Privacy Computing with Right to Be Forgotten in Trusted Execution Environment

Hui Liu*, Hongzhi Luo*, Shaofeng Li[†], Tian Dong*, Guoxing Chen*, Yan Meng*, and Haojin Zhu*[‡]
*Department of Computer Science and Engineering, Shanghai Jiao Tong University, Shanghai, China
[†]Department of Mathematics and Theories, Peng Cheng Laboratory, Shenzhen, China
Email: {sjtuliuhui, luohz1, tian.dong, guoxingchen, yan_meng, zhu-hj}@sjtu.edu.cn, lishf@pcl.ac.cn

Abstract—**Sharing private data is at risk of potential data breaches, including the violation of the "right to be forgotten" principle, undermining people's willingness to share their data. A common solution is to involve the Trusted Execution Environment (TEE), which allows the data provider to verify the computation process without trusting others. However, previous works have either encountered incomplete computations or lacked scalability. In this paper, we propose TEErase, a secure data-sharing framework that addresses these issues. TEErase protects every phase of the data lifecycle and enables individuals to share personal data with a predefined privacy budget. In particular, TEErase applies comprehensive *privacy budgeting mechanisms* to efficiently manage privacy budgets and employs an *asynchronized execution* approach that decouples budget consumption from data computation. TEErase records the predefined privacy budgets, verifies privacy consumption requests, updates the remaining budgets, and deletes data that have exhausted their budgets by preventing any attempts to access them. We implement a prototype of TEErase and evaluate its effectiveness with a realistic case study on Genome-Wide Association Study.**

Index Terms—**Data Sharing, Differential Privacy, Trusted Execution Environment, Data Access and Usage Control**

I. Introduction

Data privacy has been gaining more attention as various data protection laws have been enacted, such as GDPR, CCPA, PIPL, *etc*. When collecting massive sensitive personal data for research and development purposes (*e.g.*, genomic information for clinical care), the privacy information of personal data must be carefully protected and secured. One primary concern for data providers to share their data is the lack of control over the sensitive data after the data release. After releasing personal data, in most cases, data providers have no control over data usage and are not aware of potential privacy violations. For example, sensitive data can be shared with unauthorized parties or violate the "right to be forgotten" principle regulated by GDPR. There is another concern that privacy breaches may occur due to various data inference attacks that have been disclosed [14]. The more this same data is shared and used, the higher the risk of potential privacy violations. Thus, to reassure data providers, it is desired that the usage and destruction of their data are under their predefined constraints. Particularly, data should be used only when privacy risks are under control and purged before the leakage is beyond acceptable bounds.

Mechanisms for self-expiring/self-destructing have been studied previously [4], [12], [15]. Specifically, they encrypt private data and transmit encryption keys to data seekers who want to access the data. Some prior work [4], [12] rely on the distributed key managers to revoke encryption keys, while others [15] require data users to puncture the encryption key in order to revoke their capacity of decrypting ciphertext. However, they assume that the removal of encryption keys is executed correctly and do not regulate the usage of private data to prevent potential data breaches.

Further works [3], [9] introduce TEE, allowing key expiration without establishing a trust chain between data providers and data seekers in advance. In particular, the predefinable nature of programs running on TEE lends to their inviolability by third parties. This characteristic can be utilized to manage encryption keys and regulate private data computation based on seeker-provider agreement. To defend *rollback* or *forking* attacks that cause the TEE's state to be rolled back to a previous version or forked into multiple versions, they implement additional integrity verification procedures. However, this safeguard may lead to incomplete computation if the enclave fails and cannot deliver the expected results.

Besides, most policy-based expiration schemes [3], [12] aim to facilitate fine-grained access control, making it challenging to effectively verify whether a large batch of data requests adheres to the expiration conditions. To accommodate custom expiration conditions, these policies are typically stored and updated independently and require the participation of data providers. When making data requests, the communication time increases linearly with the number of key requests and key managers, which ultimately leads to scalability issues.

In this paper, we propose TEErase, a secure and controllable data-sharing framework that addresses limitations of scalability and incomplete computation issues that existing work possessed. TEErase restricts the utilization of private data by associating data entries with *privacy budgets* which quantify the acceptable bounds of privacy leakage. Hence, A *privacy budgeting mechanism* is proposed to quantify the levels of privacy leakage and aggregate the budgets to determine when to destruct data. TEErase relies on confidentiality and integrity guarantees of TEE to manage privacy budgets and destruct budget-exhausted data, thereby ensuring privacy leakage within a limited scope. On the other hand, TEErase adopts asynchronized execution, decoupling the management

[‡]Haojin Zhu is corresponding author.

