TRUST: A Toolkit for TEE-Assisted Secure Outsourced Computation over Integers

Bowen Zhao, *Member, IEEE*, Jiuhui Li, Peiming Xu*, Xiaoguo Li, Qingqi Pei, *Senior Member, IEEE*, and Yulong Shen, *Member, IEEE*

Abstract—Secure outsourced computation (SOC) provides secure computing services by taking advantage of the computation power of cloud computing and the technology of privacy computing (e.g., homomorphic encryption). Expanding computational operations on encrypted data (e.g., enabling complex calculations directly over ciphertexts) and broadening the applicability of SOC across diverse use cases remain critical yet challenging research topics in the field. Nevertheless, previous SOC solutions frequently lack the computational efficiency and adaptability required to fully meet evolving demands. To this end, in this paper, we propose a toolkit for TEE-assisted (Trusted Execution Environment) SOC over integers, named TRUST. In terms of system architecture, TRUST falls in a single TEE-equipped cloud server only through seamlessly integrating the computation of REE (Rich Execution Environment) and TEE. In consideration of TEE being difficult to permanently store data and being vulnerable to attacks, we introduce a (2, 2)-threshold homomorphic cryptosystem to fit the hybrid computation between REE and TEE. Additionally, we carefully design a suite of SOC protocols supporting unary, binary and ternary operations. To achieve applications, we present **SEAT**, secure data trading based on TRUST. Security analysis demonstrates that TRUST enables SOC, avoids collusion attacks among multiple cloud servers, and mitigates potential secret leakage risks within TEE (e.g., from side-channel attacks). Experimental evaluations indicate that TRUST outperforms the state-of-the-art and requires no alignment of data as well as any network communications. Furthermore, **SEAT** is as effective as the **Baseline** without any data protection.

Index Terms—Secure computing, privacy protection, TEE, homomorphic encryption, data trading.

I. INTRODUCTION

OUTSOURCING computation allows a user with limited resources to outsource operations (or rather, computations) over data to a cloud server. Secure outsourced computation (SOC) ensures computations over data without disclosing the data itself [1]. In other words, SOC provides not only convenient computing services for the user with limited resources, but also privacy protection for the outsourced data. Broadly speaking, many types of secure operations can be considered as SOC, for instance, searchable encryption [2], private set intersection [3], secure training or inference for machine learning [4], and secure computation as a service [5]. Thus, SOC can be applied to various fields including search, product recommendation, artificial intelligence, production optimization, etc.

Technically, a cloud server providing SOC services usually adopts privacy computing technologies to safeguard the privacy of outsourced data. Homomorphic encryption (HE), secure multi-party computation (MPC), and trusted execution environment (TEE) [6] are common privacy computing technologies. HE allows operations over encrypted data (also known as ciphertexts) directly and accesses no decryption key [7]. HE includes partially HE achieving addition or multiplication only and fully HE supporting addition and multiplication simultaneously. MPC enables multiple parties jointly performing an operation and safeguards each party's inputs [8]. SOC solutions based on multiple servers usually rely on MPC. Either HE or MPC achieves SOC through software method, while TEE is through a hardware method. TEE creates an enclave offering hardware-based memory encryption for code and data, in which no unauthorized entities from outside the enclave access inside code or data [9]. TEE offers a promising hardware-based approach to secure computation, achieving CPU-level performance and effectively bridging the gap between academic research and industrial adoption in the field.

Existing SOC solutions remain constrained by limited computational operations and often assume no collusion among cloud servers. On the one hand, SOC based on HE usually supports secure addition, secure multiplication, or both. However, in practice, a user requires more operations (e.g., secure comparison) offered by SOC to handle outsourced data. On the other hand, although SOC based on MPC offers more secure operations, it requires at least two cloud servers and assumes that there is no collusion between cloud servers. In this case, more cloud servers mean more deployment costs. Additionally, more network communications and extra data alignment operations are required to enable secure computations. Furthermore, arguably, the assumption of no collusion may be too strong, as it is difficult to determine whether cloud servers launch the collusion attack or not. SOC based on TEE enriches computational operations and avoids the collusion attack launched by multiple cloud servers [10]. Unfortunately,

Bowen zhao is with Guangzhou Institute of Technology, Xidian University, Guangzhou 510555, China and also with the State Key Laboratory of Public Big Data, Guizhou University, Guiyang 550025, China (e-mail: bwinzhao@gmail.com).

Jiuhui Li is with Guangzhou Institute of Technology, Xidian University, Guangzhou 510555, China (e-mail: lijiuhui@stu.xidian.edu.cn).

Peiming Xu is with Electric Power Research Institute, CSG, Guangzhou Guangdong, 510663, China, and also with Guangdong Provincial Key Laboratory of Power System Network Security, Guangzhou Guangdong, 510663, China (email: xupm@csg.cn).

Xiaoguo Li is with the College of Computer Science, Chongqing University, Chongqing 400044, China (e-mail: csxgli@cqu.edu.cn).

Qingqi Pei and Yulong Shen are with the State Key Laboratory of Integrated Service Networks, Xidian University, Xi'an 710126, China (e-mail: qqpei@mail.xidian.edu.cn, ylshen@mail.xidian.edu.cn).

(Corresponding author: Peiming Xu)

any enclave holds limited storage space, and its storage is random access memory (RAM) [11]. Thus, it is not trivial to permanently store and process outsourced data in an enclave. Furthermore, TEE is not always trusted because side-channel attacks are high likely to expose inside code and data in the enclave [12].

In order to mitigate the above limitations faced by SOC, in this work, we concentrate on devising a hybrid SOC framework by integrating advantages from the software-based method and the hardware-based method. Intuitively, this framework is confronted with several challenges. The **first** one is to balance the software method and the hardware method, and then eliminate collusion attacks. The software-based method and the hardware-based method follow entirely different designs achieving SOC. A basic operation is supported by the former but not by the latter, and vice versa. The **second** one is to fulfill secure computations under lacking a fully trusted enclave. Due to the degradation of trust in enclave, some excellent features provided by fully trusted enclave cannot be adopted. Thus, this significantly limits the functionality of the enclave. The **last** one is to enrich secure operations falling this framework. To eliminate the collusion attack, this framework should not utilize MPC to enable SOC, which restricts secure operations. Moreover, the enclave in this framework fails to enrich secure operations by reason of trust concerns. Worse, HE-based SOC only achieves simple secure operations.

In response to the aforementioned challenges, in this work, we propose TRUST[1], a toolkit for TEE-assisted SOC over integer. In brief, the proposed TRUST employs a single TEE-equipped cloud server, in which a rich execution environment (REE) of the cloud server cooperates with TEE to provide SOC services. A (2, 2)-threshold Paillier cryptosystem [13] is introduced in TRUST to construct SOC protocols. After that, we come up with secure data trading based on TRUST, called SEAT[2] to explore the application of SOC. Our innovations in this work involve three fronts.

- **Practical SOC framework**. We propose a practical SOC framework that relies on a single TEE host, where the TEE host calls a software-based method (i.e., the threshold Paillier cryptosystem [13]), and the enclave of TEE offers hardware-based capabilities. The TEE host and the enclave serve as the two parties of SOC and jointly perform SOC protocols. Particularly, the proposed SOC framework does not require a fully trusted enclave.
- **Rich SOC operations**. Following the proposed framework and assumption, we carefully design a suite of underlying SOC protocols encompassing unary, binary and ternary operations.
- **Feasible SOC application**. We apply the proposed TRUST to data trading and present a solution SEAT, which not only fulfills the outsourced task requirements but also effectively avoids data resale and privacy leakage.

The rest of this work is organized as follows. We briefly

---

review related work in Section II. Subsequently, Section III presents the preliminaries essential for constructing TRUST. After that, we present the system model and threat model of TRUST and elaborate on its design in Section IV and Section V, respectively. In Section VI and Section VII, the application and the security of TRUST are given, respectively. We extensively evaluate the efficiency and effectiveness of both TRUST and SEAT in Section VIII. Finally, we conclude this work in Section IX.

II. RELATED WORK

The methods for achieving SOC can be broadly classified into the software-based (e.g., HE, MPC) one and the hardware-based (e.g., TEE) one. In this section, we briefly review these two approaches.

A. Software-based SOC

Recently, significant advancements in cryptographic primitives and protocols have expanded secure computation capabilities, enabling diverse operations on ciphertexts (e.g., arithmetic, logical) and supporting various numeric representations (e.g., integers, floating-point numbers) [1] [14]. Despite continuous optimization and integration of technologies to enhance privacy and performance, achieving a balance between security and efficiency remains a fundamental challenge in privacy-preserving outsourced computation.

HE-based solutions (e.g., [15] [16] [17] [18]) enable direct computation on ciphertexts. Erkin *et al.* [15] implemented personalized recommendation generation over ciphertexts by jointly using two additively homomorphic cryptosystems, including the Paillier cryptosystem [19] and Damgård-Geisler-Krøigaard (DGK) [20]. The work [17] proposed a novel SkNN protocol, which is also based on the Paillier cryptosystem to enable k-nearest neighbor search over ciphertexts in the cloud, providing key protocols such as secure multiplication, etc., to accomplish this goal. Although the Paillier cryptosystem has the capability to extend operation types over ciphertexts, some of solutions (e.g., [17], [18]) suffer from stability issues, as they rely on entrusting the complete decryption key to a single decryption server, which introduces a critical single point of failure. While, other solutions (e.g., somewhat HE [21] and fully HE [22]) either lack support for arbitrary computations or remain impractical due to substantial computational overhead.

The integration of a twin-server architecture with HE has emerged as a novel paradigm, supporting arbitrary operations while mitigating the single point of security failure. The schemes in [7], [13], [23], [24] adopted the Paillier cryptosystem with threshold decryption [25] within a twin-server architecture to enrich computational operations over ciphertexts. Liu *et al.* [24] proposed a framework named POCF, which enhances operations over ciphertexts and facilitates collaborative decryption by distributing the Paillier cryptosystem private key between two servers, ensuring that neither can independently access the original data. Importantly, this design enables collaborative decryption by the twin-server, allowing the cloud to complete the SOC task and return the result to the client in a single round of communication, while preserving privacy.

Likewise, the work [7] implemented a toolkit for SOC on integers, named SOCI, which falls in a twin-server architecture utilizing the threshold Paillier cryptosystem. In particular, the toolkit provides more efficient secure multiplication and secure comparison protocols compared to those proposed by [24]. Furthermore, SOCI+, as proposed by the work [13], significantly outperforms SOCI [7], attributed to a more efficient variant of the threshold Paillier cryptosystem and an offline-online mechanism to enhance precomputation capabilities. However, all the aforementioned work fundamentally relies on the non-collusion assumption between the twin-server, which falls short of real-world requirements. Even multi-server applications using a (t, n)-threshold Paillier cryptosystem still require the servers to refrain from collusion.

MPC-based solutions, which are more inclined to extend SOC for specific applications, are represented by Yao's garbled circuits (GC) [26] and secret sharing (SS) [27], and enable multiple parties to jointly compute a result without revealing their private inputs. Blanton *et al.* [28] presented a secure sequence comparison scheme (i.e., edit distance) that employs GC in a novel non-black-box manner with two servers, suitable for privacy-preserving DNA alignment and other string matching tasks. A scheme for iris identification is proposed in [29], which enables encrypted iris code matching on untrusted servers while achieving privacy via predicate encryption in a single-server model or secure MPC with SS in a multi-server model. Additionally, ABY [30] and ABY 2.0 [31] employed SS to implement mixed frameworks for secure two-party computation, integrating arithmetic and boolean. Although these schemes support general computation for arbitrarily complex functions, they demand substantial storage and communication resources and always rely on the non-collusion assumption among parties. To avoid existing limitations, the design of TRUST aims to effectively balance computational efficiency and system generality, achieving an optimal synergy between the two.

B. Hardware-based SOC

Benefiting from the ability of TEE to prevent malicious software attacks and ensure privacy protection, along with strong guarantees of data confidentiality and integrity, TEE-based secure computations are emerging [9], [32].

Building on the above core capabilities, existing TEEs (e.g., Intel® Software Guard Extensions (SGX) [33], ARM's Trust-Zone) underpin advancements of secure computing research and trusted computing applications. Representative examples encompass privacy-preserving outsourced computation [10], secure credential migration [34], secure root-of-trust design [35], secure pipeline management [36], formal verification framework [37], secure electronic healthcare data sharing [38], robust federated learning [39], identity authentication and authorization services [40], among others.

Liu *et al.* [41] presented a SOC scheme integrating blockchain, smart contracts, a TEE-equipped cloud service provider, and a trusted authority responsible for managing cryptographic keys as a fully trusted third party. The scheme involves multi-round data transmission among entities, increasing communication overhead, while the deployment of extensive processing tasks on-chain introduces notable computational costs.

