1-2024

SOCI+: An enhanced toolkit for Secure Outsourced Computation on Integers

Bowen ZHAO

Weiquan DENG

Xiaoguo LI

Ximeng LIU

Qingqi PEI

See next page for additional authors

SOCI⁺: An Enhanced Toolkit for Secure Outsourced Computation on Integers

Bowen Zhao, *Member*, *IEEE*, Weiquan Deng, Xiaoguo Li, Ximeng Liu, *Senior Member, IEEE*, Qingqi Pei, *Senior Member, IEEE*, Robert H. Deng, *Fellow, IEEE*

***Abstract*—Secure outsourced computation is critical for cloud computing to safeguard data confidentiality and ensure data usability. Recently, secure outsourced computation schemes following a twin-server architecture based on partially homomorphic cryptosystems have received increasing attention. The Secure Outsourced Computation on Integers (SOCI) [1] toolkit is the state-of-the-art among these schemes which can perform secure computation on integers without requiring the costly bootstrapping operation as in fully homomorphic encryption; however, SOCI suffers from relatively large computation and communication overhead. In this paper, we propose SOCI⁺ which significantly improves the performance of SOCI. Specifically, SOCI⁺ employs a novel $(2, 2)$-threshold Paillier cryptosystem with fast encryption and decryption as its cryptographic primitive, and supports a suite of efficient secure arithmetic computation on integers protocols, including a secure multiplication protocol (SMUL), a secure comparison protocol (SCMP), a secure sign bit-acquisition protocol (SSBA), and a secure division protocol (SDIV), all based on the $(2, 2)$-threshold Paillier cryptosystem with fast encryption and decryption. In addition, SOCI⁺ incorporates an offline and online computation mechanism to further optimize its performance. We perform rigorous theoretical analysis to prove the correctness and security of SOCI⁺. Compared with SOCI, our experimental evaluation shows that SOCI⁺ is up to 5.4 times more efficient in computation and 40% less in communication overhead.**

***Index Terms*—Secure outsourced computation; Paillier cryptosystem; threshold cryptosystem; homomorphic encryption; secure computing.**

I. Introduction

CLOUD computing provides flexible and convenient services for data outsourced computation, but it is prone to leak outsourced data. Users with limited computation and storage capabilities can outsource their data to the cloud server and perform efficient computations over outsourced data [2]. However, the cloud server may intentionally or unintentionally steal and leak the outsourced data, leading to privacy concerns. At present, numerous data breaches have occured over the world. For example, Facebook exposed a large amount of user data online for a fortnight due to misconfiguration in the cloud [3]. In addition, according to the data breach chronology published by [4], from 2005 to 2022, there have been about 20,000 instances of data breaches in the United States, affecting approximately two billion records.

To prevent data leakages, users can encrypt data before outsourcing [5]. However, performing computations over encrypted data (also known as ciphertext) is challenging, as conventional cryptosystems usually fail to enable computations over ciphertext directly. Secure outsourced computation is an effective manner balancing data security and data usability [2], which enables computations on encrypted data directly. Secure outsourced computation that ensures data security offers a promising computing paradigm for cloud computing, and it can be used in many fields, such as privacy-preserving machine learning training [6] and privacy-preserving evolutionary computation [7].

Homomorphic cryptosystems enable secure outsourced computation as their features achieving addition, multiplication, or both of addition and multiplication over ciphertext. Unfortunately, secure outsourced computation based on homomorphic cryptosystems still suffers from several challenges. Secure outsourced computation solely based on homomorphic cryptosystems is challenging to achieve nonlinear operations (e.g., comparison) [1] and obtain the intermediate result. In certain scenarios, it is necessary to obtain the intermediate result, such as privacy-preserving person re-identification [8]. Homomorphic cryptosystems, such as fully homomorphic encryption that supports addition and multiplication over ciphertext simultaneously, suffer from significant storage costs [1]. Partially homomorphic encryption (PHE) supports addition or multiplication over ciphertext and has a less ciphertext size. To mitigate the limitations imposed by restricted computation types and high storage costs, a combination of PHE and a twin-server architecture has emerged as a promising and increasingly popular paradigm. Despite these advancements, secure outsourced computation solutions [1], [9] based on PHE and the twin-server architecture still bring slightly high computation costs and communication costs.

To tackle the above challenges, in this paper, we propose an enhanced toolkit for secure outsourced computation on integers, named SOCI⁺, which is inspired by SOCI (a toolkit for secure outsourced computation on integers) [1]. Building on the Paillier cryptosystem with fast encryption and decryption [10], we propose a novel $(2, 2)$-threshold Paillier cryptosystem to mitigate computation costs. Subsequently, we redesign all secure computation protocols proposed by SOCI [1]. Additionally, considering the underlying features of secure outsourced computation protocols, which allow a multitude of pre-encryption processes, we divide the computations of these protocols into two phases: offline phase and online phase. In short, the contributions of this paper are three-fold.

- **A novel (2, 2)-threshold Paillier cryptosystem (FastPaiTD).** For the first time, we propose a novel $(2, 2)$-threshold Paillier cryptosystem called FastPaiTD, which is based on the Paillier cryptosystem with fast encryption

Manuscript received x x, x; revised x x, x.

and decryption [10]. FastPaiTD is specially designed to seamlessly adapt to a twin-server architecture.

- **An offline and online mechanism.** To expedite the computations of secure computation protocols, we introduce an offline and online mechanism. Specifically, the encryption of random numbers and some constants in secure computation protocols are computed in advance at the offline phase, while the online phase only perform operations except for these operations performed offline.
- **A suite of secure computation protocols with superior performance.** To support linear operations and nonlinear operations, inspired by SOCI [1], we adopt the proposed FastPaiTD to design a suite secure computation protocols, including a secure multiplication protocol (`SMUL`), a secure comparison protocol (`SCMP`), a secure sign bit-acquisition protocol (`SSBA`), and a secure division protocol (`SDIV`). Compared with SOCI [1], our proposed protocols can improve up to 5.4 times in computation efficiency and saves up to 40% in communication overhead.

The rest of this paper is organized as follows. We briefly review related work in Section II, and show the preliminaries for constructing SOCI⁺ in Section III. The system model and threat model are given in Section IV. In Section V, we firstly present the proposed $(2,2)$-threshold Paillier cryptosystem, along with the introduction of the offline and online mechanism to speed up computations. Subsequently, we elaborate on four secure computation protocols based on the proposed threshold Paillier cryptosystem. The analysis of correctness and security is presented in Section VI, and experimental evaluations are executed in Section VII. Finally, this paper is concluded in Section VIII.