979-8-3503-1090-0/23/$31.00 © 2023 IEEE

of budgeting requests and data computation to ensure both privacy budgeting compliance and result delivery. This allows TEERASE to support recalculation after failure. To avoid unauthorized budget consumption resulting from processing data in varying conditions, TEERASE tracks the hash of the provided data from data users and binds a fixed random seed to guarantee deterministic output. Through centralized and unified budget management, data seekers can effectively collect massive data from different individuals.

In summary, our work makes the following contributions:

- We propose TEERASE, a centralized framework that enhances data providers' control over their released data.
- We design comprehensive privacy budgeting mechanisms that help to regulate privacy breaches.
- We employ an aysnchronized execution approach that separates budget requests from data access, addressing incomplete computation issues encountered by other TEE-based schemes. This enables more flexible usage of data.
- We implement a prototype of TEERASE with Intel SGX, conduct a realistic case study and empirically analyze the efficiency of TEERASE.

The remainder of this paper is organized as follows. We present the background in Sec. II, and Sec. III introduces the working scenario and design goals. In Sec. IV, we describe the details of TEERASE. Finally, we show experimental evaluation in Sec. V and conclude the paper in Sec. VI.

II. BACKGROUND

A. Trusted Execution Environment

A Trusted Execution Environment (TEE) provides a shielded execution environment that ensures the integrity and confidentiality of data and computation processes. It prevents unauthorized access from external sources, such as malicious operating systems, to tamper with codes or obtain private data.

Intel SGX. Intel Software Guard eXtensions (SGX)[2] is Intel's implementation of TEE, which extends the x86 instruction set architecture to provides an isolated execution space, called *enclave*. SGX offers *remote attestation* to verify the authenticity of an enclave and its initial state in a remote host by examining the signed hash of the data and code inside enclave. With remote attestation, the client can establish a secure communication channel with remote enclaves.

Monotonic Counters. Monotonic counters are used to provide state continuity for TEEs, there are several ways to implement it [7], [8]. Hardware-based solutions, such as SGX monotonic counters, use non-volatile storage that restricts the speed of write access. Software-based solutions [7], [8] introduce distributed enclaves to break the limits of writing cycles. What's more, Niu *et al.* introduces blockchain to initialize the system, without relying on a central authority for trust [8].

B. Differential Privacy

Differential Privacy (DP) is a rigorous model that ensures that an attacker cannot infer the existence of a specific data point in the sensitive dataset through any query computed from it. Formally, we have

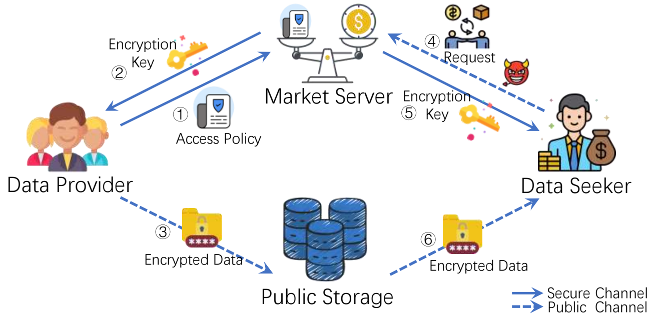


Fig. 1: Working Scenario of TEERASE.

Definition 1. *For any $\epsilon > 0$ and $\delta \in [0, 1]$, a randomized mechanism $\mathcal{M}$ is (ϵ, δ)-DP if for two adjacent dataset $\mathcal{D}$ and $\mathcal{D}'$, which differs only one record, and any output $\mathcal{O} \subseteq Range(\mathcal{M})$, it satisfies:*

$$\Pr[\mathcal{M}(\mathcal{D}) \in \mathcal{O}] \leq \exp(\epsilon) \times \Pr[\mathcal{M}(\mathcal{D}') \in \mathcal{O}] + \delta. \quad (1)$$

The parameter ϵ quantifies the probability of privacy leakage with probability $1-\delta$. If we set $\delta = 0$, we have ϵ-DP satisfied. A key strength of DP is its composition property:

Theorem 1. *For any $i \in \{1, \ldots, n\}$, if mechanism $\mathcal{M}_i$ satisfies (ϵ_i, δ_i)-DP, then the composition of all these mechanisms satisfies $(\sum_{i=1}^{n} \epsilon_i, \sum_{i=1}^{n} \delta_i)$-DP.*

C. Self-Expiring/Self-Destructing

Geambasu *et al.* first proposed Vanish, a self-expiring system that utilizes Distributed Hash Tables (DHTs) [4]. They encrypt the private data, split the encryption key into pieces through Shamir secret shares, and store them across periodically updating DHTs. Tang *et al.* improved Vanish by facilitating key managers with attribute-based encryption, which enables fine-grained access control [12]. Wei *et al.* further developed the concept of forward-secure attribute-based puncturable encryption that allows for data erasure without interactions [15]. However, these works only control the expiration process of the encryption key but cannot prevent attackers from storing a plain-text copy of the data.