The previous works [10] combined SS with the threshold Paillier cryptosystem to design a fast processing toolkit on a single cloud server, aiming to resist side-channel attacks against TEE. Yet, this approach relies on multiple trusted processing units, leading to relatively high computation costs and frequent inter-server communication rounds during protocol execution. The work [42] put forward SDTE, a blockchain-based secure data trading system utilizing TEE and AES-256 symmetric encryption. However, it relies on TEE to fully decrypt data using symmetric keys and directly operates over plaintext, neglecting the vulnerability of TEE to side-channel attacks. More than the aforementioned work, many existing TEE-based solutions leverage its strong data protection capabilities, but they do not fully consider the limitations of TEE (e.g., restricted computational resources and scalability constraints), which hinder its applicability in complex, resource-intensive scenarios. TEE is not bullet-proof and has been successfully attacked numerous times in various ways, as shown in [43], [44], [45]. Undoubtedly, treating TEE as a trusted third party to directly perform operations on plaintext or to maintain a master decryption key fails to adequately consider the vulnerability to side-channel attacks, which could lead to privacy leakage and undermine its trustworthiness.

Inspired by the design proposed in the literatures [10] and [13], TRUST employs the (2, 2)-threshold Paillier cryptosystem and integrates software and hardware to maximize the security offered by cryptographic primitives and improve the computational efficiency of TEE leveraging CPU resources. TRUST seeks to overcome the real-world challenges in privacy-preserving computation, such as the limited storage resources of TEE and overly strong security assumptions (e.g., non-collusion among multiple servers and the absolute security of TEE), which are misaligned with practical scenarios.

III. PRELIMINARYS

In this section, we introduce the threshold Paillier cryptosystem and TEE, which serve as the foundation for the design of TRUST.

A. FastPaiTD

FastPaiTD proposed by the work [13], a $(2, 2)$-threshold Paillier cryptosystem, serves as the preliminarys of our proposed TRUST. FastPaiTD involves 5 probabilistic polynomial time algorithms listed as follows.

1) KGen (key generation): KGen takes a secure parameter κ (e.g., $\kappa = 112$) as an input and outputs a public/private pair (pk, sk) and two partially private keys (sk_1, sk_2). Formally, KGen$(1^\kappa) \rightarrow (pk, sk, sk_1, sk_2)$. Specifically, $pk = (N, h)$, $sk = \alpha$, and (sk_1, sk_2) satisfy $sk_1 + sk_2 = 0 \mod 2\alpha$ and $sk_1 + sk_2 = 1 \mod N$. More details about the above parameters can refer to the work [13]. Note that we set sk_1 as a σ-bit (e.g., $\sigma = 128$) random number and compute $sk_2 = 2\alpha \cdot (2\alpha)^{-1} \mod N - sk_1$.

2) Enc (encryption): Enc takes as input a message $m \in \mathbb{Z}_N$ and the public key pk, and outputs a ciphertext $[\![m]\!]$ as follows

$$[\![m]\!] \leftarrow (1+N)^m \cdot (h^r \bmod N)^N \bmod N^2, \qquad (1)$$

where r is a 4κ-bit random number. Formally, $\text{Enc}(pk, m) \rightarrow [\![m]\!]$.

3) Dec (decryption): Dec takes as input $[\![m]\!]$ adn the private key sk and outputs a message m as follows

$$m \leftarrow (\frac{[\![m]\!]^{2\alpha} \bmod N^2 - 1}{N} \bmod N) \cdot (2\alpha)^{-1} \bmod N. \quad (2)$$

Formally, $\text{Dec}(sk, [\![m]\!]) \rightarrow m$.

4) PDec (partial decryption): PDec takes as input $[\![m]\!]$ and a partially private key sk_i ($i \in \{1, 2\}$) and outputs a ciphertext as follows

$$[m]_i \leftarrow [\![m]\!]^{sk_i} \bmod N^2. \qquad (3)$$

Formally, $\text{PDec}(sk_i, [\![m]\!]) \rightarrow [m]_i$ for $i \in \{1, 2\}$.

5) TDec (threshold decryption): TDec takes as input $[m]_1$ and $[m]_2$ and outputs a message m as follows

$$m \leftarrow \frac{([m]_1 \cdot [m]_2 \bmod N^2) - 1}{N} \bmod N. \qquad (4)$$

Formally, $\text{TDec}([m]_1, [m]_2) \rightarrow m$.

Note that FastPaiTD supports additive homomorphism and scalar-multiplication homomorphism. Specifically,

- Additive homomorphism: Given two ciphertexts $[\![m_1]\!]$ and $[\![m_2]\!]$, $\text{Dec}(sk, [\![m_1 + m_2 \bmod N]\!] = \text{Dec}(sk, [\![m_1]\!] \cdot [\![m_2]\!] \bmod N^2)$ holds.
- Scalar-multiplication homomorphism: Given a ciphertext $[\![m]\!]$ and a constant $c \in \mathbb{Z}_N$, $\text{Dec}(sk, [\![c \cdot m \bmod N]\!] \bmod N^2) = \text{Dec}(sk, [\![m]\!]^c \bmod N^2)$ holds. Particularly, when $c = -1$, $\text{Dec}(sk, [\![m]\!]^c \bmod N^2) = -m$ holds.

B. Trusted Execution Environment (TEE)

TEE performs confidential computing by providing an isolated secure enclave within the processor, specifically designed to securely execute user-defined sensitive code in an untrusted host environment. More importantly, TEE provides robust protection for data and computation against any potentially malicious entities residing in the system (including the kernel, hypervisor, etc.) [9]. Notably, TEE-based secure computation has attracted significant interest from researchers in both academia and industry over the past two decades (e.g., [46], [47], [48], [49]), thanks to its several essential features.

- Attestation [50]. At its core, attestation establishes trust between an enclave and another enclave or a remote machine through a secure communication channel. It includes two mechanisms: local attestation, enabling mutual verification between enclaves on the same platform, and remote attestation, which allows clients to verify the integrity and configuration of their program in a remote server.
- Isolated Execution [51]. Applications leveraging TEE are divided into trusted and untrusted regions, with code and data in the trusted region executing within TEE, offering both confidentiality and integrity guarantees,
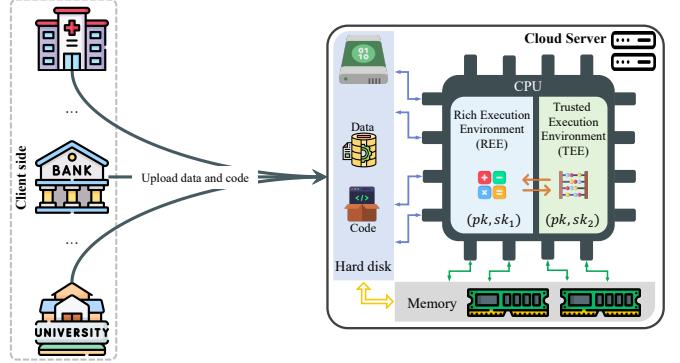

Fig. 1. System model.

while those in the untrusted region operate in REE and remain accessible to the operating system. User-defined programs are loaded into enclaves for isolated execution, where sensitive data is securely maintained in Enclave Page Caches (EPCs) within Processor Reserved Memory (PRM), which is pre-configured in Dynamic Random Access Memory (DRAM) during the bootstrapping phase. The code loaded inside an enclave can access both PRM and non-PRM memory, whereas external programs are restricted from accessing PRM. Programmer-defined entry points facilitate secure and adaptable communication between TEE and REE, enabling controlled interaction across these regions.

- Sealing and Unsealing [50]. Sealing and unsealing is a mechanism that allows a TEE to securely store and retrieve a secret in non-volatile external storage. On one hand, sealing uses a CPU-derived key, enabling TEE to bind the secret to a specific enclave identity, then encrypt and store it externally. On the other hand, unsealing enables TEE to retrieve and decrypt the secret from storage.

While TEE safeguards processed data by encrypting both incoming and outgoing data flows, research has shown that it remains vulnerable to various side-channel attacks, including those targeting page tables [43], [52], DRAM [44], [53], [54], and cache [45], [55], [56].

IV. SYSTEM MODEL AND THREAT MODEL

The system model formulates entities in a system and their abilities. As decipted in Fig. 1, our system model contains two entities, i.e., a client and a cloud server, where the cloud server is equipped a TEE chip.

- **Client.** The client is a user who has the requirement of SOC over ciphertexts. The user may be a hospital, a bank, or an university and so on. To this end, the user initializes the $(2, 2)$-threshold Paillier cryptosystem and generates the public key pk and two partially private keys sk_1 and sk_2. Additionally, the user assigns (pk, sk_1) to REE of the cloud server, and transmits (pk, sk_2) to the TEE of the cloud server via a secure channel. The user stores ciphertexts and code on the hard disk of the cloud server.
- **Cloud server.** The cloud server is required to equip a TEE on its CPU, and provides SOC services for the user

through its REE and TEE. Specifically, REE initiates a calculation and obtain the calculation result from TEE through cooperating with TEE.

The threat model assumes the attack capability of entities in the system model. In TRUST, we suppose the client is trusted. The REE and TEE of the cloud server are semi-honest, which means either REE or TEE follows protocols but tries to obtain the client's data and calculation results. Particularly, any adversary is allowed to corrupt REE or TEE, but cannot corrupt REE and TEE simultaneously.

V. TRUST DESIGN

In this section, we firstly formalize the SOC of TRUST. After that, a suite of SOC protocols integrated into TRUST is introduced.

A. Problem Formulation

Suppose encrypted outsourced data $[\![m_1]\!], [\![m_2]\!], \cdots, [\![m_n]\!]$ require to perform SOC denoted by $\mathcal{F}$, where n is the number of operands and $n \geq 1$. Our proposed TRUST assumes f_r and f_t denote the computation over REE and TEE, respectively. Particularly, TRUST formalizes the SOC $\mathcal{F}([\![m_1]\!], \cdots, [\![m_n]\!])$ as follows.

Definition 1. *Suppose a SOC be denoted by $\mathcal{F}([\![m_1]\!], \cdots, [\![m_n]\!])$, where n is the size of outsourced data, TRUST formulates it as a two-party computation between REE and TEE denoted by*

$$\mathcal{F}([\![m_1]\!], \cdots, [\![m_n]\!]) \Longleftrightarrow f_t(key_t, f_r(key_r, [\![m_1]\!], \cdots, [\![m_n]\!])). \tag{5}$$

key_r and key_t indicate the key of REE and TEE, respectively. Note that f_t can be omitted when f_r enables computations required by $\mathcal{F}$.

From Eq. (5), we see that only REE accesses outsourced data $[\![m_1]\!], \cdots, [\![m_n]\!]$ directly, while TEE takes the output of REE as an input. According to the system model of TRUST, key_r and key_t denote (pk, sk_1) and (pk, sk_2), respectively, thus, the output of f_r is encrypted data. Note that when $n = 1$, $\mathcal{F}([\![m_1]\!])$ means a unary operation (e.g., absolute value, factorial). $\mathcal{F}([\![m_1]\!], [\![m_2]\!])$ $(n = 2)$ indicates a binary operation (e.g., addition, multiplication, comparison), while $\mathcal{F}([\![m_1]\!], [\![m_2]\!], [\![m_3]\!])$ $(n = 3)$ denotes a ternary operation (e.g., ternary conditional operator), and so on.

Fig. 2 shows a typical workflow of TRUST. As depicted in Fig. 2, TRUST reconstructs the computation $\mathcal{F}([\![m_1]\!], \cdots, [\![m_n]\!])$ into $f_r(pk, sk_1, [\![m_1]\!], \cdots, [\![m_n]\!]) \rightarrow [\![R_0]\!]$ and $f_t(pk, sk_2, [\![R_0]\!]) \rightarrow \mathcal{F}([\![m_1]\!], \cdots, [\![m_n]\!])$ (Case 1), or $f_r(pk, [\![m_1]\!], \cdots, [\![m_n]\!])$ (Case 2), where $[\![R_0]\!]$ is an intermediate result, and n means the amount of operands.