II. Related Work

Secure outsourced computation is a powerful tool that allows users with limited storage and computation capabilities to outsource their data and computations over data to cloud in a secure manner. To enhance the security of data stored in the cloud, numerous solutions for secure outsourced computation have been proposed.

Rahulamathavan *et al.* [11] proposed a privacy-preserving approach for outsourcing support vector machine data classification to the cloud, which is based on a single-server architecture. In their work [11], the operations over encrypted data are rely on Paillier cryptosystem [12] and secure two-party computation. In the work [13], a secure outsourced approach for logistic regression in cloud is proposed, which is also based on Paillier cryptosystem and single-server architecture. Despite the utilization of a powerful cloud server in the aforementioned work, the burden on the client is not truly alleviated due to the execution of some interactions between client and server.

Twin-server architecture emerges as a more practical solution for secure outsourced computation, which significantly reduces the burden of client (or data user). The schemes proposed in [14], [15], [16], [17] exploited twin-server architecture and Paillier cryptosystem to implement secure outsourced computation. Wang *et al.* [18] implemented a secure addition using the twin-server architecture and the ElGamal-based proxy re-encryption with multiplicatively homomorphism. Feng *et al.* [19] leveraged Paillier cryptosystem and twin-server architecture, proposing a secure integer division protocol (SD) and a secure integer square root protocol (SSR). Cui *et al.* [20] proposed a secure division computation protocol (SDC) that leverages random numbers to conceal the real value of dividend and divisor. However, in all of the aforementioned work, a server with a private key is introduced, leading to a single point of security failure. Furthermore, the above work fails to access to intermediate result, making it unsuitable in some settings such as privacy-preserving person re-identification [8].

To mitigate the risk of a single point of security failure and enable access to intermediate result, extensive research has been conducted. The work in [9] and [21] proposed secure outsourced computation solutions by exploiting threshold Paillier cryptosystem. Specifically, in the work [9], the private key of Paillier cryptosystem is split into two parts and distributed to two servers. Consequently, the ciphertext can be decrypted collaboratively by the two servers. In the work [21], the private key is split into multiple partially private keys held by multiple servers, and a ciphertext can be decrypted by a threshold number of servers holding different partially private keys. However, both [9] and [21] introduce a trusted third party to distribute and manage the private keys.

To overcome all the aforementioned weaknesses, a toolkit for secure outsourced computation on integer named SOCI is proposed in [1], which is based on twin-server architecture and threshold Paillier cryptosystem. In addition to supporting additive homomorphism and scalar multiplication homomorphism, SOCI enables secure outsourced computation for four types, including secure multiplication, secure comparison, secure sign bit-acquisition, and secure division. Compared to the protocols in the integer calculation toolkit proposed by [9], the protocols of SOCI are more efficient. However, the computation costs and communication costs of SOCI are still relatively high.

III. Preliminaries

Ma *et al.* [10] proposed a Paillier cryptosystem with fast encryption and decryption. In the rest of this paper, we refer to it as FastPai for its fast encryption and decryption. FastPai is comprised of the following components. $n(\kappa)$ and $l(\kappa)$ refer to the bit length of N and private key, respectively.

*1) **N Generation** (`NGen`):* FastPai calls `NGen` to generate the modulus N for the Paillier cryptosystem. Specifically, `NGen` takes a security parameter κ as input and outputs (N, P, Q, p, q).

The execution of `NGen` proceeds as follows.

(i) Randomly select $\frac{l(\kappa)}{2}$-bit odd primes p, q.
(ii) Randomly select $(\frac{n(\kappa)-l(\kappa)}{2}-1)$-bit odd integers p', q'.
(iii) Compute $P = 2pp' + 1$ and $Q = 2qq' + 1$.
(iv) If p, q, p', q' are not co-prime, or if P or Q is not a prime, then go back to step (i).
(v) Compute $N = PQ$, and output (N, P, Q, p, q).

*2) **Key generation** (`KeyGen`):* `KeyGen` generates a private key sk and a public key pk based on a given parameter κ. It firstly calls `NGen` to obtain (N, P, Q, p, q). Subsequently, it computes $\alpha = pq$ and $\beta = (P-1)(Q-1)/(4pq)$. Next, it computes $h = -y^{2\beta}(\bmod N)$, where y is a number chosen from $\mathbb{Z}_N^*$ uniformly and randomly. Finally, it outputs $pk = (N, h)$ and $sk = \alpha$.

3) ***Encryption*** *(`Enc`):* `Enc` takes a message $m \in \mathbb{Z}_N$ and a public key $pk = (N, h)$ as input and outputs a ciphertext $c \in \mathbb{Z}_{N^2}$, which is defined as follows.

$$c \leftarrow \texttt{Enc}(pk, m) = (1+N)^m \cdot (h^r \bmod N)^N \bmod N^2. \tag{1}$$

In Eq.(1), r is a random number and satisfying $r \leftarrow \{0,1\}^{l(\kappa)}$. In the rest of this paper, we use $[\![x]\!]$ to represent an encrypted x.

4) ***Decryption*** *(`Dec`):* `Dec` takes a ciphertext $c \in \mathbb{Z}_{N^2}$ and a private key $sk = \alpha$ as input and outputs a plaintext message $m \in \mathbb{Z}_N$, which is defined as follows.

$$\begin{aligned} m &\leftarrow \texttt{Dec}(sk, c) \\ &= \Big(\frac{(c^{2\alpha} \mod N^2) - 1}{N} \mod N\Big) \cdot (2\alpha)^{-1} \mod N. \end{aligned} \tag{2}$$

IV. System model and threat model

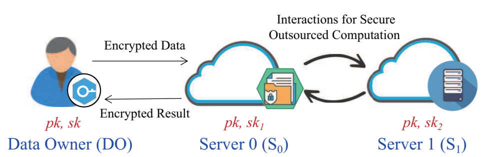


Fig. 1. SOCI+ system architecture

A. System Model

As shown in Fig. 1, SOCI+ consists of a Data Owner (DO) and two non-colluding servers (S_0 and S_1).

- **Data Owner (DO)**. DO is responsible for generating the private key and public key of FastPaiTD, and distributing the public key and the partially private keys sk_1 and sk_2 to S_0 and S_1, respectively. To ensure data security, DO encrypts data with pk and outsources the encrypted data to S_0. Subsequently, DO outsources the computations on ciphertext to S_0 and S_1.
- **Servers**. S_0 is responsible for the storage and the management of the encrypted data uploaded by DO. Additionally, S_0 interacts with S_1 to perform the proposed secure outsourced computation protocols. S_1 only provides computation services and collaborates with S_0 to perform the proposed secure outsourced computation protocols.