Gao *et al.* leveraged TEE with attested execution to regulate the functional access [3]. They establish a distributed access committee on independent TEEs and run the Raft consensus protocol to regulate data access. Ren *et al.* proposed a counter-based data assured deletion scheme using TrustZone, which can save traffic cost with one-time key retrieval [9]. However, compared to TEERASE, these works cannot guarantee result delivery upon computation failure.

III. OVERVIEW

A. Working Scenario

As illustrated in Fig. 1 , TEERASE involves three parties: the data provider, the data seeker, and a market server.

- The **Data Provider** owns private data and wants to share them within a certain threshold of leakage probability. They may specify a *privacy budget* (*e.g.*, the maximum number of times the data can be accessed) to present and the risk of privacy leakage it can suffer with certain usage.
- The **Data Seeker** looks through the market server's website to find its desirable data and calculates this data within TEE to gain the statistic result.
- The **Market Server** is a mediator between the data provider and data seeker. It collects data and its privacy budget from its owner and advertises them on its website. It enables the privacy budgeting mechanism with the assistance of TEE.

B. Threat Model

We assume that the data provider is trustworthy and all components on her host are reliable. However, she does not trust either the data seeker or the market server, as they may attempt to disclose her private information. Besides, the data seeker can be an attacker who tries to recalculate private data inside enclaves that have been rolled back and the service provider may tamper with privacy budgets for profiting.

We assume the TEERASE enclaves running on the data seeker side and market server are trusted. We also assume the attacker is powerful to manipulate the system software stack (*e.g.*, OS or hypervisor) over the host where TEE is provisioned. The attacker cannot extract confidential information protected by TEEs nor corrupt states inside them, but has access to read and modify memory for all non-enclave processes and has permission to read/write persistent storage. There is no guarantee of message reliability when sent from or received by enclaves. Additionally, Denial of Service attacks and side channel attacks are not taken into account.

C. Problem Formulation

Consider N data providers $\mathrm{PRD}_1, \ldots, \mathrm{PRD}_N$, each of which has a data entry d_i, $i = \{1, \ldots, N\}$, to be shared. Let $\mathcal{D} = \{d_1, \ldots, d_N\}$ denote the set consisting of all data entries from these data providers. When uploading d_i to the market server, the data provider PRD_i specifies a privacy budget $\mathcal{B}(d_i)$ to represent the acceptable privacy leakage risk.

Consider M data seekers SKR_1, ..., SKR_M. Each data seeker $\mathrm{SKR}_j, j = \{1, \ldots, M\}$, would like to collect a subset of data entries to run her algorithm $\mathcal{A}_j$. We assume that all data entries involved in one algorithm $\mathcal{A}_j$ would suffer from a same level privacy leakage risk, denoted by $\mathcal{L}(\mathcal{A}_j)$.

The market server collects data entries along with their privacy budgets from data providers and serves requests from data seekers. Considering a list of L data requests, $r_l = (s_l, D_l, \mathcal{L}(\mathcal{A}_{s_l})), l = \{1, \ldots, L\}$, where $s_l \in \{1, \ldots, M\}$ presents the l-th request comes from the data seeker SKR_{s_l}. $D_l \subset \mathcal{D}$ indicates that the SKR_{s_l} is requesting the subset data entries D_l in the l-th request. For each data entry d_i, the current accumulated privacy leakage risk for should be less than its privacy budget $\mathcal{B}(d_i)$:

$$\sum_{l\in\{l|d_i\in D_l, l=1,\ldots,L\}} p_i \mathcal{L}(\mathcal{A}_{s_l}) \le \mathcal{B}(d_i), \qquad (2)$$

where $p_l = 1$ represents r_l is approved and vice versa.

The market server tracks the unused privacy budget $\mathcal{U}^L(d_i)$ for data d_i, where $\mathcal{U}^L(d_i)$ is calculated by

$$\mathcal{U}^L(d_i) = \mathcal{B}(d_i) - \sum_{l\in\{l|d_i\in D_l, l=1,\ldots,L\}} p_l \mathcal{L}(\mathcal{A}_{s_l}). \qquad (3)$$

D. Running Example

We use a running example to illustrate our budgeting mechanism and its execution process. Let us consider N data providers PRD_i, $i = \{1, \ldots, N\}$, who want to share their genes at the form of Single Nucleotide Polymorphism (SNP) records, denotes as d_i. We assume PRD_i sets privacy budget $\mathcal{B}(d_i) = (5, 10^{-3})$ in DP mechanism (*i.e.*, $\epsilon = 5, \delta = 10^{-3}$) due to privacy concerns, and allows d_i being used on algorithms (*e.g.*, machine learning, genome wide association).