B. Protocols Design

In this section, we detail operations including multiplication, comparison, equality, absolute value, ternary conditional operator, etc. supported by TRUST. Obviously, TRUST is easy to achieve addition and scalar-multiplication as they are homomorphic features of FastPaiTD. Take addition as an
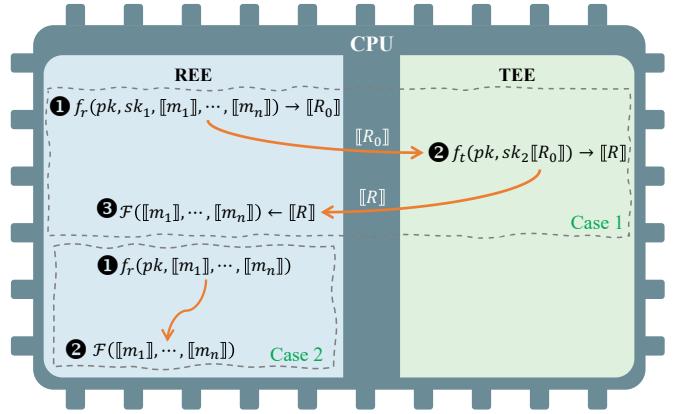

Fig. 2. Typical workflow of TRUST.

example, $\mathcal{F}([\![m_1]\!], [\![m_2]\!])$ is formulated as $\mathcal{F}([\![m_1]\!], [\![m_2]\!]) \rightarrow [\![m_1+m_2]\!]$. In this case, f_r is formulated as $f_r([\![m_1]\!], [\![m_2]\!]) = [\![m_1]\!] \cdot [\![m_2]\!] \mod N^2$, while f_t can be omitted. Following the prior work [7], we assume $m_1, m_2, \cdots, m_n \in (-2^\ell, 2^\ell)$, where ℓ is a constant (e.g., $\ell = 32$).

Algorithm 1 shows a secure multiplication algorithm following Definition 1. Formally, $\mathcal{F}([\![m_1]\!], \cdots, [\![m_n]\!])$ is denoted by $\mathcal{F}_{\mathsf{mul}}([\![m_1]\!], [\![m_2]\!]) \rightarrow [\![m_1 \cdot m_2]\!]$, where $n = 2$. At Step 1, REE takes as input keys (pk and sk_1), $[\![m_1]\!]$, and $[\![m_2]\!]$, and outputs $[\![m_1+r]\!]$, $[m_1+r]_1$, $[\![m_2]\!]$, and $[\![-r \cdot m_2]\!]$, where r is a σ-bit (e.g., $\sigma = 128$) random number, and $\leftarrow\$$ means random sampling in this work.

After receiving the output of REE, TEE firstly obtains $m_1 + r$ by calling PDec and TDec sequentially and then outputs the final result $[\![R]\!]$. According to the homomorphic properties of FastPaiTD, we have

$$[\![m_2]\!]^{m_1+r} \cdot [\![-r \cdot m_2]\!] \cdot \mathsf{Enc}(pk, 0) \rightarrow [\![m_1 \cdot m_2 + r \cdot m_2 - r \cdot m_2 + 0]\!]. \tag{6}$$

Thus, it is easy to verify $R = m_1 \cdot m_2$. In other words, $\mathcal{F}_{\mathsf{mul}}$ outputs $[\![m_1 \cdot m_2]\!]$ correctly.

Algorithm 2 depicts a secure comparison algorithm following Definition 1. Formally, $\mathcal{F}([\![m_1]\!], \cdots, [\![m_n]\!])$ is denoted by $\mathcal{F}_{\mathsf{cmp}}([\![m_1]\!], [\![m_2]\!]) \rightarrow [\![\mu]\!]$, where $\mu = 0$ when $m_1 \geq m_2$, otherwise ($m_1 < m_2$), $\mu = 1$. At Step 1, REE takes as input keys (pk and sk_1), $[\![m_1]\!]$, and $[\![m_2]\!]$, and outputs $[\![d]\!]$, $[d]_1$, and $[\![\pi]\!]$. Specifically, REE firstly generates r_1 and r_2 ($r_1 + r_2 > \frac{N}{2}$) by random sampling. After that, REE randomly samples π from $\{0, 1\}$ and computes $[\![d]\!]$. According to the homomorphic properties of FastPaiTD, we have

$$d = \begin{cases} r_1 \cdot (m_1 - m_2 + 1) + r_2, & \pi = 0; \\ r_1 \cdot (m_2 - m_1) + r_2. & \pi = 1; \end{cases} \tag{7}$$

After receiving the output of REE, TEE firstly obtains d by calling PDec and TDec sequentially. Then, TEE computes and returns $[\![R]\!]$ to REE. It is easy to verify

$$R = \mu_0 + (1 - 2\mu_0) \cdot \pi. \tag{8}$$

If $\mu_0 = 0$ (i.e., $d > \frac{N}{2}$), we have $R = \pi$, whereas $\mu_0 = 1$ (i.e., $d < \frac{N}{2}$), we have $R = 1 - \pi$. As shown in Eq. (8), the value of R depends on the ones μ_0 and π, while the one μ_0 depends on that d.

Algorithm 1: $\mathcal{F}_{\mathsf{mul}}(\llbracket m_1 \rrbracket, \llbracket m_2 \rrbracket) \to \llbracket m_1 \cdot m_2 \rrbracket$

Input: $\llbracket m_1 \rrbracket$ and $\llbracket m_2 \rrbracket$.
Output: $\llbracket m_1 \cdot m_2 \rrbracket$.

▷ Step 1: Computations in REE

- $\llbracket m_1 + r \rrbracket \leftarrow \llbracket m_1 \rrbracket \cdot \mathsf{Enc}(pk, r)$, where $r \leftarrow \$ \{0,1\}^\sigma$;
- $\llbracket -r \cdot m_2 \rrbracket \leftarrow \llbracket m_2 \rrbracket^{-r}$;
- $\mathsf{PDec}(sk_1, \llbracket m_1 + r \rrbracket) \to [m_1 + r]_1$;

REE $\quad \langle \llbracket m_1 + r \rrbracket, [m_1+r]_1, \llbracket m_2 \rrbracket, \llbracket -r \cdot m_2 \rrbracket \rangle \quad$ TEE

▷ Step 2: Computations in TEE

- $\mathsf{PDec}(sk_2, \llbracket m_1 + r \rrbracket) \to [m_1 + r]_2$;
- $\mathsf{TDec}([m_1+r]_1, [m_1+r]_2) \to m_1 + r$;
- $\llbracket R \rrbracket \leftarrow \llbracket m_2 \rrbracket^{m_1+r} \cdot \llbracket -r \cdot m_2 \rrbracket \cdot \mathsf{Enc}(pk, 0)$;

REE $\quad \xleftarrow{\quad \llbracket R \rrbracket \quad} \quad$ TEE

▷ Step 3: Computations in REE

- $\mathcal{F}_{\mathsf{mul}} \leftarrow \llbracket R \rrbracket$;

Algorithm 2: $\mathcal{F}_{\mathsf{cmp}}(\llbracket m_1 \rrbracket, \llbracket m_2 \rrbracket) \to \llbracket \mu \rrbracket$

Input: $\llbracket m_1 \rrbracket$ and $\llbracket m_2 \rrbracket$.
Output: $\llbracket \mu \rrbracket$, if $m_1 \geq m_2$, $\mu = 0$, otherwise, $\mu = 1$.

▷ Step 1: Computations in REE

- $r_1 \leftarrow \$ \{0,1\}^\sigma$ and $r_2 \leftarrow \$ (\frac{N}{2} - r_1, \frac{N}{2})$;
- $\pi \leftarrow \$ \{0,1\}$,

$$\llbracket d \rrbracket \leftarrow \begin{cases} \llbracket m_1 \rrbracket^{r_1} \cdot \llbracket m_2 \rrbracket^{-r_1} \cdot \mathsf{Enc}(pk, r_1 + r_2), & \pi = 0; \\ \llbracket m_1 \rrbracket^{-r_1} \cdot \llbracket m_2 \rrbracket^{r_1} \cdot \mathsf{Enc}(pk, r_2), & \pi = 1; \end{cases}$$

- $\mathsf{PDec}(sk_1, \llbracket d \rrbracket) \to [d]_1$ and $\mathsf{Enc}(pk, \pi) \to \llbracket \pi \rrbracket$;

REE $\quad \xrightarrow{\quad \langle \llbracket d \rrbracket, [d]_1, \llbracket \pi \rrbracket \rangle \quad} \quad$ TEE

▷ Step 2: Computations in TEE

- $\mathsf{PDec}(sk_2, \llbracket d \rrbracket) \to [d]_2$ and $\mathsf{TDec}([d]_1, [d]_2) \to d$;

$$\begin{cases} \mu_0 = 0 \text{ and } \llbracket \mu_0 \rrbracket \leftarrow \mathsf{Enc}(pk, 0), & d > \frac{N}{2}; \\ \mu_0 = 1 \text{ and } \llbracket \mu_0 \rrbracket \leftarrow \mathsf{Enc}(pk, 1), & d < \frac{N}{2}; \end{cases}$$

- $\llbracket R \rrbracket \leftarrow \llbracket \mu_0 \rrbracket \cdot \llbracket \pi \rrbracket^{1-2\mu_0}$;

REE $\quad \xleftarrow{\quad \llbracket R \rrbracket \quad} \quad$ TEE

▷ Step 3: Computations in REE

- $\mathcal{F}_{\mathsf{cmp}} \leftarrow \llbracket R \rrbracket$;

Case 1 ($\pi = 0$): If $r_1 \cdot (m_1 - m_2 + 1) + r_2 > \frac{N}{2}$ (i.e., $d > \frac{N}{2}$), we have $(m_1 - m_2 + 1) \geq 1$ because $r_1 + r_2 > \frac{N}{2}$ and $r_2 < \frac{N}{2}$. In this case, $m_1 \geq m_2$, while $R = \mu_0 = 0$. If $r_1 \cdot (m_1 - m_2 + 1) + r_2 < \frac{N}{2}$ (i.e., $d < \frac{N}{2}$), we have $(m_1 - m_2 + 1) \leq 0$ because $r_1 + r_2 > \frac{N}{2}$ and $r_2 < \frac{N}{2}$. In this case, $m_1 < m_2$, while $R = \mu_0 = 1$. In other words, when $\pi = 0$, if $m_1 \geq m_2$, $\mathcal{F}_{\mathsf{cmp}}$ outputs $\mu = 0$, otherwise ($m_1 < m_2$), outputs $\mu = 1$.

Case 2 ($\pi = 1$): If $r_1 \cdot (m_2 - m_1) + r_2 > \frac{N}{2}$ (i.e., $d > \frac{N}{2}$), we have $m_2 - m_1 \geq 1$ because $r_1 + r_2 > \frac{N}{2}$ and $r_2 < \frac{N}{2}$. In this case, $m_1 < m_2$, while $R = 1 - \mu_0 = 1$. If $r_1 \cdot (m_2 - m_2) + r_2 < \frac{N}{2}$ (i.e., $d < \frac{N}{2}$), we have $m_2 - m_1 \leq 0$ because $r_1 + r_2 > \frac{N}{2}$ and $r_2 < \frac{N}{2}$. In this case, $m_1 \geq m_2$, while $R = 1 - \mu_0 = 0$. In other words, when $\pi = 1$, if $m_1 \geq m_2$, $\mathcal{F}_{\mathsf{cmp}}$ outputs $\mu = 0$, otherwise ($m_1 < m_2$), outputs $\mu = 1$.

Taken together, either $\pi = 0$ or $\pi = 1$, $\mathcal{F}_{\mathsf{cmp}}$ always computes μ correctly.