B. Threat Model

Following the previous work falling in twin-server architecture [1], [22], [23], [24], SOCI+ comprises three entities, DO, S_0 and S_1. DO is regarded as fully trusted. In SOCI+, there is only one type of adversary, which involves S_0 and S_1, and the adversary attempts to obtain DO's data during execution of secure outsourced computations. Similar to the previous the solutions [25], [26], we assume that S_0 and S_1 are non-colluding. Moreover, we assume S_0 and S_1 both are *curious-but-honest*, i.e., both S_0 and S_1 strictly adhere to the principle of not revealing additional information to each other, except for the necessary information required for performing secure outsourced computations.

It is practical that assuming S_0 and S_1 are non-colluding, when S_0 and S_1 are two different and competitive cloud service providers. The collusion between S_0 and S_1 means that they share the private information (e.g., the partially private keys and the random numbers) to each other. Once S_0 leaks information to S_1, S_1 can leverage the law to punish S_0 and further occupies the market share of S_0, and vise versa. For the interest of business, both S_0 and S_1 will not reveal its private information to each other.

V. SOCI+ Design

A. $(2,2)$-*threshold Paillier cryptosystem (FastPaiTD)*

Inspired by the work [1] and [9], we propose FastPaiTD, a novel $(2,2)$-threshold Paillier cryptosystem, which is based on FastPai [10]. FastPaiTD encompasses the operations of `NGen`, `KeyGen`, `Enc`, and `Dec` from FastPai.

Previous works such as the PaillierTD [27] adopted by SOCI [1] and the PCPD in POCF [9] split the private key (e.g., $sk = \lambda$) into two partially private keys sk_1 and sk_2. In contrast to these methods, we split FastPai's double private key (e.g., $2sk = 2\alpha$) into two partially private keys sk_1 and sk_2, s.t., $sk_1 + sk_2 = 0 \bmod 2\alpha$ and $sk_1 + sk_2 = 1 \bmod N$. In the output of `keygen` in FastPai, N is an odd number that satisfies $\gcd(2\alpha, N) = 1$. To hold $sk_1 + sk_2 = 0 \bmod 2\alpha$ and $sk_1 + sk_2 = 1 \bmod N$ at the same time, we can apply the Chinese remainder theorem [28] to calculate $\delta = sk_1 + sk_2 = (2\alpha) \cdot ((2\alpha)^{-1} \bmod N) \mod (2\alpha \cdot N)$. We can randomly set sk_1 as a σ-bit (e.g., $\sigma = 128$) number, and set $sk_2 = ((2\alpha)^{-1} \bmod N) \cdot (2\alpha) - sk_1 + \eta \cdot 2\alpha \cdot N$, where $\eta \geq 0$. The splitting operation of the private key should be performed in the `keygen` phase.

In addition to the fundamental components of FastPai, FastPaiTD incorporates `PDec` and `TDec` operations. These supplementary operations significantly enhance the flexibility of FastPaiTD, making it a practical tool for secure outsourced computation.

Partial Decryption (`PDec`): This operation enables a party to partially decrypt the ciphertext without revealing the original message. `PDec` takes a ciphertext $c \in \mathbb{Z}_{N^2}$ and a partially private key sk_i ($i \in \{1,2\}$) as input, and outputs a ciphertext $M_i \in \mathbb{Z}_{N^2}$. The partial decryption process is defined as follows.

$$M_i \leftarrow \texttt{PDec}(sk_i, c) = c^{sk_i} \mod N^2. \tag{3}$$

Threshold Decryption (`TDec`): This operation enables two authorized parties to collaboratively decrypt the ciphertext and obtain the original message without knowing the private key sk. `TDec` takes the results of partial decryption M_1 and

Typical Workflow of SOCI

	Server 0 (S_0)		Server 1 (S_1)
1 :	S_0 masks the inputs with with random numbers by computing $masked\_val = \mathcal{M}([\![x]\!], [\![y]\!], r_1, r_2)$; $\mathcal{M}(ciphertexts, r)$ means masking ciphertexts with random numbers; S_0 computes $PDec\_val = PDec(masked\_val)$.	$\xrightarrow{masked\_val,\, PDec\_val}$	
2 :			S_1 performs functions on $masked\_val$ and $PDec\_val$, i.e., performing $operated\_val = \mathcal{F}(masked\_val, PDec\_val)$; $\mathcal{F}(masked\_val, PDec\_val)$ may involve decryption, partial decryption, threshold decryption and so on.
		$\xleftarrow{operated\_val}$	
3 :	S_0 gets the unmasked results in an encrypted form by computing $\mathcal{M}^{-1}(operated\_val, r)$, where $\mathcal{M}^{-1}(operated\_val, r)$ means unmasking the $operated\_val$ with random numbers.		

Fig. 2. Typical Workflow of SOCI

M_2 as input, and outputs a plaintext $m \in \mathbb{Z}_N$. The threshold decryption process is defined as follows.

$$m \leftarrow \texttt{TDec}(M_1, M_2) = \frac{(M_1 \cdot M_2 \mod N^2) - 1}{N} \mod N. \quad (4)$$

Remark. Similar to the PaillierTD [27] adopted by SOCI [1], our $(2,2)$-threshold Paillier cryptosystem (FastPaiTD) supports additive homomorphism and scalar-multiplication homomorphism:

- $\texttt{Enc}(pk, m_1) \cdot \texttt{Enc}(pk, m_2) = \texttt{Enc}(pk, m_1 + m_2)$.
- $\texttt{Enc}(pk, m)^r = \texttt{Enc}(pk, r \cdot m)$, where r is a constant. When $r = N-1$, it holds $\texttt{Enc}(pk, m)^r = \texttt{Enc}(pk, -m)$.

Besides, to enable FastPaiTD supporting the operations on negative integers, we perform a conversion on the negative number m as $m = N - |m|$. Specifically, we take the message spaces $[0, \frac{N}{2}]$ and $[\frac{N}{2}+1, N-1]$ for non-negative numbers and negative numbers, respectively.

B. Offline and online mechanism

As shown in Fig. 2, to hide the real values of inputs, SOCI masks the inputs with random numbers. To securely and correctly obtain the results, the protocols in SOCI involve a large amount of encryption for random numbers. To avoid the expensive encryption overhead during executing the secure outsourced computation protocols, we propose an offline and online mechanism for SOCI⁺ as detailed below.