Let SKR_1 be a data seeker who sends request $r_1 = (1, D_1, \mathcal{A}_1)$, where $\mathcal{A}_1$ trains a model with DP-SGD [1] and it requires privacy cost $\mathcal{L}(\mathcal{A}_1) = (1.26, 10^{-5})$, *i.e.*, $\mathcal{A}_1$ adds noise when updating gradient with distribution $\mathcal{N}(0, \delta^2 C^2 \boldsymbol{I})$ where $\delta = 4$, C is the clipping threshold of gradient, and trains 10000 batches for batch size $0.01|D_1|$ according to [1]. After granting r_1, market server updates the unused privacy budget $\mathcal{U}^1(d_i) = \mathcal{B}(d_i) - \mathcal{L}(\mathcal{A}_1) = (3.74, 9.9 \times 10^{-4})$ for $i \in D_1$. SKR_2 aims to determine if the Hardy-Weinberg Equilibrium (HWE) applies to a specific allele. She sends request $r_2 = (2, D_2, \mathcal{A}_2)$, where $\mathcal{A}_2$ calculate HWE with privacy costs $\mathcal{L}(\mathcal{A}_2) = (1, 10^{-5})$ (representing adding Laplace noise with variance $\delta^2 = \frac{375.6N^2}{(N+2)^2}$) [13]. The market server then evaluate r_2 and update $\mathcal{U}^2(d_i) = \mathcal{U}^1(d_i) - \mathcal{L}(\mathcal{A}_2)$ for $i \in D_2$.

E. Design Goals and Challenges

We claim the design goals that TEERASE must offer:

***G*1: Full Life-cycle Confidentiality Guarantee.** The main goal of TEERASE is to prevent the data providers' sensitive data from being leaked throughout its entire life-cycle. It means that any party except the data provider should never be able to access the plain-text data besides obtaining the results computed from it.

***G*2: Efficiently Access Control and Self-Destructing.** In addition, TEERASE wants to reassure data providers that their data usage and destruction are in accordance with their predefined privacy budgeting mechanism autonomously and efficiently. Once a data provider selects her privacy budgeting mechanism and uploads her data, any actions that violate it will be limited by TEERASE.

***G*3: Guaranteed Result Delivery.** Previous works either shared plain-text data [4] or cause incomplete computation issues [3]. To prevent rollback attacks, [3] validates that a Token remains exclusively within the private memory along

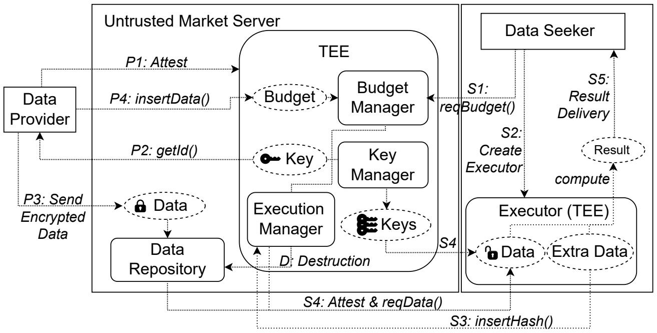


Fig. 2: The architecture of TEERASE. The Market Server consists of three trusted modules (Budget Manager, Key Manager, and Execution Manager) and a public Data Repository.

with the decryption key, ensuring that it is executed only once even if it encounters a failure. TEERASE must ensure that it complies with privacy regulations while also delivering computation results.

To achieve $\boldsymbol{G}1$, TEERASE stores encrypted data in persistent storage and allows decryption only within the enclave. Computation takes place within TEEs, with the query result being the only information returned to the data seeker.

