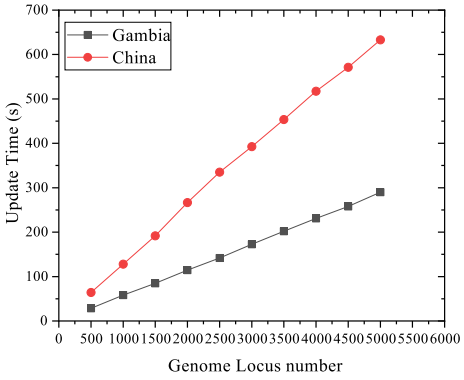

Figure 5. The comparison of update time.

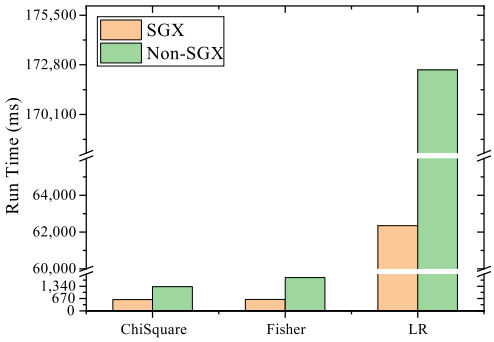

Figure 6. The comparison of analysis time.

7. Conclusions

In this paper, we construct a secure and efficient dynamic analysis scheme for genome data within an SGX-assisted server. First, we design a multi-party genetic data analysis architecture based on Intel SGX and IBE in edge computing scenarios. This framework relies on Intel SGX to ensure the confidentiality and integrity of genetic data while leveraging the IBE to enable the multi-party analysis scenario. To mitigate the threat of access pattern leakage, we employ SGX to construct an oblivious ORAM tree structure for obfuscating memory access patterns. Simultaneously, we not only implement plaintext genomic data analysis within trusted hardware but also provide various analytical methods for genomic data. Finally, the SEDASGX implements dynamic updates of genomic data to ensure more accurate analysis in cases of genetic mutations due to environmental and other factors. Moreover, the experimental results show that SEDASGX is more efficient than non-SGX in genome data analysis.

Future work includes deploying the scheme in a real-world environment (e.g., a large-scale hospital) with the aims of evaluating and refining the scheme (if necessary) to provide additional functionalities without compromising on security and efficiency.

Additionally, we will also consider the situation of a more powerful adversary and pursue higher analysis efficiency.

Author Contributions: Conceptualization, B.L. and Q.W.; Methodology, B.L. and Q.W.; Software, B.L.; Formal analysis, Q.W.; Investigation, D.F.; Resources, F.Z.; Data curation, D.F.; Writing—original draft, B.L.; Writing—review and editing, B.L.; Supervision, F.Z.; Project administration, F.Z. and Q.W.; Funding acquisition, F.Z. All authors have read and agreed to the published version of the manuscript.

Funding: This work was supported in part by the National Natural Science Foundation of China under Grant 62202090, 62173101 and 62072090, by Liaoning Province Natural Science Foundation Medical-Engineering Cross Joint Fund under Grant 2022-YGJC-24, by Doctoral Scientific Research Foundation of Liaoning Province under Grant 2022-BS-077, and by the Fundamental Research Funds for the Central Universities under Grant N2217009.