1) Offline Phase: In contrast to SOCI, we pre-encrypt the random numbers and some constants in the offline phase, such as r_1, r_2, $-r_1 \cdot r_2$, 0 and 1, thereby avoiding to encrypt them in the online phase, which alleviates the computation costs for the secure outsourced computation protocols. Specifically, in the offline phase, we separately establish a tuple for S_0 and S_1, and denote them as $tuple_{S_0}$ and $tuple_{S_1}$, respectively. $tuple_{S_0}$ is consist of r_1, r_2, $[\![r_1]\!]$, $[\![r_2]\!]$, $[\![-r_1 \cdot r_2]\!]$, r_3, r_4, $[\![r_3 + r_4]\!]$, $[\![r_4]\!]$, $[\![0]\!]$ and $[\![1]\!]$. The elements in $tuple_{S_0}$ satisfy the following properties.

- $r_1, r_2 \leftarrow \{0,1\}^\sigma$ (e.g., $\sigma = 128$).
- $r_3 \leftarrow \{0,1\}^\sigma \backslash \{0\}$ (e.g., $\sigma = 128$).
- r_4 is a random number, s.t., $r_4 \leq \frac{N}{2}$ and $r_3 + r_4 > \frac{N}{2}$.

$tuple_{S_1}$ is consist of $[\![0]\!]$ and $[\![1]\!]$. Compared to SOCI, S_0 and S_1 in SOCI⁺ have a simplified process where they only need to extract a ciphertext from their tuples and refresh it atfer usage when it comes to encryption of random numbers and some constants.

However, there is still a number needed to be encrypted in the online phase when we adopt the above mechanism in our SOCI⁺. To speed up the encryption, we can construct a pre-computation table in the offline phase. Moreover, the $\texttt{Enc}$ in FastPai has another equivalent form, as shown below.

$$c \leftarrow \texttt{Enc}(pk, m) = (1 + m \cdot N) \cdot (h^N \mod N^2)^r \mod N^2. \quad (5)$$

The $\texttt{Enc}$ involves a constant $h^N \bmod N^2$, hence we can pre-compute this constant to speed up the $\texttt{Enc}$. Besides, the $\texttt{Enc}$ in FastPai involves a fixed-base modular exponentiation as below.

$$(h^N \mod N^2)^r \mod N^2. \quad (6)$$

Therefore, constructing a pre-computation table can optimize the efficiency of the $\texttt{Enc}$. Ma *et al.* [10] presented the method of constructing a pre-computation table, which is detailed as follows.

To compute $y = a^x$, where a is a fixed base, we can pre-compute the powers of a so that turn the modular exponentiation into modular multiplication since modular multiplication is more efficient than modular exponentiation. Specifically, we let $x = \sum_{i=0}^{\lceil len/b \rceil - 1} x_i \cdot 2^{ib}$, where len is the bit length of x and x_i is the i-th b-bit block. Note that the last block $x_{\lceil len/b \rceil - 1}$ may be less than b bit. We can calculate $y = a^x$ by the following equation.

$$y = a^x = a^{\sum_{i=0}^{\lceil len/b \rceil - 1} x_i \cdot 2^{ib}} = \prod_{i=0}^{\lceil len/b \rceil - 1} (a^{2^{ib}})^{x_i}. \quad (7)$$

Therefore, we can build a two-dimensional pre-computation table with $\lceil len/b \rceil$ rows and 2^b columns, and the index of rows and columns start from 0. The element in row i and column j is $(a^{2^{ib}})^j$, where $i \in [0, \lceil len/b \rceil - 1]$ and $j \in [0, 2^b - 1]$. The table has $\lceil len/b \rceil \cdot 2^b$ elements and every element belongs to $\mathbb{Z}_{N^2}$, hence the table size is $\lceil len/b \rceil \cdot 2^b \cdot (2n)$ bits.

2) Online Phase: In SOCI⁺, S_0 and S_1 construct the $tuple_{S_0}$, $tuple_{S_1}$ and pre-computation table in the offline phase, and utilize the $tuple_{S_0}$, $tuple_{S_1}$ and pre-computation table when perform secure outsourced computation protocols in the online phase.

During the execution of secure outsourced computation protocols, SOCI performs encryption operations on random

Secure Multiplication (`SMUL`): $\texttt{SMUL}(\llbracket x \rrbracket, \llbracket y \rrbracket) \rightarrow \llbracket x \cdot y \rrbracket$

	Server 0 (S_0)		Server 1 (S_1)
	/ `Offline Phase` S_0 constructs a $tuple_{S_0}$, which is consist of r_1, r_2, $\llbracket r_1 \rrbracket$, $\llbracket r_2 \rrbracket$, $\llbracket -r_1 \cdot r_2 \rrbracket$, r_3, r_4, $\llbracket r_3 + r_4 \rrbracket$, $\llbracket r_4 \rrbracket$, $\llbracket 0 \rrbracket$ and $\llbracket 1 \rrbracket$.		S_1 constructs a pre-computation table for speeding up the encryption that cannot pre-compute.
	/ `Online Phase`		
1 :	S_0 inputs $\llbracket x \rrbracket$ and $\llbracket y \rrbracket$, where $x, y \in [-2^l, 2^l]$; S_0 extracts r_1, r_2, $\llbracket r_1 \rrbracket$, $\llbracket r_2 \rrbracket$, and $\llbracket -r_1 \cdot r_2 \rrbracket$ from $tuple_{S_0}$; S_0 refreshes $\llbracket r_1 \rrbracket$, $\llbracket r_2 \rrbracket$ and $\llbracket -r_1 \cdot r_2 \rrbracket$ in $tuple_{S_0}$, using the method mentioned in V-B2; S_0 computes $X = \llbracket x \rrbracket \cdot \llbracket r_1 \rrbracket$ and $Y = \llbracket y \rrbracket \cdot \llbracket r_2 \rrbracket$; S_0 computes $C = X^L \cdot Y$ and $C_1 \leftarrow \texttt{PDec}(sk_1, C)$, where L is a constant satisfying $L \geq 2^{\sigma+2}$.	$\xrightarrow{C, C_1}$	
2 :			S_1 computes $C_2 \leftarrow \texttt{PDec}\ (sk_2, C)$ and $(L \cdot (x + r_1) + y + r_2) \leftarrow \texttt{TDec}(C_1, C_2)$; S_1 sets $(x + r_1) = \lfloor (L \cdot (x + r_1) + y + r_2)/L \rfloor$; S_1 computes $(y + r_2) = (L \cdot (x + r_1) + y + r_2) \mod L$ and $\llbracket (x + r_1) \cdot (y + r_2) \rrbracket \leftarrow \texttt{Enc}(pk, (x + r_1) \cdot (y + r_2))$.
		$\xleftarrow{\llbracket (x + r_1) \cdot (y + r_2) \rrbracket}$	
3 :	S_0 computes $\llbracket -r_2 \cdot x \rrbracket = \llbracket x \rrbracket^{-r_2}$ and $\llbracket -r_1 \cdot y \rrbracket = \llbracket y \rrbracket^{-r_1}$; S_0 gets $\llbracket x \cdot y \rrbracket = \llbracket (x + r_1) \cdot (y + r_2) \rrbracket \cdot \llbracket -r_2 x \rrbracket \cdot \llbracket -r_1 y \rrbracket \cdot \llbracket -r_1 \cdot r_2 \rrbracket$.		