Algorithms 1 and 2 denote a linear secure operation and a non-linear secure operation, respectively. In addition to the non-linear comparison operation, a equality operation is also common. Algorithm 3 lists the detail of a secure equality algorithm. Formally, $\mathcal{F}(\llbracket m_1 \rrbracket, \cdots, \llbracket m_n \rrbracket)$ is denoted by $\mathcal{F}_{\mathsf{eql}}(\llbracket m_1 \rrbracket, \llbracket m_2 \rrbracket) \to \llbracket \mu \rrbracket$, where $\mu = 0$ if $m_1 = m_2$ holds, otherwise ($m_1 \neq m_2$), $\mu = 1$. $\mathcal{F}_{\mathsf{eql}}$ is concreted to two critical steps. At Step 1, REE computes $\llbracket d_1 \rrbracket$ and $\llbracket d_2 \rrbracket$ in the same way as $\llbracket d \rrbracket$ in Algorithm 2, and outputs $\llbracket d_1 \rrbracket$, $[d_1]_1$, $\llbracket \pi_1 \rrbracket$, $\llbracket d_2 \rrbracket$, $[d_2]_1$, and $\llbracket \pi_2 \rrbracket$. According to the homomorphic properties of FastPaiTD, we have

$$d_1 = \begin{cases} r_1 \cdot (m_1 - m_2 + 1) + r_2, & \pi = 0; \\ r_1 \cdot (m_2 - m_1) + r_2, & \pi = 1; \end{cases} \quad (9)$$

$$d_2 = \begin{cases} r'_1 \cdot (m_2 - m_1 + 1) + r'_2, & \pi = 0; \\ r'_1 \cdot (m_1 - m_2) + r'_2. & \pi = 1; \end{cases} \quad (10)$$

At Step 2, TEE obtains d_1 and d_2 by calling PDec and TDec. Essentially, d_1 and d_2 implies the magnitude relationship between m_1 and m_2, and the one between m_2 and m_1, respectively. Mathematically, $m_1 = m_2$ is equivalent to $m_1 \geq m_2$ & $m_2 \geq m_1$. Inspired by this idea, at Step 2, TEE computes $\llbracket R \rrbracket$, where

$$R = \mu_0 + (1 - 2\mu_0) \cdot \pi_1 + \mu'_0 + (1 - 2\mu'_0) \cdot \pi_2. \quad (11)$$

According to Algorithm 2, if $m_1 \geq m_2$, $\mu_0 + (1 - 2\mu_0) \cdot \pi_1 = 0$. Additionally, if $m_2 \geq m_1$, $\mu'_0 + (1 - 2\mu'_0) \cdot \pi_2 = 0$. And since whether it is $\mu_0 + (1 - 2\mu_0) \cdot \pi_1$ or $\mu'_0 + (1 - 2\mu'_0) \cdot \pi_2$, it can only result in 0 and 1. Furthermore, $\mu_0 + (1 - 2\mu_0) \cdot \pi_1$ and $\mu'_0 + (1 - 2\mu'_0) \cdot \pi_2$ cannot both are equal to 1 as $m_1 < m_2$ & $m_2 < m_1$ cannot be true at the same time. Therefore, $R = 0$ if and only if $m_1 \geq m_2$ & $m_2 \geq m_1$ (i.e., $m_1 = m_2$). Also, $R = 1$ if and only if $m_1 > m_2$ & $m_2 < m_1$ or $m_1 < m_2$ & $m_2 > m_1$. In other words, if $m_1 = m_2$, $\mathcal{F}_{\mathsf{eql}}$ outputs $\llbracket 0 \rrbracket$, otherwise $m_1 \neq m_2$ (i.e., $m_1 > m_2$ or $m_2 > m_1$), it outputs $\llbracket 1 \rrbracket$. Thus, we say that $\mathcal{F}_{\mathsf{eql}}$ outputs $\llbracket \mu \rrbracket$ correctly.

Now, we discuss one unary operation (i.e., absolute value). If $m \geq 0$, $|m| = m$, otherwise ($m < 0$), $|m| = -m$. Thus, $|m|$ can be denoted by $|m| = (1 - 2\mu) \cdot m$, where $\mu = 0$ when $m \geq 0$, otherwise ($m < 0$), $\mu = 1$.

Algorithm 3: $\mathcal{F}_{\mathsf{eql}}(\llbracket m_1 \rrbracket, \llbracket m_2 \rrbracket) \to \llbracket \mu \rrbracket$

Input: $\llbracket m_1 \rrbracket$ and $\llbracket m_2 \rrbracket$.
Output: $\llbracket \mu \rrbracket$, if $m_1 = m_2$, $\mu = 0$, otherwise, $\mu = 1$.

▷ Step 1: Computations in REE

- $r_1 \leftarrow\$ \{0,1\}^\sigma, r_2 \leftarrow\$ (\frac{N}{2} - r_1, \frac{N}{2})$ and
  $r_1' \leftarrow\$ \{0,1\}^\sigma, r_2' \leftarrow\$ (\frac{N}{2} - r_1', \frac{N}{2})$;
- $\pi_1 \leftarrow\$ \{0,1\}$ and $\pi_2 \leftarrow\$ \{0,1\}$,

$$\llbracket d_1 \rrbracket \leftarrow \begin{cases} \llbracket m_1 \rrbracket^{r_1} \cdot \llbracket m_2 \rrbracket^{-r_1} \cdot \mathsf{Enc}(pk, r_1 + r_2), & \pi_1 = 0; \\ \llbracket m_1 \rrbracket^{-r_1} \cdot \llbracket m_2 \rrbracket^{r_1} \cdot \mathsf{Enc}(pk, r_2), & \pi_1 = 1; \end{cases}$$

$$\llbracket d_2 \rrbracket \leftarrow \begin{cases} \llbracket m_2 \rrbracket^{r_1'} \cdot \llbracket m_1 \rrbracket^{-r_1'} \cdot \mathsf{Enc}(pk, r_1' + r_2'), & \pi_2 = 0; \\ \llbracket m_2 \rrbracket^{-r_1'} \cdot \llbracket m_1 \rrbracket^{r_1'} \cdot \mathsf{Enc}(pk, r_2'), & \pi_2 = 1; \end{cases}$$

- $\mathsf{PDec}(sk_1, \llbracket d_1 \rrbracket) \to [d_1]_1, \mathsf{Enc}(pk, \pi_1) \to \llbracket \pi_1 \rrbracket$ and
  $\mathsf{PDec}(sk_1, \llbracket d_2 \rrbracket) \to [d_2]_1, \mathsf{Enc}(pk, \pi_2) \to \llbracket \pi_2 \rrbracket$;

REE $\quad \langle \llbracket d_1 \rrbracket, [d_1]_1, \llbracket \pi_1 \rrbracket, \llbracket d_2 \rrbracket, [d_2]_1, \llbracket \pi_2 \rrbracket \rangle \longrightarrow$ TEE

▷ Step 2: Computations in TEE

- $\mathsf{PDec}(sk_2, \llbracket d_1 \rrbracket) \to [d_1]_2, \mathsf{TDec}([d_1]_1, [d_1]_2) \to d_1$ and
  $\mathsf{PDec}(sk_2, \llbracket d_2 \rrbracket) \to [d_2]_2, \mathsf{TDec}([d_2]_1, [d_2]_2) \to d_2$;

- $\begin{cases} \mu_0 = 0 \text{ and } \llbracket \mu_0 \rrbracket \leftarrow \mathsf{Enc}(pk, 0), & d_1 > \frac{N}{2}; \\ \mu_0 = 1 \text{ and } \llbracket \mu_0 \rrbracket \leftarrow \mathsf{Enc}(pk, 1), & d_1 < \frac{N}{2}; \\ \mu_0' = 0 \text{ and } \llbracket \mu_0' \rrbracket \leftarrow \mathsf{Enc}(pk, 0), & d_2 > \frac{N}{2}; \\ \mu_0' = 1 \text{ and } \llbracket \mu_0' \rrbracket \leftarrow \mathsf{Enc}(pk, 1), & d_2 < \frac{N}{2}; \end{cases}$

- $\llbracket R \rrbracket \leftarrow \llbracket \mu_0 \rrbracket \cdot \llbracket \pi_1 \rrbracket^{1-2\mu_0} \cdot \llbracket \mu_0' \rrbracket \cdot \llbracket \pi_2 \rrbracket^{1-2\mu_0'}$;

REE $\quad \overleftarrow{\qquad \llbracket R \rrbracket \qquad}$ TEE

▷ Step 3: Computations in REE

- $\mathcal{F}_{\mathsf{eql}} \leftarrow \llbracket R \rrbracket$;

As $|m_1| = (1 - 2\mu) \cdot m_1$, where $\mu = 0$ when $m_1 \geq 0$, otherwise, $\mu = 1$, according to Algorithm 2, we see $\mu = \mu_0 + (1 - 2\mu_0) \cdot \pi$. Then, we have

$$\begin{aligned} |m_1| &= [1 - 2 \cdot (\mu_0 + (1 - 2\mu_0) \cdot \pi)] \cdot m_1 \\ &= (1 - 2\mu_0) \cdot m_1 + (4\mu_0 - 2) \cdot \pi \cdot m_1 \\ &= R. \end{aligned} \tag{14}$$

Thus, we can say that $\mathcal{F}_{\mathsf{abs}}$ oputputs $\llbracket |m_1| \rrbracket$ correctly.

Algorithm 4: $\mathcal{F}_{\mathsf{abs}}(\llbracket m_1 \rrbracket) \to \llbracket |m_1| \rrbracket$

Input: $\llbracket m_1 \rrbracket$.
Output: $\llbracket |m_1| \rrbracket$.

▷ Step 1: Computations in REE

- $r_1 \leftarrow\$ \{0,1\}^\sigma$ and $r_2 \leftarrow\$ (\frac{N}{2} - r_1, \frac{N}{2})$;
- $\pi \leftarrow\$ \{0,1\}$,

$$\llbracket d \rrbracket \leftarrow \begin{cases} \llbracket m_1 \rrbracket^{r_1} \cdot \mathsf{Enc}(pk, r_1 + r_2), & \pi = 0; \\ \llbracket m_1 \rrbracket^{-r_1} \cdot \mathsf{Enc}(pk, r_2), & \pi = 1; \end{cases}$$

- $\mathsf{PDec}(sk_1, \llbracket d \rrbracket) \to [d]_1, \llbracket \pi \cdot m_1 \rrbracket \leftarrow \llbracket m_1 \rrbracket^\pi \cdot \mathsf{Enc}(pk, 0)$;

REE $\quad \langle \llbracket d \rrbracket, [d]_1, \llbracket m_1 \rrbracket, \llbracket \pi \cdot m_1 \rrbracket \rangle \longrightarrow$ TEE

▷ Step 2: Computations in TEE

- $\mathsf{PDec}(sk_2, \llbracket d \rrbracket) \to [d]_2$ and $\mathsf{TDec}([d]_1, [d]_2) \to d$;

- $\mu_0 = \begin{cases} 0, & d > \frac{N}{2}; \\ 1, & d < \frac{N}{2}; \end{cases}$

- $\llbracket R \rrbracket \leftarrow \llbracket m_1 \rrbracket^{1-2\mu_0} \cdot \llbracket \pi \cdot m_1 \rrbracket^{4\mu_0 - 2} \cdot \mathsf{Enc}(pk, 0)$;

REE $\quad \overleftarrow{\qquad \llbracket R \rrbracket \qquad}$ TEE

▷ Step 3: Computations in REE

- $\mathcal{F}_{\mathsf{abs}} \leftarrow \llbracket R \rrbracket$;

Inspired by the above observation, Algorithm 4 depicts a secure absolute value algorithm. Formally, $\mathcal{F}(\llbracket m_1 \rrbracket)$ is denoted by $\mathcal{F}_{\mathsf{abs}}(\llbracket m_1 \rrbracket) \to \llbracket |m| \rrbracket$, where n is set as $n = 1$. Specifically, at Step 1, REE takes as input keys and $\llbracket m_1 \rrbracket$, and outputs four ciphertexts including $\llbracket d \rrbracket$, $[d]_1$, $\llbracket m_1 \rrbracket$, and $\llbracket \pi \cdot m_1 \rrbracket$, where $\llbracket \pi \cdot m_1 \rrbracket \leftarrow \llbracket m_1 \rrbracket^\pi \cdot \mathsf{Enc}(pk, 0)$. According to the homomorphic properties of FastPaiTD, we see

$$d = \begin{cases} r_1 \cdot (m_1 + 1) + r_2, & \pi = 0; \\ r_1 \cdot (-m_1) + r_2. & \pi = 1; \end{cases} \tag{12}$$

At Step 2, REE obtains d by calling PDec and TDec. From Eq.(12), we see that d essentially implies the magnitude relationship between m_1 and 0. After that, TEE computes and returns $\llbracket R \rrbracket$ to REE. It is easy to verify

$$R = (1 - 2\mu_0) \cdot m_1 + (4\mu_0 - 2) \cdot \pi \cdot m_1, \tag{13}$$

where $\mu_0 = 0$ when $d > \frac{N}{2}$, otherwise, $\mu_0 = 1$.

Finally, we describe a secure ternary operation (i.e., ternary conditional operator). Formally, the ternary conditional operation can be formulated as "*if a then b else c*". Without loss of generality, $\mathcal{F}(\llbracket m_1 \rrbracket, \llbracket m_2 \rrbracket, \llbracket m_3 \rrbracket)$ is denoted by $\mathcal{F}_{\mathsf{trn}}(\llbracket m_1 \rrbracket, \llbracket m_2 \rrbracket, \llbracket m_3 \rrbracket) \to \llbracket m \rrbracket$, where if $m_1 = 1$, then $m = m_2$, else $m = m_3$. Note that $m = m_2 + \mu \cdot (m_3 - m_2)$ is always true if $\mu = 0$ when $m_1 = 1$, otherwise $(m_1 \neq 1)$, $\mu = 1$.