To achieve $\boldsymbol{G}2$, when TEERASE receives $(L+1)^{th}$ data request $r_{L+1} = (s_{L+1}, D_{L+1}, \mathcal{A}_{L+1})$, it only approves r_{L+1} if there is sufficient remaining privacy budget to cover the cost of the request, *i.e.*,

$$\forall d_i \in D_k, \mathcal{U}^L(d_i) \geq \mathcal{L}(\mathcal{A}_{s_{L+1}}), \tag{4}$$

and then the corresponding amount will be deducted from the unused budget, *i.e.*,

$$\mathcal{U}^{(L+1)}(d_i) = \mathcal{U}^L(d_i) - \mathcal{L}(\mathcal{A}_{s_{L+1}}), d_i \in D_{L+1}. \tag{5}$$

Once the expiry condition

$$\mathcal{U}^{(L+1)}(d_i) = 0, d_i \in \mathcal{D} \tag{6}$$

is met (*e.g.*, after a certain period of time or when the privacy budget is used up), the destruction process commences immediately to prevent any unauthorized access to d_i.

To achieve $\boldsymbol{G}3$, TEERASE adopts asynchronized execution that separates the budget requests and data access, allowing multiple computation requests inside TEEs.

IV. SYSTEM DESIGN

A. System Workflow

Fig. 2 shows the workflow of TEERASE and the execution procedures inside TEE are explained in Algo. 1. In the beginning, PRD$_i$ sets her global privacy budget $\mathcal{B}(d_i)$ according to her risk tolerance preference. She then attests to the functionality of the market server enclave for safeguarding d_i ($P1$). Next, she asks the market server to execute $getId$ for generating a unique Id for d_i and the encryption key (the key management will be detailed later in Sec. IV-E). These are sent back through the secure channel established earlier ($P2$). With the received encryption key, PRD$_i$ encrypts d_i and uploads the encrypted data to the Data Repository ($P3$). To finish budget upload process, PRD$_i$ calls $insertData$ to update $\mathcal{B}(d_i)$ ($P4$).

Upon receiving $r_l = (s_l, D_l, \mathcal{L}(\mathcal{A}_{s_l}))$, the market server invokes $reqBudget$ to request the Budget Manager to verify if $\mathcal{U}^{(l-1)}(d_i), d_i \in D_l$ are sufficient to use ($S1$). Once the Budget Manager receives r_l where $d_i \in D_l$ retains adequate budgets, it sends a signal to the Execution Manager. The signal will also trigger automated data destruction if the data runs out of the budget (D). Upon approval, SKR$_{s_l}$ creates a Trusted Executor $\boldsymbol{E}$ to load $\mathcal{A}_{s_l}$ whenever she is ready to compute with D_l ($S2$). If SKR$_{s_l}$ provides auxiliary data for computation, it's necessary to call $insertHash$ to upload its hash value to the Execution Manager ($S3$). Before calling $reqKey$ to release the encryption key of D_l to $\boldsymbol{E}$, the Execution Manager within the market server enclave attests $\boldsymbol{E}$ to validate the correctness of $\mathcal{A}_{s_l}$ and the consistency of additional data ($S4$). After fetching encrypted data and receiving keys, $\boldsymbol{E}$ computes with D_k and potentially extra data, releasing the result ($S5$).

B. Budgeting Mechanism

Given d_i, we aim to propose a privacy budgeting mechanism that bridges the gap between $\mathcal{L}(\mathcal{A}_j)$, $\mathcal{B}(d_i)$, and $\mathcal{U}(d_i)$. We categorize $\mathcal{B}(d_i)$ and $\mathcal{U}(d_i)$ into *general budgets* and *algorithm dependent budgets*. *General budgets* are those can be adapted to different algorithms, *e.g.*, the number of accesses and (ϵ, δ) in DP. There are various methods for calculating $\mathcal{L}(\mathcal{A}_j)$ for different $\mathcal{A}_j$(*e.g.*, the example described in Sec. III-D). The level of privacy breach $\mathcal{L}(\mathcal{A}_j)$ is related to each specific $\mathcal{A}_j$. Hence, when updating $\mathcal{U}^j(d_i)$ with Eq. 5, the market server must pre-quantify $\mathcal{L}(\mathcal{A}_j)$ based on the type of $\mathcal{A}_j$.

When dealing with *algorithm dependent budgets*, the deduction of $\mathcal{B}(d_i)$ is associated with the usage of data ($\mathcal{A}_j$), *e.g.*, if PRD$_i$ believes that the leakage level of a machine learning task is related to training hyper-parameters, she may set the number of maximum training epochs as $\mathcal{B}(d_i)$. Therefore, a straightforward budget subtracting can be implemented.

As for different budget types of $\mathcal{B}(d_i)$, such as the number of access and training epochs, TEERASE calculates them separately, and the self-destructing process is triggered when any budget meets Eq. 6.