Fig. 3. Secure Multiplication (`SMUL`)

numbers and some constants, whereas SOCI+ extracts the pre-computed encryption values from tuples and hence reduces a large amount of encryption operations. In the proposed FastPaiTD, multiplying a ciphertext $\texttt{Enc}(pk, m)$ by a ciphertext $\texttt{Enc}(pk, 0)$ produces a new ciphertext $\texttt{Enc}(pk, m + 0)$. Although $\texttt{Enc}(pk, m)$ and $\texttt{Enc}(pk, m + 0)$ are not identical, their decrypted results are identical. Consequently, after utilizing the ciphertexts in tuples, S_0 and S_1 refresh the ciphertexts in their tuples by multiplying $\llbracket 0 \rrbracket$. It should be noted that the $\llbracket 0 \rrbracket$ is also included in their tuples. By refreshing the ciphertext, even if the plaintext remains unchanged, the corresponding ciphertext changes. This refresh process creates the illusion that all random numbers and constants are re-encrypted, providing security while reducing computation cost. Moreover, S_0 and S_1 can utilize the pre-computation table to expedite the encryption process when encrypting messages other than the aforementioned random numbers and constants.

C. *Secure Multiplication Protocol (`SMUL`)*

FREED [29] proposed a `SMUL` protocol which is more efficient than the one in SOCI and with the same input and output as SOCI. Same as SOCI, FREED splits the private key of Paillier cryptosystem into two parts and achieves `SMUL` through the interaction between the two servers. In this paper, we re-design the `SMUL` in FREED by incorporating the proposed FastPaiTD and the offline and online computation mechanism.

Given $\llbracket x \rrbracket$ and $\llbracket y \rrbracket$ as input, where $x, y \in [-2^l, 2^l]$, S_0 and S_1 collaboratively compute $\llbracket x \cdot y \rrbracket \leftarrow \texttt{SMUL}(\llbracket x \rrbracket, \llbracket y \rrbracket)$ as output. It should be noted that the input is held by S_0 and only S_0 has the access to the output. When describing `SMUL` in Fig. 3, we omit the input and output for conciseness. As shown in Fig. 3, `SMUL` has two phase, i.e., offline phase and online phase. In the offline phase, S_0 constructs a $tuple_{S_0}$ which is consist of r_1, r_2, $\llbracket r_1 \rrbracket$, $\llbracket r_2 \rrbracket$, $\llbracket -r_1 \cdot r_2 \rrbracket$, r_3, r_4, $\llbracket r_3 + r_4 \rrbracket$, $\llbracket r_4 \rrbracket$, $\llbracket 0 \rrbracket$ and $\llbracket 1 \rrbracket$ (how to choose these random numbers is elaborated in V-B1). Meanwhile, S_1 constructs a pre-computation table for speeding up the encryption that cannot pre-compute. The online phase of `SMUL` comprises three steps as detailed below.

(1) S_0 extracts r_1, r_2, $\llbracket r_1 \rrbracket$, $\llbracket r_2 \rrbracket$, and $\llbracket -r_1 \cdot r_2 \rrbracket$ from $tuple_{S_0}$. Subsequently, S_0 refreshes these ciphertexts in $tuple_{S_0}$, and masks x and y through additive homomorphism. This is accomplished by computing $X = \llbracket x \rrbracket \cdot \llbracket r_1 \rrbracket$ and $Y = \llbracket y \rrbracket \cdot \llbracket r_2 \rrbracket$. S_0 then computes $C = X^L \cdot Y$, where L is a constant satisfying $L \geq 2^{\sigma+2}$. After partially decrypting C to obtain C_1 by calling `PDec`, S_0 sends C and C_1 to S_1.

(2) Upon receiving C and C_1, S_1 calls `PDec` to partially decrypt C, resulting in C_2. Additionally, S_1 obtains $L \cdot (x + r_1) + y + r_2$ by calling `TDec` with C_1 and C_2. Subsequently, S_1 computes $\lfloor (L \cdot (x + r_1) + y + r_2)/L \rfloor$ and $(L \cdot (x + r_1) + y + r_2) \bmod L$ to derive the values of $(x + r_1)$ and $(y + r_2)$, respectively. Finally, S_1 calls `Enc` to encrypt $(x + r_1) \cdot (y + r_2)$ and sends $\llbracket (x + r_1) \cdot (y + r_2) \rrbracket$ to S_0.

(3) As having the knowledge of $\llbracket x \rrbracket$, $\llbracket y \rrbracket$, r_1, r_2 and $\llbracket -r_1 \cdot r_2 \rrbracket$, S_0 can computes $\llbracket x \rrbracket^{-r_2}$ and $\llbracket y \rrbracket^{-r_1}$ to get $\llbracket -r_2 \cdot x \rrbracket$ and $\llbracket -r_1 \cdot y \rrbracket$, respectively. Subsequently, S_0 computes $\llbracket (x + r_1) \cdot (y + r_2) \rrbracket \cdot \llbracket -r_2 x \rrbracket \cdot \llbracket -r_1 y \rrbracket \cdot \llbracket -r_1 \cdot r_2 \rrbracket$ to get $\llbracket x \cdot y \rrbracket$.

D. *Secure Comparison Protocol (`SCMP`)*

In this subsection, we re-design the `SCMP` in SOCI by leveraging the proposed FastPaiTD and the offline and online computation mechanism.