Algorithm 5 shows the detail of the secure ternary operation. Specifically, at Step 1, REE computes $\llbracket d_1 \rrbracket$ and $\llbracket d_2 \rrbracket$ in the same way as those in Algorithm 3, and outputs eight ciphertexts, where $\llbracket \pi_1 \cdot (m_3 - m_2) \rrbracket \leftarrow \llbracket m_3 - m_2 \rrbracket^{\pi_1} \cdot \mathsf{Enc}(pk, 0)$, and $\llbracket \pi_2 \cdot (m_3 - m_2) \rrbracket \leftarrow \llbracket m_3 - m_2 \rrbracket^{\pi_2} \cdot \mathsf{Enc}(pk, 0)$.

At Step 2, TEE firstly obtains d_1 and d_2 by calling PDec and TDec. d_1 and d_2 essentially implies the magnitude relationship between m_1 and 1 the one between 1 and m_1, respectively.

After that, TEE computes $[\![R]\!]$. According to Eq. (13), it is easy to verify that

$$
\begin{aligned}
R =\ & m_2 + (\mu_0 + \mu_0') \cdot (m_3 - m_2) + (1 - 2\mu_0) \cdot \pi_1 \cdot \cdot (m_3 - m_2) \\
& + (1 - 2\mu_0') \cdot \pi_2 (m_3 - m_2) \\
=\ & [(\mu_0 + \mu_0') + (1 - 2\mu_0) \cdot \pi_1 + (1 - 2\mu_0') \cdot \pi_2] \cdot (m_3 - m_2) \\
& + m_2 \\
=\ & m_2 + \mu \cdot (m_3 - m_2), \qquad\qquad\qquad\qquad (15)
\end{aligned}
$$

where $\mu = 0$ if $m_1 = 1$, otherwise, $\mu = 1$. Thus, we say $\mathcal{F}_{\mathsf{trn}}$ outputs $[\![m]\!]$ correctly.

Algorithm 5: $\mathcal{F}_{\mathsf{trn}}([\![m_1]\!], [\![m_2]\!], [\![m_3]\!]) \to [\![m]\!]$

Input: $[\![m_1]\!], [\![m_2]\!],$ and $[\![m_3]\!].$
Output: $[\![m]\!],$ if $m_1 = 1,$ then $m = m_2,$ else $m = m_3.$

▷ Step 1: Computations in REE

- $r_1 \leftarrow\!\$ \{0,1\}^\sigma, r_2 \leftarrow\!\$ (\frac{N}{2} - r_1, \frac{N}{2})$ and
  $r_1' \leftarrow\!\$ \{0,1\}^\sigma, r_2' \leftarrow\!\$ (\frac{N}{2} - r_1', \frac{N}{2});$
- $\pi_1 \leftarrow\!\$ \{0,1\}$ and $\pi_2 \leftarrow\!\$ \{0,1\},$

$$
[\![d_1]\!] \leftarrow
\begin{cases}
[\![m_1]\!]^{r_1} \mathsf{Enc}(pk, r_2), & \pi_1 = 0; \\
[\![m_1]\!]^{-r_1} \cdot \mathsf{Enc}(pk, r_1 + r_2), & \pi_1 = 1;
\end{cases}
$$

$$
[\![d_2]\!] \leftarrow
\begin{cases}
[\![m_1]\!]^{-r_1'} \cdot \mathsf{Enc}(pk, 2r_1' + r_2'), & \pi_2 = 0; \\
[\![m_1]\!]^{r_1'} \cdot \mathsf{Enc}(pk, r_2' - r_1'), & \pi_2 = 1;
\end{cases}
$$

- $\mathsf{PDec}(sk_1, [\![d_1]\!]) \to [d_1]_1$ and $\mathsf{PDec}(sk_1, [\![d_2]\!]) \to [d_2]_1;$

- $[\![m_3 - m_2]\!] \leftarrow [\![m_3]\!] \cdot [\![m_2]\!]^{-1}, [\![\pi_1 \cdot (m_3 - m_2)]\!] \leftarrow [\![m_3 - m_2]\!]^{\pi_1} \cdot \mathsf{Enc}(pk, 0),$ and $[\![\pi_2 \cdot (m_3 - m_2)]\!] \leftarrow [\![m_3 - m_2]\!]^{\pi_2} \cdot \mathsf{Enc}(pk, 0);$

REE $\quad \langle [\![m_3 - m_2]\!], [\![d_1]\!], [d_1]_1, [\![\pi_1 \cdot (m_3 - m_2)]\!],$ TEE
$\qquad\quad [\![m_2]\!], [\![d_2]\!], [d_2]_1, [\![\pi_2 \cdot (m_3 - m_2)]\!] \rangle \longrightarrow$

▷ Step 2: Computations in TEE

- $\mathsf{PDec}(sk_2, [\![d_1]\!]) \to [d_1]_2, \mathsf{TDec}([d_1]_1, [d_1]_2) \to d_1$ and $\mathsf{PDec}(sk_2, [\![d_2]\!]) \to [d_2]_2, \mathsf{TDec}([d_2]_1, [d_2]_2) \to d_2;$

$$
\begin{cases}
\mu_0 = 0, & d_1 > \frac{N}{2}; \\
\mu_0 = 1, & d_1 < \frac{N}{2}; \\
\mu_0' = 0, & d_2 > \frac{N}{2}; \\
\mu_0' = 1, & d_2 < \frac{N}{2};
\end{cases}
$$

- $[\![R]\!] \leftarrow [\![m_2]\!] \cdot [\![m_3 - m_2]\!]^{\mu_0 + \mu_0'} \cdot [\![\pi_1 \cdot (m_3 - m_2)]\!]^{1 - 2\mu_0} \cdot [\![\pi_2 \cdot (m_3 - m_2)]\!]^{1 - 2\mu_0'} \cdot \mathsf{Enc}(pk, 0);$

REE $\qquad\qquad \xleftarrow{\quad [\![R]\!] \quad}$ TEE

▷ Step 3: Computations in REE

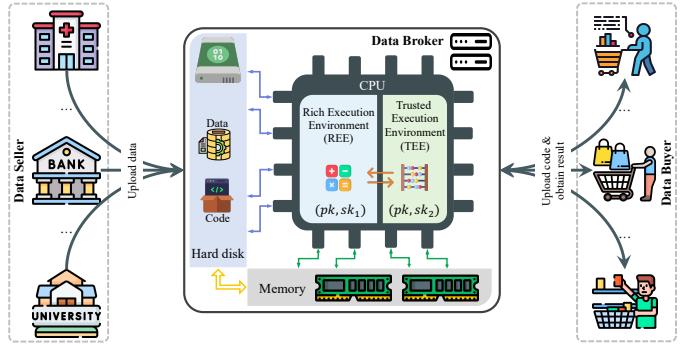- $\mathcal{F}_{\mathsf{trn}} \leftarrow [\![R]\!];$

Fig. 3. System architecture of SEAT.

VI. SEAT Details

In this section, we describe SEAT, a secure data trading solution based on our proposed TRUST. Data trading is key to driving the digital economy [57], [58]. SEAT aims to answer the following two questions faced by existing data trading:

(1) Can the trading data be prevented from being copied?
(2) If the answer for the first question is no, can the trading data be prevented from data resale?

Our proposed SEAT argues that SOC is one of feasible methods to prevent from data resale. Roughly speaking, SEAT trades data use right instead of data ownership through SOC.

Fig. 3 shows the system architecture of SEAT that consists of three entities: data seller, data broker, and data buyer. The data seller owning data sells the data use right to the data buyer with the help of the data broker. The workflow of SEAT involves following steps.

❶ The data seller initializes system keys including a public/private key pair (pk, sk) and two pairs of threshold keys (sk_1, sk_2) and $(\mathsf{sk}_1, \mathsf{sk}_2)$. After that, the data seller encrypts data item by item through calling Enc, and transmits ciphertexts and corresponding keys to the data broker.

❷ The data buyer submits data usage requirements to the data broker.

❸ The data broker performs computations as required by the data buyer through calling SOC operations of TRUST, and then returns encrypted result $[\![\mathcal{R}]\!]$ to the data seller.

❹ The data seller calculates $\mathsf{PDec}(\mathsf{sk}_1, [\![\mathcal{R}]\!]) \to [\mathcal{R}]_1$ and sends $\langle [\![\mathcal{R}]\!], [\mathcal{R}]_1, \mathsf{sk}_2 \rangle$ to the data buyer.

❺ The data buyer calls $\mathsf{PDec}(\mathsf{sk}_2, [\![\mathcal{R}]\!]) \to [\mathcal{R}]_2$ and $\mathsf{TDec}([\mathcal{R}]_1, [\mathcal{R}]_2)$ to obtain the final result $\mathcal{R}$.

From the workflow of SEAT, we see that either the data broker or the data buyer fails to learn the origin data of the data seller. In other words, no one of the data broker and the data buyer can resell the origin data from the the data seller. Although the data broker can copy the encrypted origin data, he is only able to perform operations over ciphertexts and lacks decryption capabilities, thereby being incapable to provide any data usage service.

VII. Security Analysis

In this section, we aim to demonstrate that TRUST does not leak any outsourced data to REE and TEE under the assumption of our threat model.

Lemma 1. $\text{Dec}(sk, [\![m]\!] \cdot \text{Enc}(pk, 0)) = \text{Dec}(sk, [\![m]\!])$ *is always true, and* $[\![m]\!] \cdot \text{Enc}(pk, 0) \neq [\![m]\!]$ *is also true.*

Proof. We firstly prove that $\text{Dec}(sk, [\![m]\!] \cdot \text{Enc}(pk, 0)) = \text{Dec}(sk, [\![m]\!])$ is true. According to the additive homomorphism of FastPaiTD, we have

$$[\![m]\!] \cdot \text{Enc}(pk, 0) = [\![m + 0]\!]. \tag{16}$$

And since $m + 0 = m$ is always true $\forall m \in \mathbb{Z}_N$, $\text{Dec}(sk, [\![m]\!] \cdot \text{Enc}(pk, 0)) = m$. Thus, $\text{Dec}(sk, [\![m]\!] \cdot \text{Enc}(pk, 0)) = \text{Dec}(sk, [\![m]\!])$ is always true.

According to Eq. (1), we have

$$\begin{aligned} [\![m]\!] \cdot \text{Enc}(pk, 0) &= [\![m]\!] \cdot (1 + N)^0 \cdot (h^r \bmod N)^N \\ &= [\![m]\!] \cdot (h^r \bmod N)^N, \end{aligned} \tag{17}$$

where $r \leftarrow\$ \{0, 1\}^{4\kappa}$. Obviously, $(h^r \bmod N)^N \neq 1$, so $[\![m]\!] \cdot \text{Enc}(pk, 0) \neq [\![m]\!]$. $\square$

Lemma 2. *When* $m \in (-2^\ell, 2^\ell)$, $r \leftarrow\$ \{0, 1\}^\sigma$, *and* $2^{\sigma-\ell+2}$ *is a negligible function,* $m + r$ *is a chosen-plaintext attack secure one-time key encryption scheme, where* m *and* r *are the plaintext to be encrypted and the key, respectively.*

Proof. Refer to the work [7] for the detail of the proof. $\square$

Theorem 1. *Given* $([\![m_1]\!], [\![m_2]\!])$, $\mathcal{F}_{\text{mul}}$ *securely computes* $[\![m_1 \cdot m_2]\!]$, *and does not leak* m_1, m_2, *or* $m_1 \cdot m_2$ *to REE and TEE.*

Proof. At Step 1, REE only performs operations over $[\![m_1]\!]$ and $[\![m_2]\!]$. As long as FastPaiTD is secure, no m_1, m_2, or $m_1 \cdot m_2$ is leaked to REE.

At Step 2, although TEE can learn $m_1 + r$, he fails to obtain m_1 according to Lemma 2. Thus, we say that m_1 is not leaked to TEE. And since FastPaiTD is secure, TEE cannot obtain m_2 and $m_1 \cdot m_2$.

At Step 3, REE can learn $[\![m_2]\!]^{m_1+r} \cdot [\![-r \cdot m_2]\!] \cdot \text{Enc}(pk, 0)$. Given $[\![m_2]\!]^{m_1+r} \cdot [\![-r \cdot m_2]\!]$, REE is easy to learn m_1 under knowing $[\![m_2]\!]$, r, and $[\![-r \cdot m_2]\!]$. According to the computational indistinguishability experiment [7], REE successfully wins the experiment. Specifically, REE randomly chooses m_0 and m_1 and sends them to a challenger. After that, the challenger $b \leftarrow\$ \{0, 1\}$ and computes and returns $[\![m_2]\!]^{m_b+r} \cdot [\![-r \cdot m_2]\!]$ to REE. In this case, if

$$[\![m_2]\!]^{m_b+r} \cdot [\![-r \cdot m_2]\!] = [\![m_2]\!]^{m_0+r} \cdot [\![-r \cdot m_2]\!],$$

REE ouputs $b = 0$, while

$$[\![m_2]\!]^{m_b+r} \cdot [\![-r \cdot m_2]\!] = [\![m_2]\!]^{m_1+r} \cdot [\![-r \cdot m_2]\!],$$

REE ouputs $b = 1$. In other words, REE always successfully guesses b and win the experiment.