C. Budgeting Integrity Guarantee

Although the unused budgets $\mathcal{U}$ are protected by TEEs, it's still vulnerable to rollback attacks wherein an attacker reverts the states of required budgets, resulting in a violation of privacy constraints. To prevent this, TEERASE introduces monotonic counters inspired by [7]. Specifically, TEERASE maintains three types of monotonic counters: ID_CTR, BGT_CTR, and KEY_CTR. ID_CTR tracks the number of assigned data id, which uniquely determines the encryption key. BGT_CTR tracks all budget updates, including initialization of $\mathcal{B}(d_i)$ and updates to $\mathcal{U}(d_i)$. KEY_CTR tracks the number of data entries with immutable additional data, to ensure consistency and indicate the completion of budget consumption. The increments of counters are displayed in Algo. 1.

Algorithm 1: Budget Secure Updating Algorithm

```
Initialize $\text{ID\_CTR}, \text{BGT\_CTR}, \text{KEY\_CTR}$;
/* Initialize storage, master key */
$\mathcal{MK} \leftarrow \text{KEY\_GEN}(\text{GET\_RANDOM}(), pwd)$;
$budgetDict \leftarrow \emptyset, hashDict \leftarrow \emptyset$ ;

/* Prepare Insert Data */
Procedure getId($\text{PRD}_i$):
    $Id \leftarrow \text{INC\_CTR}(\text{ID\_CTR})$ ;
    $encKey \leftarrow \text{KEY\_GEN}(Id, \mathcal{MK})$ ;
    $Send(\text{PRD}_i, encKey, Id)$;

/* Request Insert Data */
Procedure insertData($Id, Enc(d_{Id}), \mathcal{B}(d_{Id})$):
    assert $(Id \notin budgetDict)$ ;
    store $Enc(d_{Id})$;
    $\text{INC\_CTR}(\text{BGT\_CTR})$ ;
    $budgetDict[Id] \leftarrow \mathcal{B}(d_{Id})$;

/* Request Budget Cost */
Procedure reqBudget($\text{SKR}_j, Id, \mathcal{L}(\mathcal{A}_j)$):
    assert $(Id \in budgetDict)$ ;
    $\mathcal{U}(d_{Id}) \leftarrow budgetDict[Id]$;
    if $\mathcal{U}(d_{Id}) > \mathcal{L}(\mathcal{A})$ then
        $\text{INC\_CTR}(\text{BGT\_CTR})$;
        $budgetDict[Id] \leftarrow \mathcal{U}(d_{Id}) - \mathcal{L}(\mathcal{A}_j)$;
        $hashDict[Id] \leftarrow (0, Null)$
    else                                  /* Budget Not Enough */
        $Send(\text{SKR}_j, Rej)$;
    end

/* Asynchronization Hash Insertion */
Procedure insertHash($\text{SKR}_j, Id, \mathcal{H}$):
    assert $(Id \in hashDict)$;
    $(fixFlag, prevHash) \leftarrow hashDict[Id]$ ;
    if $fixFlag == 0$ then
        $hashDict[Id][1] \leftarrow hash(prevHash, \mathcal{H})$;
    else
        $Send(\text{SKR}_j, Rej)$;
    end

/* Asynchronization Key Request */
Procedure reqKey($Id, \text{SKR}_j, \mathcal{H}$):
    assert $Id \in hashDict$ ;
    $fixFlag, storedHash \leftarrow hashDict[Id]$ ;
    if $storedHash == \mathcal{H}$ then
        if $fixFlag == 0$ then
            $\text{INC\_CTR}$ $(\text{KEY\_CTR})$ ;
            $r \leftarrow \text{GET\_RANDOM}()$;
            $hashDict[Id][0] \leftarrow r$ ;
        else
            $r \leftarrow fixFlag$;
        end
        $encKey \leftarrow \text{KEY\_GEN}(Id, \mathcal{MK})$ ;
        $Send(\text{SKR}_j, r, encKey)$ ;
    else
        $Send(\text{SKR}_j, Rej)$;
    end
```

D. Asynchronized Execution

Previous work [3], [9] cannot guarantee the delivery of results due to incomplete computations from trusted process failures. Furthermore, to avoid depleting the privacy budget, the data seeker may request it in advance but not use it. Simply separating the budget management and data computation creates a vulnerability that can be exploited to perform multiple computations under different conditions.

TEErase addresses this issue by maintaining a table to record data ids and extra hashes of outsourced data. The Market server marks the extra hash immutable and generates a random number before releasing the key. The random number as the random seed makes the stochastic computation process becomes a deterministic process within the executor enclave. If the market server receives a second data request, *e.g.*, the computation process fails or the result is lost, it checks the consistency of outsourced data hash, and then releases the key and the stored random number ($reqKey$ in Algo. 1). Note that with the same random seed and algorithm, the computation over the fixed outsourced data and the provided private data is deterministic, which will not violate the budget restriction.