Given $\llbracket x \rrbracket$ and $\llbracket y \rrbracket$ as input, where $x, y \in [-2^l, 2^l]$, S_0 and S_1 collaboratively compute $\llbracket \mu \rrbracket \leftarrow \texttt{SCMP}(\llbracket x \rrbracket, \llbracket y \rrbracket)$ as output. If $\mu = 0$, $x \geq y$, otherwise, $x < y$. It should be noted that the input is held by S_0 and only S_0 has the access to the output. When describing `SCMP` in Fig. 4, we omit the input and output for conciseness. As shown in Fig. 4, the proposed `SCMP` has offline phase and online phase. In the offline phase, S_0 constructs a $tuple_{S_0}$, which is consist of r_1, r_2, $\llbracket r_1 \rrbracket$, $\llbracket r_2 \rrbracket$, $\llbracket -r_1 \cdot r_2 \rrbracket$, r_3, r_4, $\llbracket r_3 + r_4 \rrbracket$, $\llbracket r_4 \rrbracket$, $\llbracket 0 \rrbracket$ and $\llbracket 1 \rrbracket$. Meanwhile, S_1 constructs a $tuple_{S_1}$, which is consist of $\llbracket 0 \rrbracket$ and $\llbracket 1 \rrbracket$. The online phase of `SCMP` consists of three steps as detailed bellow.

Secure Comparison (SCMP): $\texttt{SCMP}([\![x]\!],[\![y]\!]) \rightarrow [\![\mu]\!]$

Server 0 (S_0)

/ Offline Phase

S_0 constructs a $tuple_{S_0}$, which is consist of r_1, r_2, $[\![r_1]\!]$, $[\![r_2]\!]$, $[\![-r_1 \cdot r_2]\!]$, r_3, r_4, $[\![r_3+r_4]\!]$, $[\![r_4]\!]$, $[\![0]\!]$ and $[\![1]\!]$.

/ Online Phase

1 : S_0 inputs $[\![x]\!]$ and $[\![y]\!]$, where $x, y \in [-2^l, 2^l]$;
S_0 extracts r_3, r_4, $[\![r_3+r_4]\!]$ and $[\![r_4]\!]$ from $tuple_{S_0}$, then labels them as r_1, r_2, $[\![r_1+r_2]\!]$ and $[\![r_2]\!]$, respectively;
S_0 refreshes $[\![r_3+r_4]\!]$ and $[\![r_4]\!]$ in $tuple_{S_0}$, using the method mentioned in V-B2;
S_0 randomly chooses a number π, where $\pi \leftarrow \{0,1\}$;
If $\pi = 0$, S_0 calculates $D = ([\![x]\!] \cdot [\![y]\!]^{N-1})^{r_1} \cdot [\![r_1+r_2]\!]$, otherwise, S_0 calculates $D = ([\![y]\!] \cdot [\![x]\!]^{N-1})^{r_1} \cdot [\![r_2]\!]$;
S_0 computes $D_1 \leftarrow \texttt{PDec}(sk_1, D)$.

$\xrightarrow{D, D_1}$

Server 1 (S_1)

S_1 constructs a $tuple_{S_1}$, which is consist of $[\![0]\!]$ and $[\![1]\!]$.

2 : S_1 computes $D_2 \leftarrow \texttt{PDec}(sk_2, D)$ and $d \leftarrow \texttt{TDec}(D_1, D_2)$;
S_1 extracts $[\![0]\!]$ and $[\![1]\!]$ from $tuple_{S_1}$, then refreshes $[\![0]\!]$ and $[\![1]\!]$ in $tuple_{S_1}$, using the method mentioned in V-B2;
If $d > \frac{N}{2}$, S_1 sets $[\![\mu_0]\!] = [\![0]\!]$, otherwise, S_1 sets $[\![\mu_0]\!] = [\![1]\!]$.

$\xleftarrow{[\![\mu_0]\!]}$

3 : If $\pi = 0$, S_0 sets $[\![\mu]\!] = [\![\mu_0]\!]$;
Conversely, if $\pi = 1$, S_0 extracts $[\![1]\!]$ from $tuple_{S_0}$, and then refreshes $[\![1]\!]$ in $tuple_{S_0}$, using the method mentioned in V-B2, finally S_0 sets $[\![\mu]\!] = [\![1]\!] \cdot [\![\mu_0]\!]^{N-1}$.

Fig. 4. Secure Comparison (SCMP)

Secure Sign Bit-Acquisition Protocol (SSBA): $\texttt{SSBA}([\![x]\!]) \rightarrow ([\![s_x]\!], [\![x^*]\!])$

Server 0 (S_0)

/ Offline Phase

S_0 constructs a $tuple_{S_0}$, which is consist of r_1, r_2, $[\![r_1]\!]$, $[\![r_2]\!]$, $[\![-r_1 \cdot r_2]\!]$, r_3, r_4, $[\![r_3+r_4]\!]$, $[\![r_4]\!]$, $[\![0]\!]$ and $[\![1]\!]$.

/ Online Phase

1 : S_0 inputs $[\![x]\!]$, where $x \in [-2^l, 2^l]$;
S_0 extracts $[\![0]\!]$ and $[\![1]\!]$ from $tuple_{S_0}$;
S_0 refreshes $[\![0]\!]$ and $[\![1]\!]$ in $tuple_{S_0}$, using the method mentioned in V-B2.

2 : S_0 and S_1 collaboratively perform $\texttt{SCMP}([\![x]\!], [\![0]\!])$;

$\xrightarrow{\text{Interaction for SCMP}}$

Server 1 (S_1): S_1 and S_0 collaboratively perform $\texttt{SCMP}([\![x]\!], [\![0]\!])$;

$\xleftarrow{\text{Interaction for SCMP}}$

S_0 gets $[\![s_x]\!] \leftarrow \texttt{SCMP}([\![x]\!], [\![0]\!])$.

3 : S_0 computes $[\![1-2s_x]\!] = [\![1]\!] \cdot [\![s_x]\!]^{N-2}$.

4 : S_0 and S_1 collaboratively perform $\texttt{SMUL}([\![1-2s_x]\!], [\![x]\!])$;

$\xrightarrow{\text{Interaction for SMUL}}$

Server 1 (S_1): S_1 and S_0 collaboratively perform $\texttt{SMUL}([\![1-2s_x]\!], [\![x]\!])$;

$\xleftarrow{\text{Interaction for SMUL}}$

S_0 gets $[\![x^*]\!] \leftarrow \texttt{SMUL}([\![1-2s_x]\!], [\![x]\!])$.

Fig. 5. Secure Sign Bit-Acquisition Protocol (SSBA)

(1) S_0 extracts r_3, r_4, $[\![r_3+r_4]\!]$ and $[\![r_4]\!]$ from $tuple_{S_0}$, then labels them as r_1, r_2, $[\![r_1+r_2]\!]$ and $[\![r_2]\!]$, respectively. S_0 then refreshes $[\![r_3+r_4]\!]$ and $[\![r_4]\!]$ in $tuple_{S_0}$. Next, S_0 randomly selects a number π from the set $\{0,1\}$. If $\pi = 0$, S_0 calculates $D = ([\![x]\!] \cdot [\![y]\!]^{N-1})^{r_1} \cdot [\![r_1+r_2]\!]$. If $\pi = 1$, S_0 calculates $D = ([\![y]\!] \cdot [\![x]\!]^{N-1})^{r_1} \cdot [\![r_2]\!]$. Subsequently, S_0 performs a partial decryption of D using PDec to obtain D_1, and sends D and D_1 to S_1.