However, as $[\![m]\!] \cdot \text{Enc}(pk, 0) \neq [\![m]\!]$ (See Lemma 2), $[\![m_2]\!]^{m_b+r} \cdot [\![-r \cdot m_2]\!] \cdot \text{Enc}(pk, 0)$ may be equal to $[\![m_2]\!]^{m_0+r} \cdot [\![-r \cdot m_2]\!] \cdot \text{Enc}(pk, 0)$ or $[\![m_2]\!]^{m_1+r} \cdot [\![-r \cdot m_2]\!] \cdot \text{Enc}(pk, 0)$. Thus, REE fails to guess b with a probability greater than $1/2$. In other words, REE cannot win the experiment, so he fails to learn m_1. And since FastPaiTD is secure, m_2 and $m_1 \cdot m_2$ are not leaked to REE. $\square$

Theorem 2. *Given* $([\![m_1]\!], [\![m_2]\!])$, $\mathcal{F}_{\text{cmp}}$ *securely compares* $[\![m_1]\!]$ *and* $[\![m_2]\!]$, *and does not leak* m_1, m_2, *and* μ *to REE and TEE.*

Proof. At Step 1, REE only performs operations over $[\![m_1]\!]$ and $[\![m_2]\!]$. As long as FastPaiTD is secure, m_1, m_2, and μ are leaked to REE.

At Step 2, TEE can learn d and μ_0, but it is easy to demonstrate μ_0 and d do leak m_1, m_2, and μ. Specifically, according to Algorithm 2, we have $\mu = \mu_0 + (1 - 2\mu_0) \cdot \pi$. As FastPaiTD is secure, TEE fails to learn π under knowing $[\![\pi]\!]$. Thus, given μ_0 and $[\![\pi]\!]$, μ is not leaked to TEE.

Now, we adopt the computational indistinguishability experiment to prove that d does not disclose m_1 and m_2. Let TEE acts as an adversary. Then, TEE randomly generates $(m_{1,0}, m_{2,0})$ and $(m_{1,1}, m_{2,1})$ and sends them to a challenger. Mathematically, $(m_{1,0} = m_{2,0})$ and $(m_{1,1} = -2^\ell + 1, m_{2,1} = 2^\ell - 1)$, TEE obtains the highest probability of guessing success. After that, the challenger generates $b, \pi \leftarrow\$ \{0, 1\}$ and $r_1 \leftarrow\$ \{0, 1\}^\sigma, r_2 \leftarrow\$ (\frac{N}{2} - r_1, \frac{N}{2})$, and computes

$$d = \begin{cases} r_1 + r_2, & \text{for } b = 0, \pi = 0 \\ r_1 \cdot 0 + r_2, & \text{for } b = 0, \pi = 1 \\ r_1 \cdot (-2^{\ell+1} + 1) + r_2, & \text{for } b = 1, \pi = 0 \\ r_1 \cdot (2^{\ell+1} - 2) + r_2. & \text{for } b = 1, \pi = 1 \end{cases} \tag{18}$$

The challenger returns d to TEE. Subsequently, TEE guesses $b' = 0$ or $b' = 1$. In this case, the probability of guessing success of TEE can be formulated as

$$\Pr[b' = b | d] \leq \frac{1}{2} + \frac{1}{2} \cdot \frac{1 - (2^{\ell+1} + 1)}{2^\sigma} = \frac{1}{2} + \frac{2^\ell}{2^\sigma}. \tag{19}$$

As $2^\ell \cdot 2^{-\sigma}$ is a negligible function [7], $\Pr[b' = b | d] \leq \frac{1}{2} + \text{negl}(\sigma)$ holds. Thus, we say that The probability of guessing success of TEE is negligible. In other words, TEE fails to learn m_1 and m_2 from d.

At Step 3, REE only obtains an encrypted μ, therefore, μ is not disclosed to REE. $\square$

Theorem 3. *Given* $([\![m_1]\!], [\![m_2]\!])$, $\mathcal{F}_{\text{eql}}$ *securely determines the relationship between* $[\![m_1]\!]$ *and* $[\![m_2]\!]$, *and does not leak* m_1, m_2, *and* μ *to REE and TEE.*

Proof. At Steps 1 and 3, REE either performs operations over $[\![m_1]\!]$ and $[\![m_2]\!]$ or obtains an encrypted μ, therefore, m_1, m_2, and μ are not leaked to REE.

At Step 2, although TEE can learn d_1 and d_2, according to Theorem 2, no m_1 or m_2 is disclosed to TEE. From Algorithm 3, we have $\mu = \mu_0 + (1 - 2\mu_0) \cdot \pi_1 + \mu'_0 + (1 - 2\mu'_0) \cdot \pi_2$. Thus, TEE cannot obtain μ as π_1 and π_2 is unknown. $\square$

Theorem 4. *Given* $[\![m_1]\!]$, $\mathcal{F}_{\text{abs}}$ *securely computes* $[\![|m_1|]\!]$, *and does not leak* $|m_1|$ *and* m_1 *to REE and TEE.*

Proof. At Step 1, REE performs operations over $[\![m_1]\!]$, thus, m_1 is leaked to REE when FastPaiTD is secure.

At Step 2, according Lemma 2 and Theorem 2, TEE fails to learn π and m_1. And since $|m_1| = (1 - 2\mu_0) \cdot m_1 + (4\mu_0 - 2) \cdot \pi \cdot m_1$, $|m_1|$ is not disclosed to TEE due to no π and m_1 being leaked.

TABLE I
COMPARISON OF RUNTIME AND STORAGE COSTS FOR BASIC CRYPTOGRAPHIC PRIMITIVE OPERATIONS (112-BIT SECURITY LEVEL)

Schemes	KGen	Enc	Dec	PDec + TDec[1]	Addition	Scalar-Multiplication	Subtraction	pk	sk	ciphertext
TRUST	**61.779 ms**	**0.346 ms**	**1.617 ms**	10.104 ms	**0.003 ms**	**0.021 ms**	**0.035 ms**	**0.500 KB**	**0.055 KB**	**0.500 KB**
SOCI+ [13]	62.334 ms	0.347 ms	1.619 ms	**9.401 ms**	0.003 ms	0.021 ms	0.035 ms	0.500 KB	0.055 KB	0.500 KB
SOCI [7]	75.776 ms	7.076 ms	7.072 ms	15.005 ms	0.003 ms	0.021 ms	0.035 ms	0.500 KB	0.250 KB	0.500 KB
POCF [24]	74.978 ms	7.053 ms	7.041 ms	14.928 ms	0.003 ms	0.021 ms	0.035 ms	0.500 KB	0.250 KB	0.500 KB

[1] PDec + TDec refers to executing PDec twice (with keys sk_1 and sk_2, respectively) followed by one TDec operation.

TABLE II
COMPARISON OF COMPUTATION AND COMMUNICATION COSTS (112-BIT SECURITY LEVEL)

Protocols	Computation costs (ms)				Communication costs (KB)			
	TRUST	SOCI+[13]	SOCI [7]	POCF [24]	TRUST	SOCI+ [13]	SOCI [7]	POCF [24]
$\mathcal{F}_{mul}$	**11.532**	23.084	62.120	71.271	**0**	1.499	2.499	2.499
$\mathcal{F}_{cmp}$	**11.476**	24.941	46.801	47.791	**0**	1.499	1.499	1.499
$\mathcal{F}_{eql}$	**23.013**	36.224	85.881	253.658	**0**	3.497	3.497	7.998
$\mathcal{F}_{abs}$	**11.024**	20.965	49.115	48.728	**0**	2.498	2.498	2.498
$\mathcal{F}_{trn}$	**22.004**	36.228	70.913	70.322	**0**	4.497	4.497	4.497

Note. The computation costs is the sum of computation time and communication time. Also, we implement $\mathcal{F}_{eql}$, $\mathcal{F}_{abs}$, and $\mathcal{F}_{trn}$ for SOCI+, SOCI, and FOCF by using their own underlying protocols and the idea of TRUST.

At Step 3, REE only learns $[\![R]\!]$, thus, as long as FastPaiTD is secure and Lemma 2 holds, REE fails to obtain $|m_1|$.   □

Theorem 5. *Given* $([\![m_1]\!], [\![m_2]\!], [\![m_3]\!])$, $\mathcal{F}_{trn}$ *securely computes* $[\![m]\!]$, *and does not leak* m_1, m_2, m_3 *and* m *to REE and TEE.*

Proof. At Step 1, REE only performs operations over $[\![m_1]\!]$, $[\![m_2]\!]$ and $[\![m_3]\!]$, therefore, no m_1, m_2, or m_3 is leaked to REE under FastPaiTD being secure.

At Step 2, according Lemma 2 and Theorem 2, TEE fails to learn π_1, π_2 and m_1. And since $m = m_2 + [(\mu_0 + \mu'_0) + (1 - 2\mu_0) \cdot \pi_1 + (1 - 2\mu'_0) \cdot \pi_2] \cdot (m_3 - m_2)$, m is not leaked to TEE. Additionally, as long as FastPaiTD is secure, given $[\![m_2]\!]$ and $[\![m_3 - m_2]\!]$, no m_2 or m_3 is disclosed to TEE.

At Step 3, REE only obtains $[\![R]\!]$, thus, m is not leaked to REE under FastPaiTD being secure and Lemma 2 holding.   □

VIII. EXPERIMENTAL EVALUATIONS

In this section, we extensively evaluate the proposed TRUST and SEAT based on TRUST. Note that all experiments are measured 1000 times and take their average as the experimental result.

A. Experimental Setup

System Configuration. To evaluate the performance of TRUST, we construct the framework using Intel® SGX in hardware mode as a specific implementation of TEE. In detail, we implement TRUST[3] and SEAT in C++ and conduct experiments on a single server. The server runs a 64-bit Ubuntu 20.04 LTS operating system and is equipped with an Intel® Xeon® Silver 4410Y CPU @ 2.00 GHz, along with 128GB of memory, 64GB of which is allocated to the EPC.

---

[3]This work has been open-sourced and can be accessed at: https://github.com/Aptx4869AC/TRUST

Network Configuration. For the comparison schemes SOCI+ [13], SOCI [7], and POCF [24], we deploy two distinct programs on this server, each representing different entities (e.g., the cloud platform and the computation service provider) to simulate communication in a twin-server architecture within a LAN environment, with bandwidth reaching 46.4 Gbps (5.8 GB/s). In contrast, we simulate data transmission between trusted and untrusted environments by implementing secure interfaces (e.g., ECall and OCall functions provided by SGX Enclave). In our testbed, communication costs are defined as the volume of data transmitted between servers. However, in our proposed TRUST architecture, which operates on a single server, inter-server data transmission is avoided. Furthermore, the communication latency between trusted and untrusted regions remains minimal due to intra-server data transfer.

Note that TRUST also adopts the offline-online mechanism proposed by Zhao *et al.* [13] to accelerate the computations. In this work, REE and TEE environments independently generate tuples during the offline phase. Specifically, the encryptions of random numbers and certain constants are precomputed offline, while the online phase handles only remaining operations from the offline phase. For further details, refer to [13].

B. Basic Cryptographic Primitive Operations Evaluations

In this subsection, we evaluate the performance of basic cryptographic primitive operations (including KGen, Enc, Dec, PDec + TDec, and basic homomorphic operations) for TRUST and compare it with SOCI+ [13], SOCI [7] and POCF [24]. The comparison of runtime and storage costs for basic cryptographic primitive operations between different schemes are presented in Table I, where the bit length of N is 2048 to achieve 112-bit security.