E. Data Storage and Key Management

TEEs with limited hardware-protected memory regions introduce expensive page swaps. It's common to encrypt private data, store the encrypted data outside the enclave, and keep only the encryption key within the enclave. However, using a single key may increase the risk of key compromise, while maintaining different keys for each data could also drain the limited memory space. Thus, we adopt a deterministic key derivation function (KDF) that uses a master key and unique data IDs to generate the corresponding encryption keys.

Except for encryption keys, there are also unused budgets $\mathcal{U}$ and outsourced data hash $\mathcal{H}$ need to be stored inside protected memory. We apply L2-Cache to lower the memory footprint and implement keyed hash values (*i.e.*, Message Authentication Codes, MACs) for protecting the integrity of data entries upon eviction from the protected memory region. Note that it's optional to encrypt the entries to protect the confidentiality of $\mathcal{U}$ and $\mathcal{H}$, depending on the security level PRD_i required. We organize $(Id, \mathcal{U}(d_i), \mathcal{H})$ into pages as data items to calculate the hash. The MAC is calculated from the page item, page size, and an IV/counter, protecting data stored in unprotected memory from unauthorized modification. Inspired by [5], TEErase maintains in-enclave MAC hashes instead of maintaining a Merkle Tree.

In summary, in our design, only the master key, MAC of $\mathcal{U}$ and $\mathcal{H}$, and their cache stay within the enclave memory to keep a low memory footprint.

V. Evaluation

Implementation. We implemented a prototype of TEErase considering a realistic scenario where data providers share their genes for Genome-Wide Association Study (GWAS). It offers four types of GWAS: LD, HWE, CATT, and FET [10]. Each type supports two budgeting mechanisms: the number of access times and DP [13]. To port GWAS inside enclaves, we preprocess SNPs in the market server with [6].

Our prototype employs multi-threading to optimize performance, assigning worker threads to process encryption/ decryption data files and privacy budget updates based on their ID hash to reduce the need for thread synchronization. We generate the master key using the $BPKDF2SHA512$ algorithm and derive child keys using the $BIP32$ protocol. The implementation is based on SGX SDK 2.17, with IPP Cryptography Library and OpenSSL Library for encryption and hashing.

TABLE I: Response time comparison for different number of worker threads.

Threads	reqBudget (ns)	reqKey (ns)	insertHash (ns)
2	1030.7	1168.9	606.23
4	454.55	600.83	230.37
6	187.55	334.77	62.71
8	108.80	265.78	28.83
16	72.37	296.16	17.37

Experimental settings. We evaluate TEERASE on the Intel Xeon Gold 5318Y processor. The machine runs Ubuntu $20.04.5$ LTS 64 bit with Linux kernel $5.15.0$. We use the dataset from $1,000$ Genomes Phase 3 [11] for GWAS. This dataset comprises $493,463$ SNPs that were gathered from 273 participants. The seeker machine simulates 32 concurrent users that generate requests for private data.

Evaluation Results. To quantify the effectiveness of multi-threading optimization, we measure TEERASE's performance with varying numbers of worker threads. Table I shows the response time of the server when receiving requests outlined in Algo. 1: requesting privacy budget, inserting outsourced hash, and requesting encryption keys for private data. The comparison shows that our paralleling design significantly reduces the response time for all operations.

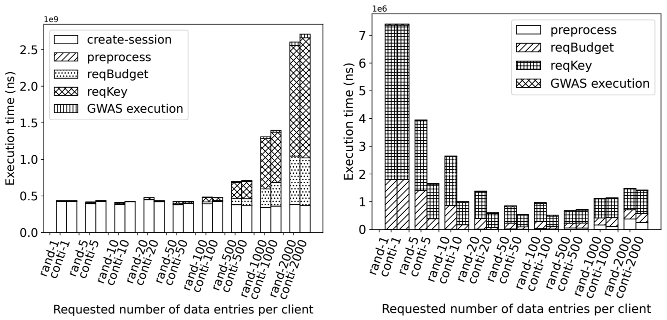
Fig. 3a reports the time taken by a data seeker for various stages, including creating a secure communication channel, pre-processing, budget requesting, private data requesting, and the execution of GWAS analysis. The breakdown is shown in relation to the number of data entries requested. In particular, establishing a secure session (attestation) takes an average of 403.6 ms, which becomes the heaviest workload when requesting a single data entry. When a data seeker requests an increasing number of data entries, the latency remains relatively constant if the request size is less than 100 entries. Even when requesting $2,000$ data entries, it only takes five times longer than requesting one entry.