(2) Upon receiving D and D_1, S_1 performs a partial decryption of D using PDec to obtain D_2. Subsequently, S_1 obtains d by calling TDec with D_1 and D_2. Next, S_1 extracts $[\![0]\!]$ and $[\![1]\!]$ from $tuple_{S_1}$ and refreshes them in $tuple_{S_1}$. If $\pi = 0$, $d = r_1 \cdot (x-y) + (r_1+r_2)$, otherwise, $d = r_1 \cdot (y-x) + r_2$. If $d > \frac{N}{2}$, S_1 sets $[\![\mu_0]\!] = [\![0]\!]$. Conversely, if $d \le \frac{N}{2}$, S_1 sets $[\![\mu_0]\!] = [\![1]\!]$. Finally, S_1 sends $[\![\mu_0]\!]$ to S_0.

(3) If $\pi = 0$, S_0 sets $[\![\mu]\!] = [\![\mu_0]\!]$. Conversely, if $\pi = 1$, S_0 extracts $[\![1]\!]$ from $tuple_{S_0}$, and refreshes it in $tuple_{S_0}$, then sets $[\![\mu]\!] = [\![1]\!] \cdot [\![\mu_0]\!]^{N-1}$.

E. Secure Sign Bit-Acquisition Protocol (SSBA)

In this subsection, we re-design the SSBA in SOCI by leveraging the proposed FastPaiTD and the offline and online computation mechanism.

Given $[\![x]\!]$ as input, where $x \in [-2^l, 2^l]$, S_0 and S_1 collaboratively compute $([\![s_x]\!], [\![x^*]\!]) \leftarrow \texttt{SSBA}([\![x]\!])$ as output. s_x is the sign bit of x, and x^* represents the magnitude of x. If $x \ge 0$, $s_x = 0$ and $x^* = x$, otherwise, $s_x = 1$ and $x^* = -x$. It should be noted that the input is held by S_0 and only S_0 has the access to the output. When describing SSBA in Fig. 5, we omit the input and output for conciseness. As shown in Fig. 5, SSBA consists of offline phase and online phase. In the offline phase, S_0 constructs a $tuple_{S_0}$, which is consist of r_1, r_2, $[\![r_1]\!]$, $[\![r_2]\!]$, $[\![-r_1 \cdot r_2]\!]$, r_3, r_4, $[\![r_3+r_4]\!]$, $[\![r_4]\!]$, $[\![0]\!]$ and $[\![1]\!]$. The online phase of SSBA consists of four steps as detailed below.

(1) S_0 extracts $[\![0]\!]$ and $[\![1]\!]$ from $tuple_{S_0}$ and refreshes them in $tuple_{S_0}$.

(2) S_0 and S_1 collaboratively perform $[\![s_x]\!] \leftarrow$

Secure Division Protocol (SDIV): $\texttt{SDIV}([\![x]\!], [\![y]\!]) \rightarrow ([\![q]\!], [\![e]\!])$

Server 0 (S_0) | Server 1 (S_1)

/ Offline Phase

S_0 constructs a $tuple_{S_0}$, which is consist of r_1, r_2, $[\![r_1]\!]$, $[\![r_2]\!]$, $[\![-r_1 \cdot r_2]\!]$, r_3, r_4, $[\![r_3 + r_4]\!]$, $[\![r_4]\!]$, $[\![0]\!]$ and $[\![1]\!]$.

/ Online Phase

1 : S_0 inputs $[\![x]\!]$ and $[\![y]\!]$, where $x \in [0, 2^l]$ and $y \in (0, 2^l]$;
S_0 extracts $[\![0]\!]$ and $[\![1]\!]$ from $tuple_{S_0}$;
S_0 refreshes $[\![0]\!]$ and $[\![1]\!]$ in $tuple_{S_0}$, using the method mentioned in V-B2; S_0 sets $q = [\![0]\!]$.

for i **in** $\{l, l-1, \ldots, 1, 0\}$ **do**

2 : S_0 computes $[\![c]\!] = [\![y]\!]^{2^i}$.
3 : S_0 and S_1 collaboratively perform $\texttt{SCMP}([\![x]\!], [\![c]\!])$;

Interaction for SCMP $\longrightarrow$

S_1 and S_0 collaboratively perform $\texttt{SCMP}([\![x]\!], [\![c]\!])$;

$\longleftarrow$ Interaction for SCMP

S_0 gets $[\![\mu]\!] \leftarrow \texttt{SCMP}([\![x]\!], [\![c]\!])$.
4 : S_0 computes $[\![\mu']\!] = [\![1]\!] \cdot [\![\mu]\!]^{N-1}$ and $[\![q]\!] = [\![q]\!] \cdot [\![\mu']\!]^{2^i}$.
S_0 and S_1 collaboratively perform $\texttt{SMUL}([\![\mu']\!], [\![c]\!])$;

Interaction for SMUL $\longrightarrow$

S_1 and S_0 collaboratively perform $\texttt{SMUL}([\![\mu']\!], [\![c]\!])$;

$\longleftarrow$ Interaction for SMUL

S_0 obtains $[\![m]\!] \leftarrow \texttt{SMUL}([\![\mu']\!], [\![c]\!])$.
5 : S_0 computes $[\![x]\!] = [\![x]\!] \cdot [\![m]\!]^{N-1}$.
endfor

6 : S_0 gets $[\![e]\!] = [\![x]\!]$.

Fig. 6. Secure Division Protocol (SDIV)

$\texttt{SCMP}([\![x]\!], [\![0]\!])$. If $x \geq 0$, $s_x = 0$, otherwise, $s_x = 1$.

(3) S_0 computes $[\![1 - 2s_x]\!] = [\![1]\!] \cdot [\![s_x]\!]^{N-2}$.

(4) Finally, S_0 and S_1 collaboratively perform $[\![x^*]\!] \leftarrow \texttt{SMUL}([\![1 - 2s_x]\!], [\![x]\!])$. Obviously, $[\![x^*]\!] = (1 - 2s_x) \cdot x$. Furthermore, if $x \geq 0$, $x^* = x$, otherwise, $x^* = -x$.

F. Secure Division Protocol (SDIV)

In this subsection, we re-design the SDIV in SOCI by leveraging the proposed FastPaiTD and the offline and online computation mechanism.