From Table I, we see that TRUST outperforms SOCI and POCF in runtime and storage costs. Additionally, TRUST provides a similar performance with SOCI+ for basic cryptographic primitive operations.
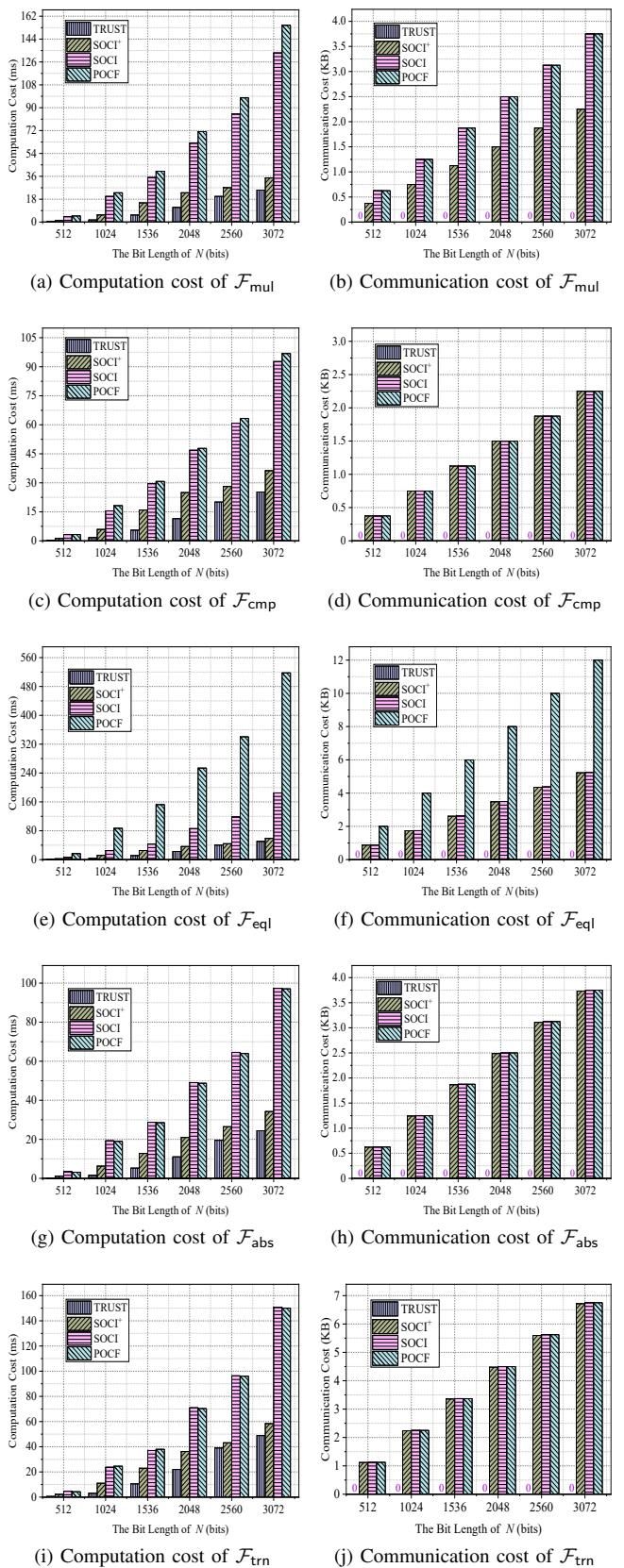
(a) Computation cost of $\mathcal{F}_{\mathsf{mul}}$

(b) Communication cost of $\mathcal{F}_{\mathsf{mul}}$

(c) Computation cost of $\mathcal{F}_{\mathsf{cmp}}$

(d) Communication cost of $\mathcal{F}_{\mathsf{cmp}}$

(e) Computation cost of $\mathcal{F}_{\mathsf{eql}}$

(f) Communication cost of $\mathcal{F}_{\mathsf{eql}}$

(g) Computation cost of $\mathcal{F}_{\mathsf{abs}}$

(h) Communication cost of $\mathcal{F}_{\mathsf{abs}}$

(i) Computation cost of $\mathcal{F}_{\mathsf{trn}}$

(j) Communication cost of $\mathcal{F}_{\mathsf{trn}}$

Fig. 4. Comparison of computation and communication costs for different schemes with a varying bit-length N ($\ell = 32, \sigma = 128$).

C. Protocol Evaluations

Table II presents the computation and communication costs of TRUST, SOCI⁺, SOCI, and POCF at the 112-bit security level. The experimental results show that TRUST outperforms SOCI⁺, SOCI, and POCF in both computation and communication costs. Specifically, experimental results indicate that TRUST improves computation costs by 1.6 - 2.1 times compared to SOCI⁺, which is the state-of-the-art in secure multiplication and secure comparison protocols within the twin-server architecture based on FastPaiTD. One possible explanation is that TRUST designs a more efficient multiplication protocol and presents a more concise system architecture.

Additionally, as shown in Table II, TRUST demonstrates significant advantages in communication costs over SOCI⁺, SOCI, and POCF across all protocols. The reason for this is that our experiments simulate real network communication within a LAN, where frequent serialization and deserialization during data transfer between different servers introduce additional performance overhead. This leads to increased communication latency, higher bandwidth consumption, and potential network congestion, all of which negatively affect overall system throughput and the efficiency of large-scale computations. Thanks to our single-server architecture for TRUST, data transfer occurs within the same machine or with minimal context switching, eliminating these issues.

As depicted in Fig. 4, we give the computation and communication costs of SOC protocols for TRUST, SOCI⁺, SOCI, and POCF with a varying bit-lengths of N, providing an intuitive comparison. Fig. 4 demonstrates that TRUST exhibits significant advantages, particularly for critical protocols such as $\mathcal{F}_{\mathsf{mul}}$ and $\mathcal{F}_{\mathsf{cmp}}$, with both requiring only around 12 ms at $|N| = 2048$. Moreover, the communication costs across all protocols in TRUST remain unaffected as the parameter N increases, consistently showing as 0. Arguably, TRUST outperforms related solutions in both computation and communication costs, suggesting that TRUST would offer significant advantages in high-performance computing, and secure data processing applications.

D. SEAT Evaluations

To evaluate SEAT, we assume the data buyer submits a requirement about training a linear regression (LR) model. The data seller holds the raw data, including the feature matrix $\mathbf{X}$ and the target vector $\vec{y}$. And the data buyer holds the initial model, which includes the weights w, the bias b, and the learning rate η, all specific to LR. Finally, the data broker, which is a single TEE-equipped cloud server, is responsible for securely outsourcing the computation based on the requirements provided by the data buyer.

To show the effectiveness, we compare SEAT with SDTE [42] using AES-GCM (Key: 256-bit; IV: 96-bit; Tag: 128-bit) and the Baseline without any privacy preservation over two open-source datasets: Student Performance[4] and Fish Market[5].
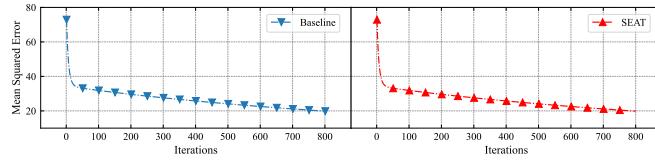
---

[4]This dataset is available at: https://www.kaggle.com/datasets/nikhil7280/student-performance-multiple-linear-regression
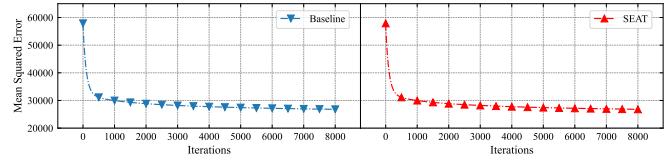
[5]This dataset is available at: https://www.kaggle.com/datasets/vipullrathod/fish-market

TABLE III
CROSS-DATASET FEASIBILITY STUDY AND PERFORMANCE ASSESSMENT OF BASELINE, SDTE, AND SEAT

Dataset	Scale	Settings	Iterations	Baseline			SDTE [42]			SEAT		
				MSE	R^2	MAE	MSE	R^2	MAE	MSE	R^2	MAE
Student Performance	10000×6	learning rate (η): 0.0001 bias (b): 0.0 weights (w): 5 dimensions, all 0 batch size: 64	200 400 600 800	30.072 26.167 22.904 20.140	0.917 0.928 0.937 0.944	4.439 4.135 3.866 3.622	30.072 26.167 22.904 20.140	0.917 0.928 0.937 0.944	4.439 4.135 3.866 3.622	30.072 26.167 22.904 20.140	0.917 0.928 0.937 0.944	4.439 4.135 3.866 3.622
Fish Market	159×7	learning rate (η): 0.0001 bias (b): 0.0 weights (w): 6 dimensions, all 0 batch size: 16	2000 4000 6000 8000	28946.865 27153.971 26344.280 25907.904	0.721 0.738 0.746 0.750	141.262 137.225 134.947 133.422	28946.865 27153.971 26344.280 25907.904	0.721 0.738 0.746 0.750	141.262 137.225 134.947 133.422	28946.865 27153.971 26344.280 25907.904	0.721 0.738 0.746 0.750	141.262 137.225 134.947 133.422

Remark. The training and test sets each comprise half of their total sample count. $\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$, $R^2 = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2}$, $\text{MAE} = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|$ (y_i: True Value, $\hat{y}_i$: Predicted Value, $\bar{y}$: Mean of True Values).

(a) Model training on Student Performance

(b) Model training on Fish Market

Fig. 5. Gradient descent performance of Baseline and SEAT across iterations.

As shown in Table III, SEAT generates the same result as the Baseline and SDTE under three common evaluation metrics. Fig. 5 intuitively shows the performance during training LR between the Baseline and SEAT. In addition, from Fig. 5, we see that SEAT performs equally as the Baseline in terms of training. According to the results of Table III and Fig. 5, we argue that SEAT is feasible.

IX. CONCLUSION

In this work, we proposed TRUST, a toolkit for TEE-assisted secure outsourced computation over integers by seamlessly integrating a (2, 2)-threshold Paillier cryptosystem and TEE. TRUST avoids any collusion attacks from a twin-server architecture of SOC. TRUST not only enriches computational operations, but also improves their performance. Also, we designed a secure data trading solution SEAT based on the proposed TRSUT and confirmed its feasibility. For future work, we will explore TRUST to more complex computation tasks, such as privacy-preserving machine learning inference.

REFERENCES

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

[2] C. Bösch, P. Hartel, W. Jonker, and A. Peter, "A survey of provably secure searchable encryption," *ACM Computing Surveys*, vol. 47, no. 2, pp. 1–51, 2014.

[3] H. Chen, K. Laine, and P. Rindal, "Fast private set intersection from homomorphic encryption," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, 2017, pp. 1243–1255.

[4] D. Rathee, M. Rathee, N. Kumar, N. Chandran, D. Gupta, A. Rastogi, and R. Sharma, "Cryptflow2: Practical 2-party secure inference," in *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, 2020, pp. 325–342.

[5] B. Zhao, W.-N. Chen, F.-F. Wei, X. Liu, Q. Pei, and J. Zhang, "PEGA: A privacy-preserving genetic algorithm for combinatorial optimization," *IEEE Transactions on Cybernetics*, 2024.

[6] G. Platform, "Global platform made simple guide: Trusted execution environment (tee) guide," *Derniere visite*, vol. 12, no. 04, 2013.

[7] B. Zhao, J. Yuan, X. Liu, Y. Wu, H. H. Pang, and R. H. Deng, "SOCI: A toolkit for secure outsourced computation on integers," *IEEE Transactions on Information Forensics and Security*, vol. 17, pp. 3637–3648, 2023.

[8] Y. Lindell, "Secure multiparty computation," *Communications of the ACM*, vol. 64, no. 1, pp. 86–96, 2020.

[9] X. Li, B. Zhao, G. Yang, T. Xiang, J. Weng, and R. H. Deng, "A survey of secure computation using trusted execution environments," *arXiv preprint arXiv:2302.12150*, 2023.

[10] X. Liu, R. H. Deng, P. Wu, and Y. Yang, "Lightning-fast and privacy-preserving outsourced computation in the cloud," *Cybersecurity*, vol. 3, pp. 1–21, 2020.

[11] M. Sabt, M. Achemlal, and A. Bouabdallah, "Trusted execution environment: What it is, and what it is not," in *2015 IEEE Trustcom/BigDataSE/ISPA*, vol. 1. IEEE, 2015, pp. 57–64.

[12] P. Wu, J. Ning, J. Shen, H. Wang, and E.-C. Chang, "Hybrid trust multi-party computation with trusted execution environment." in *Proceedings of the 2022 Network and Distributed System Security Symposium*, 2022.

[13] B. Zhao, W. Deng, X. Li, X. Liu, Q. Pei, and R. H. Deng, "SOCI+: An enhanced toolkit for secure outsourced computation on integers," *IEEE Transactions on Information Forensics and Security*, vol. 19, pp. 5607–5619, 2024.

[14] Y. Yang, X. Huang, X. Liu, H. Cheng, J. Weng, X. Luo, and V. Chang, "A comprehensive survey on secure outsourced computation and its applications," *IEEE Access*, vol. 7, pp. 159 426–159 465, 2019.

[15] Z. Erkin, T. Veugen, T. Toft, and R. L. Lagendijk, "Generating private recommendations efficiently using homomorphic encryption and data packing," *IEEE transactions on information forensics and security*, vol. 7, no. 3, pp. 1053–1066, 2012.

[16] B. K. Samanthula, H. Chun, and W. Jiang, "An efficient and probabilistic secure bit-decomposition," in *Proceedings of the 8th ACM SIGSAC symposium on Information, computer and communications security*, 2013, pp. 541–546.