Data Availability Statement: Data are contained within the article.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Vellela, S.S.; Reddy, B.V.; Chaitanya, K.K.; Rao, M.V. An Integrated Approach to Improve E-Healthcare System using Dynamic Cloud Computing Platform. In Proceedings of the 2023 5th International Conference on Smart Systems and Inventive Technology (ICSSIT), Tirunelveli, India, 23–25 January 2023; IEEE: Pisctaway, NJ, USA, 2023; pp. 776–782.
2. Garrison, E.; Kronenberg, Z.N.; Dawson, E.T.; Pedersen, B.S.; Prins, P. A spectrum of free software tools for processing the VCF variant call format: Vcflib, bio-vcf, cyvcf2, hts-nim and slivar. *PLoS Comput. Biol.* **2022**, *18*, e1009123. [CrossRef]
3. Xu, Y.; Ren, J.; Wang, G.; Zhang, C.; Yang, J.; Zhang, Y. A blockchain-based nonrepudiation network computing service scheme for industrial IoT. *IEEE Trans. Ind. Inform.* **2019**, *15*, 3632–3641. [CrossRef]
4. Gürsoy, G.; Li, T.; Liu, S.; Ni, E.; Brannon, C.M.; Gerstein, M.B. Functional genomics data: Privacy risk assessment and technological mitigation. *Nat. Rev. Genet.* **2022**, *23*, 245–258. [CrossRef] [PubMed]
5. Sero, D.; Zaidi, A.; Li, J.; White, J.D.; Zarzar, T.B.G.; Marazita, M.L.; Weinberg, S.M.; Suetens, P.; Vandermeulen, D.; Wagner, J.K.; et al. Facial recognition from DNA using face-to-DNA classifiers. *Nat. Commun.* **2019**, *10*, 2557. [CrossRef]
6. Kim, M.; Lauter, K. Private genome analysis through homomorphic encryption. BMC medical informatics and decision making. *BioMed Cent.* **2015**, *15*, 1–12.
7. Sarkar, E.; Chielle, E.; Gürsoy, G.; Mazonka, O.; Gerstein, M.; Maniatakos, M. Fast and scalable private genotype imputation using machine learning and partially homomorphic encryption. *IEEE Access* **2021**, *9*, 93097–93110. [CrossRef]
8. Wang, S.; Zhang, Y.; Dai, W.; Lauter, K.; Kim, M.; Tang, Y.; Xiong, H.; Jiang, X. HEALER: Homomorphic computation of ExAct Logistic rEgRession for secure rare disease variants analysis in GWAS. *Bioinformatics* **2016**, *32*, 211–218. [CrossRef] [PubMed]
9. Blatt, M.; Gusev, A.; Polyakov, Y.; Goldwasser, S. Secure large-scale genome-wide association studies using homomorphic encryption. *Proc. Natl. Acad. Sci. USA* **2020**, *117*, 11608–11613. [CrossRef] [PubMed]
10. Kamm, L.; Bogdanov, D.; Laur, S.; Vilo, J. A new way to protect privacy in large-scale genome-wide association studies. *Bioinformatics* **2013**, *29*, 886–893. [CrossRef]
11. Dong, C.; Weng, J.; Liu, J.N.; Yang, A.; Liu, Z.; Yang, Y.; Ma, J. Maliciously secure and efficient large-scale genome-wide association study with multi-party computation. *IEEE Trans. Dependable Secur. Comput.* **2022**, *20*, 1243–1257. [CrossRef]
12. Zhu, X.; Ayday, E.; Vitenberg, R. A privacy-preserving framework for conducting genome-wide association studies over outsourced patient data. *IEEE Trans. Dependable Secur. Comput.* **2022**, *20*, 2390–2405. [CrossRef]
13. Chen, F.; Wang, C.; Dai, W.; Jiang, X.; Mohammed, N.; Al Aziz, M.M.; Sadat, M.N.; Sahinalp, C.; Lauter, K.; Wang, S. PRESAGE: PRivacy-preserving gEnetic testing via SoftwAre guard extension. *BMC Med. Genom.* **2017**, *10*, 77–85. [CrossRef]
14. Mandal, A.; Mitchell, J.C.; Montgomery, H.; Roy, A. Data oblivious genome variants search on Intel SGX. In Proceedings of the International Workshop on Data Privacy Management, Barcelona, Spain, 6–7 September 2018; Springer International Publishing: Cham, Swizterland, 2018; pp. 296–310.
15. Kockan, C.; Zhu, K.; Dokmai, N.; Karpov, N.; Kulekci, M.O.; Woodruff, D.P.; Sahinalp, S.C. Sketching algorithms for genomic data analysis and querying in a secure enclave. *Nat. Methods* **2020**, *17*, 295–301. [CrossRef]
16. Costan, V.; Devadas, S. Intel SGX explained. Cryptology ePrint Archive. Available online: https://eprint.iacr.org/2016/086 (accessed on 7 August 2022).
17. Zheng, W.; Wu, Y.; Wu, X.; Feng, C.; Sui, Y.; Luo, X.; Zhou, Y. A survey of Intel SGX and its applications. *Front. Comput. Sci.* **2021**, *15*, 1–15. [CrossRef]
18. Amjad, G.; Kamara, S.; Moataz, T. Forward and backward private searchable encryption with SGX. In Proceedings of the 12th European Workshop on Systems Security, Dresden, Germany, 25–28 March 2019; pp. 1–6.
19. Jiang, Q.; Qi, Y.; Qi, S.; Zhao, W.; Lu, Y. Pbsx: A practical private boolean search using Intel SGX. *Inf. Sci.* **2020**, *521*, 174–194. [CrossRef]
20. Will, N.C.; Maziero, C.A. Intel Software Guard Extensions Applications: A Survey. *ACM Comput. Surv.* **2023**, *55*, 322. [CrossRef]

21. Djoko, J.B.; Lange, J.; Lee, A.J. Nexus: Practical and secure access control on untrusted storage platforms using client-side sgx. In Proceedings of the 2019 49th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN), Portland, OR, USA, 24–27 June 2019; IEEE: Piscataway, NJ, USA, 2019; pp. 401–413.
22. Johnson, D.; Menezes, A.; Vanstone, S. The elliptic curve digital signature algorithm (ECDSA). *Int. J. Inf. Secur.* **2001**, *1*, 36–63. [CrossRef]
23. Stefanov, E.; Dijk, M.V.; Shi, E.; Chan, T.H.H.; Fletcher, C.; Ren, L.; Yu, X.; Devadas, S. Path ORAM: An extremely simple oblivious RAM protocol. *J. ACM (JACM)* **2018**, *65*, 1–26. [CrossRef]
24. LeMay, C. Privacy-Preserving Chi-Squared Tests Using Homomorphic Encryption. Available online: https://www.cs.utexas.edu/~dwu4/courses/sp22/static/projects/LeMay.pdf (accessed on 15 August 2023).
25. Zhao, G.; Yang, H.; Yang, J.; Zhang, L.; Yang, X. A data-based adjustment for fisher exact test. *Eur. J. Stat.* **2021**, *1*, 74–107. [CrossRef]
26. Ayers, K.L.; Cordell, H.J. SNP selection in genome-wide and candidate gene studies via penalized logistic regression. *Genet. Epidemiol.* **2010**, *34*, 879–891. [CrossRef]
27. Naccache, D. Secure and practical identity-based encryption. *IET Inf. Secur.* **2007**, *1*, 59–64. [CrossRef]
28. McGrew, D.A.; Viega, J. The security and performance of the Galois/Counter Mode (GCM) of operation. In Proceedings of the International Conference on Cryptology in India, Chennai, India, 20–22 December 2004; Springer: Berlin/Heidelberg, Germany, 2004; pp. 343–355.
29. Bard, G.V. Modes of Encryption Secure against Blockwise-Adaptive Chosen-Plaintext Attack. Cryptology ePrint Archive. Available online: https://eprint.iacr.org/2006/271, (accessed on 15 August 2023).
30. GeneData Set. Available online: http://hgdownload-euro.soe.ucsc.edu/gbdb/hg19/1000Genomes/phase3/ (accessed on 7 August 2023).