In our implementation, data entries shared by one provider are assigned with adjacent data ids, causing space continuity of privacy budgets. We called data entries whose budgets are stored next to each other "adjacent data". Fig. 3a demonstrates that the latency remains consistent regardless of whether data seekers make random or adjacent data requests.

Fig. 3b demonstrates the average response time for each data entry when data seekers request different amounts of data entries. As the number of requested data entries increases, both the average key request time and budget request time decrease. However, the pre-processing time required to generate request packages gradually increases.

VI. Conclusion

In this paper, we propose TEERASE to address the issue of massive private entries sharing. TEERASE empowers data providers to regain control over their released data. The supervised usage and destruction of private data prevent the data seeker from violating the budgeting mechanism thus limiting the risk of exposing a particular entry.



(a) Total response time.

(b) Average response time.

Fig. 3: Data seeker's requests with varying data entries.

Acknowledgment

The work was supported by National Key R&D Program of China under Grant No. 2022YFB3103500 and National Natural Science Foundation of China under Grants No. 62132013.

References

[1] M. Abadi, A. Chu, I. Goodfellow, H. B. McMahan, I. Mironov, K. Talwar, and L. Zhang, "Deep learning with differential privacy," in *23rd ACM CCS*, 2016, pp. 308–318.

[2] V. Costan and S. Devadas, "Intel sgx explained," *Cryptology ePrint Archive*, 2016.

[3] M. Gao, H. Dang, and E.-C. Chang, "TEEKAP: Self-expiring data capsule using trusted execution environment," in *38th ACSAC*, 2021, pp. 235—247.

[4] R. Geambasu, T. Kohno, A. A. Levy, and H. M. Levy, "Vanish: Increasing data privacy with self-destructing data." in *18th USENIX security*, 2009, pp. 10–5555.

[5] T. Kim, J. Park, J. Woo, S. Jeon, and J. Huh, "Shieldstore: Shielded in-memory key-value storage with sgx," in *Proceedings of the Fourteenth EuroSys Conference 2019*, 2019, pp. 1–15.

[6] C. Kockan, K. Zhu, N. Dokmai, N. Karpov, M. O. Kulekci, D. P. Woodruff, and S. C. Sahinalp, "Sketching algorithms for genomic data analysis and querying in a secure enclave," *Nature methods*, vol. 17, no. 3, pp. 295–301, 2020.

[7] S. Matetic, M. Ahmed, K. Kostiainen, A. Dhar, D. Sommer, A. Gervais, A. Juels, and S. Capkun, "ROTE: Rollback protection for trusted execution," in *26th USENIX Security*, 2017, pp. 1289–1306.

[8] J. Niu, W. Peng, X. Zhang, and Y. Zhang, "Narrator: Secure and practical state continuity for trusted execution in the cloud," in *29th ACM CCS*, 2022, pp. 2385–2399.

[9] Z. Ren, X. Li, S. Xu, and Y. Tong, "Restricting the number of times that data can be accessed in cloud storage using trustzone," in *2022 22nd IEEE International Symposium on Cluster, Cloud and Internet Computing (CCGrid)*. IEEE, 2022, pp. 289–296.

[10] M. N. Sadat, M. M. Al Aziz, N. Mohammed, F. Chen, X. Jiang, and S. Wang, "Safety: secure gwas in federated environment through a hybrid solution," *IEEE/ACM transactions on computational biology and bioinformatics*, vol. 16, no. 1, pp. 93–102, 2018.

[11] N. Siva, "1000 genomes project," *Nature biotechnology*, vol. 26, no. 3, pp. 256–257, 2008.

[12] Y. Tang, P. P. Lee, J. C. Lui, and R. Perlman, "Secure overlay cloud storage with access control and assured deletion," *IEEE Transactions on dependable and secure computing*, vol. 9, no. 6, pp. 903–916, 2012.

[13] F. Tramèr, Z. Huang, J.-P. Hubaux, and E. Ayday, "Differential privacy with bounded priors: reconciling utility and privacy in genome-wide association studies," in *22nd ACM CCS*, 2015, pp. 1286–1297.

[14] R. Wang, Y. F. Li, X. Wang, H. Tang, and X. Zhou, "Learning your identity and disease from research papers: information leaks in genome wide association study," in *16th ACM CCS*, 2009, pp. 534–544.

[15] J. Wei, X. Chen, J. Wang, X. Huang, and W. Susilo, "Securing fine-grained data sharing and erasure in outsourced storage systems," *IEEE Transactions on Parallel and Distributed Systems*, vol. 34, no. 2, pp. 552–566, 2023.