[17] Y. Elmehdwi, B. K. Samanthula, and W. Jiang, "Secure k-nearest neighbor query over encrypted data in outsourced environments," in *2014 IEEE 30th International Conference on Data Engineering*. IEEE, 2014, pp. 664–675.

[18] T. Veugen, F. Blom, S. J. De Hoogh, and Z. Erkin, "Secure comparison protocols in the semi-honest model," *IEEE Journal of Selected Topics in Signal Processing*, vol. 9, no. 7, pp. 1217–1228, 2015.

[19] P. Paillier, "Public-key cryptosystems based on composite degree residuosity classes," in *International conference on the theory and applications of cryptographic techniques*. Springer, 1999, pp. 223–238.

[20] I. Damgård, M. Geisler, and M. Krøigaard, "Efficient and secure comparison for on-line auctions," in *Information Security and Privacy: 12th Australasian Conference, ACISP 2007, Townsville, Australia, July 2-4, 2007. Proceedings 12.* Springer, 2007, pp. 416–430.

[21] I. Damgård, V. Pastro, N. Smart, and S. Zakarias, "Multiparty computation from somewhat homomorphic encryption," in *Annual Cryptology Conference.* Springer, 2012, pp. 643–662.

[22] C. Gentry, "Fully homomorphic encryption using ideal lattices," in *Proceedings of the forty-first annual ACM symposium on Theory of computing*, 2009, pp. 169–178.

[23] B. Schoenmakers and M. Veeningen, "Universally verifiable multiparty computation from threshold homomorphic cryptosystems," in *Applied Cryptography and Network Security: 13th International Conference, ACNS 2015, New York, NY, USA, June 2-5, 2015, Revised Selected Papers 13.* Springer, 2015, pp. 3–22.

[24] X. Liu, R. H. Deng, W. Ding, R. Lu, and B. Qin, "Privacy-preserving outsourced calculation on floating point numbers," *IEEE Transactions on Information Forensics and Security*, vol. 11, no. 11, pp. 2513–2527, 2016.

[25] A. Lysyanskaya and C. Peikert, "Adaptive security in the threshold setting: From cryptosystems to signature schemes," in *Advances in Cryptology—ASIACRYPT 2001: 7th International Conference on the Theory and Application of Cryptology and Information Security Gold Coast, Australia, December 9–13, 2001 Proceedings 7.* Springer, 2001, pp. 331–350.

[26] A. C.-C. Yao, "How to generate and exchange secrets," in *27th annual symposium on foundations of computer science (Sfcs 1986).* IEEE, 1986, pp. 162–167.

[27] A. Shamir, "How to share a secret," *Communications of the ACM*, vol. 22, no. 11, pp. 612–613, 1979.

[28] M. Blanton, M. J. Atallah, K. B. Frikken, and Q. Malluhi, "Secure and efficient outsourcing of sequence comparisons," in *European Symposium on Research in Computer Security.* Springer, 2012, pp. 505–522.

[29] M. Blanton and M. Aliasgari, "Secure outsourced computation of iris matching," *Journal of Computer Security*, vol. 20, no. 2-3, pp. 259–305, 2012.

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

[31] A. Patra, T. Schneider, A. Suresh, and H. Yalame, "{ABY2. 0}: Improved {Mixed-Protocol} secure {Two-Party} computation," in *30th USENIX Security Symposium (USENIX Security 21)*, 2021, pp. 2165–2182.

[32] W. Zheng, Y. Wu, X. Wu, C. Feng, Y. Sui, X. Luo, and Y. Zhou, "A survey of intel sgx and its applications," *Frontiers of Computer Science*, vol. 15, pp. 1–15, 2021.

[33] V. Costan, "Intel sgx explained," *IACR Cryptol, EPrint Arch*, 2016.

[34] Z. Wang, L. Wang, and H. Yan, "Ma-teecm: Mutual anonymous authentication-based credential migration technology for mobile trusted execution environments," *IEEE Access*, vol. 11, pp. 3680–3690, 2023.

[35] T.-T. Hoang, C. Duran, R. Serrano, M. Sarmiento, K.-D. Nguyen, A. Tsukamoto, K. Suzaki, and C.-K. Pham, "Trusted execution environment hardware by isolated heterogeneous architecture for key scheduling," *IEEE Access*, vol. 10, pp. 46014–46027, 2022.

[36] H. Gao, C. Yue, T. T. A. Dinh, Z. Huang, and B. C. Ooi, "Enabling secure and efficient data analytics pipeline evolution with trusted execution environment," *Proceedings of the VLDB Endowment*, vol. 16, no. 10, pp. 2485–2498, 2023.

[37] H. Sun and H. Lei, "A design and verification methodology for a trustzone trusted execution environment," *IEEE Access*, vol. 8, pp. 33870–33883, 2020.

[38] J. Li, X. Luo, and H. Lei, "Trusthealth: Enhancing ehealth security with blockchain and trusted execution environments," *Electronics*, vol. 13, no. 12, p. 2425, 2024.

[39] Y. Cao, J. Zhang, Y. Zhao, P. Su, and H. Huang, "Srfl: A secure & robust federated learning framework for iot with trusted execution environments," *Expert Systems with Applications*, vol. 239, p. 122410, 2024.

[40] X. Li, G. Yang, T. Xiang, S. Xu, B. Zhao, H. Pang, and R. H. Deng, "Make revocation cheaper: Hardware-based revocable attribute-based encryption," in *2024 IEEE Symposium on Security and Privacy (SP).* IEEE Computer Society, 2024, pp. 100–100.

[41] Z. Liu, C. Hu, R. Li, T. Xiang, X. Li, J. Yu, and H. Xia, "A privacy-preserving outsourcing computing scheme based on secure trusted environment," *IEEE Transactions on Cloud Computing*, vol. 11, no. 3, pp. 2325–2336, 2022.

[42] W. Dai, C. Dai, K.-K. R. Choo, C. Cui, D. Zou, and H. Jin, "Sdte: A secure blockchain-based data trading ecosystem," *IEEE Transactions on Information Forensics and Security*, vol. 15, pp. 725–737, 2019.

[43] Y. Xu, W. Cui, and M. Peinado, "Controlled-channel attacks: Deterministic side channels for untrusted operating systems," in *2015 IEEE Symposium on Security and Privacy.* IEEE, 2015, pp. 640–656.

[44] G. Chen, S. Chen, Y. Xiao, Y. Zhang, Z. Lin, and T. H. Lai, "Sgxpectre: Stealing intel secrets from sgx enclaves via speculative execution," in *2019 IEEE European Symposium on Security and Privacy (EuroS&P).* IEEE, 2019, pp. 142–157.

[45] F. Liu, Y. Yarom, Q. Ge, G. Heiser, and R. B. Lee, "Last-level cache side-channel attacks are practical," in *2015 IEEE symposium on security and privacy.* IEEE, 2015, pp. 605–622.

[46] D. Lie, C. Thekkath, M. Mitchell, P. Lincoln, D. Boneh, J. Mitchell, and M. Horowitz, "Architectural support for copy and tamper resistant software," *Acm Sigplan Notices*, vol. 35, no. 11, pp. 168–177, 2000.

[47] D. Champagne and R. B. Lee, "Scalable architectural support for trusted software," in *HPCA-16 2010 The Sixteenth International Symposium on High-Performance Computer Architecture.* IEEE, 2010, pp. 1–12.

[48] M. Maas, E. Love, E. Stefanov, M. Tiwari, E. Shi, K. Asanovic, J. Kubiatowicz, and D. Song, "Phantom: Practical oblivious computation in a secure processor," in *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, 2013, pp. 311–324.

[49] V. Costan, I. Lebedev, and S. Devadas, "Sanctum: Minimal hardware extensions for strong software isolation," in *25th USENIX Security Symposium (USENIX Security 16)*, 2016, pp. 857–874.

[50] I. Anati, S. Gueron, S. Johnson, and V. Scarlata, "Innovative technology for cpu based attestation and sealing," in *Proceedings of the 2nd international workshop on hardware and architectural support for security and privacy*, vol. 13, no. 7. ACM New York, NY, USA, 2013.

[51] F. McKeen, I. Alexandrovich, A. Berenzon, C. V. Rozas, H. Shafi, V. Shanbhogue, and U. R. Savagaonkar, "Innovative instructions and software model for isolated execution." *Hasp@ isca*, vol. 10, no. 1, 2013.

[52] W. Wang, G. Chen, X. Pan, Y. Zhang, X. Wang, V. Bindschaedler, H. Tang, and C. A. Gunter, "Leaky cauldron on the dark land: Understanding memory side-channel hazards in sgx," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, 2017, pp. 2421–2434.

[53] M. Lipp, M. Schwarz, D. Gruss, T. Prescher, W. Haas, J. Horn, S. Mangard, P. Kocher, D. Genkin, Y. Yarom *et al.*, "Meltdown: Reading kernel memory from user space," *Communications of the ACM*, vol. 63, no. 6, pp. 46–56, 2020.

[54] J. Van Bulck, M. Minkin, O. Weisse, D. Genkin, B. Kasikci, F. Piessens, M. Silberstein, T. F. Wenisch, Y. Yarom, and R. Strackx, "Foreshadow: Extracting the keys to the intel {SGX} kingdom with transient {Out-of-Order} execution," in *27th USENIX Security Symposium (USENIX Security 18)*, 2018, pp. 991–1008.

[55] J. Götzfried, M. Eckert, S. Schinzel, and T. Müller, "Cache attacks on intel sgx," in *Proceedings of the 10th European Workshop on Systems Security*, 2017, pp. 1–6.

[56] A. Moghimi, G. Irazoqui, and T. Eisenbarth, "Cachezoom: How sgx amplifies the power of cache attacks," in *Cryptographic Hardware and Embedded Systems–CHES 2017: 19th International Conference, Taipei, Taiwan, September 25-28, 2017, Proceedings.* Springer, 2017, pp. 69–90.

[57] P. Abla, T. Li, D. He, H. Huang, S. Yu, and Y. Zhang, "Fair and privacy-preserved data trading protocol by exploiting blockchain," *IEEE Transactions on Information Forensics and Security*, vol. 19, pp. 9012–9025, 2024.

[58] B. An, M. Xiao, A. Liu, Y. Xu, X. Zhang, and Q. Li, "Secure crowdsensed data trading based on blockchain," *IEEE Transactions on Mobile Computing*, vol. 22, no. 3, pp. 1763–1778, 2023.

Bowen Zhao (Member, IEEE) received the Ph.D. degree in cyberspace security from the South China University of Technology, China, in 2020. He was a Research Scientist with the School of Computing and Information Systems, Singapore Management University, from 2020 to 2021. He is currently an Associate Professor with the Guangzhou Institute of Technology, Xidian University, Guangzhou, China. His current research interests include privacy-preserving computation and learning and privacy-preserving crowdsensing.

Yulong Shen (Member, IEEE) received the B.S. and M.S. degrees in computer science and the Ph.D. degree in cryptography from Xidian University, Xi'an, China, in 2002, 2005, and 2008, respectively. He is currently a Professor with the School of Computer Science and Technology, Xidian University. He is also an Associate Director of Shaanxi Key Laboratory of Network and System Security and a member of the State Key Laboratory of Integrated Services Networks, Xidian University. His research interests include wireless network security and cloud computing security. He has also served on the technical program committees for several international conferences, including ICEBE, INCoS, CIS, and SOWN.

Jiuhui Li received the B.S. degree in computer science and technology from Guangzhou University, Guangzhou, China, in 2023. He is currently working toward the M.S. degree in Guangzhou Institute of Technology, Xidian University, Guangzhou. His research interests are trusted computing and secure computation.

Peiming Xu received the B.S. and Ph.D. degrees from South China University of Technology, Guangzhou, China, in 2014 and 2019, respectively. He is currently a Researcher with Electric Power Research Institute, China Southern Power Grid Company Ltd., Guangzhou. His research interests mainly focus on cyber security of new power system, privacy computing, and post-quantum cryptography.

Xiaoguo Li received his Ph.D. degree in computer science from Chongqing University, China, in 2019. He was a Research Fellow at Hong Kong Baptist University and Singapore Management University. He is currently an associate professor at Chongqing University, China. His current research interests include trusted computing, secure computation, and public-key cryptography.

Qingqi Pei (Senior Member, IEEE) received the B.S., M.S., and Ph.D. degrees in computer science and cryptography from Xidian University, in 1998, 2005, and 2008, respectively. He is currently a Professor and a member of the State Key Laboratory of Integrated Services Networks, also a Professional Member of ACM, and a Senior Member of the Chinese Institute of Electronics and China Computer Federation. His research interests focus on digital content protection and wireless networks and security.