Given $[\![x]\!]$ and $[\![y]\!]$ as input, where $x \in [0, 2^l]$ and $y \in (0, 2^l]$, S_0 and S_1 collaboratively compute $([\![q]\!], [\![e]\!]) \leftarrow \texttt{SDIV}([\![x]\!], [\![y]\!])$ as output. In the output of SDIV, q represents the quotient of division and e represents the remainder of division, such that $x = q \cdot y + e$. It should be noted that the input is held by S_0 and only S_0 has the access to the output. When describing SDIV in Fig. 6, we omit the input and output for conciseness. As shown in Fig. 6, SDIV consists of offline phase and online phase. In the offline phase, S_0 constructs a $tuple_{S_0}$, which is consist of r_1, r_2, $[\![r_1]\!]$, $[\![r_2]\!]$, $[\![-r_1 \cdot r_2]\!]$, r_3, r_4, $[\![r_3 + r_4]\!]$, $[\![r_4]\!]$, $[\![0]\!]$ and $[\![1]\!]$. The online phase of SDIV consists of seven steps as detailed below.

(1) S_0 extracts $[\![0]\!]$ and $[\![1]\!]$ from $tuple_{S_0}$ and refreshes them in $tuple_{S_0}$. Subsequently, S_0 sets $[\![q]\!] = [\![0]\!]$ and $i = l$.

(2) S_0 obtains $[\![2^i \cdot y]\!]$ by computing $[\![c]\!] = [\![y]\!]^{2^i}$, where $i \in \{l, l-1, ..., 1, 0\}$.

(3) S_0 and S_1 collaboratively perform $[\![\mu]\!] \leftarrow \texttt{SCMP}([\![x]\!], [\![c]\!])$. If $x \geq 2^i \cdot y$, $\mu = 0$, otherwise, $\mu = 1$.

(4) S_0 computes $[\![\mu']\!] = [\![1]\!] \cdot [\![\mu]\!]^{N-1}$ and $[\![q]\!] = [\![q]\!] \cdot [\![\mu']\!]^{2^i}$. It is important to note that $\mu' = 1 - \mu$. Besides, $q = q + 2^i$ if $\mu' = 1$, otherwise, q remains unchanged.

(5) S_0 and S_1 collaboratively perform $[\![m]\!] \leftarrow \texttt{SMUL}([\![\mu']\!], [\![c]\!])$, where $m = \mu' \cdot 2^i \cdot y$. If $\mu' = 1$ (i.e., $x \geq 2^i \cdot y$), m = $2^i \cdot y$, otherwise, $m = 0$.

(6) S_0 obtains $[\![x]\!] = [\![x - m]\!]$ by computing $[\![x]\!] = [\![x]\!] \cdot [\![m]\!]^{N-1}$. If $m = 2^i \cdot y$ (i.e., $x \geq 2^i \cdot y$), $x = x - 2^i \cdot y$, otherwise, x remains unchanged. Next, sets $i = i - 1$. Steps (2)-(6) should be repeated until $i < 0$.

(7) S_0 obtain the remainder $[\![e]\!]$ by setting $[\![e]\!] = [\![x]\!]$.

VI. Correctness and Security Analysis

A. Correctness Analysis

In this subsection, we provide the rigorous correctness proofs for the proposed FastPaiTD and the secure outsourced computation protocols in SOCI+.

Theorem 1. *In FastPaiTD, given two ciphertexts $M_1 \leftarrow \texttt{PDec}(sk_1, c)$ and $M_2 \leftarrow \texttt{PDec}(sk_2, c)$, $\texttt{TDec}(M_1, M_2)$ can correctly recover the plaintext m.*

Proof. Before proceeding with the proof, we introduce two important equations. The work [10] has proven that their FastPai satisfies the following equation.

$$c^{2\alpha} \bmod N^2 = (1 + N)^{2\alpha m}. \tag{8}$$

In Eq. (8), c is a ciphertext, m is the corresponding plaintext, and α is the private key. Besides, it is widely recognized that the following equation holds for any integer m.

$$(1 + N)^m \bmod N^2 = (1 + m \cdot N) \bmod N^2. \tag{9}$$

Now we demonstrate the correctness of the proposed FastPaiTD. To simplify the proof process, we adopt the notations $L(x)$ for $\frac{x-1}{N}$ and $(2\alpha)^{-1}$ for $(2\alpha)^{-1} \bmod N$. The proof process is as follow.

By substituting M_1 and M_2 with $c^{sk_1} \bmod N^2$ and $c^{sk_2} \bmod N^2$, respectively, we can easily calculate the following equation.

$$\texttt{TDec}(M_1, M_2) = L(c^{(2\alpha)^{-1} \cdot (2\alpha)} \cdot c^{\eta \cdot 2\alpha \cdot N} \bmod N^2) \bmod N. \tag{10}$$

Next, we can obtain the following equation by utilizing Eqs. (8) and (9).

$$\texttt{TDec}(M_1, M_2) = L((1+N)^{2\alpha\cdot(2\alpha)^{-1}m} \mod N^2) \mod N. \quad (11)$$

Afterward, we can adopt Eq.(9) and expand $L(x)$ to get the following equation.

$$\texttt{TDec}(M_1, M_2) = \frac{(1+mN)-1}{N} \mod N. \quad (12)$$

Finally, we can conclude that $\texttt{TDec}(M_1, M_2) = m \mod N$. □

Theorem 2. *Given $[\![x]\!]$ and $[\![y]\!]$ as input, where $x, y \in [-2^l, 2^l]$, the proposed* SMUL *protocol correctly outputs $[\![x \cdot y]\!]$.*

Proof. We assume that the operations in offline phase have been executed correctly. Therefore, our focus now lies on the correctness of online phase.

In step 1, S_0 computes $X = [\![x]\!] \cdot [\![r_1]\!] = [\![x + r_1]\!]$ and $Y = [\![y]\!] \cdot [\![r_2]\!] = [\![y + r_2]\!]$. Subsequently, S_0 computes $C = X^L \cdot Y = [\![L \cdot (x + r_1) + y + r_2]\!]$ and $C_1 \leftarrow \texttt{PDec}(sk_1, C)$.

In step 2, upon receiving C and C_1, S_1 performs $C_2 \leftarrow \texttt{PDec}(sk_2, C)$. Subsequently, S_1 performs $L \cdot (x + r_1) + y + r_2 \leftarrow \texttt{TDec}(C_1, C_2)$, which yields $x + r_1$ and $y + r_2$.

In step 3, upon receiving $[\![(x + r_1) \cdot (y + r_2)]\!]$ from S_1, S_0 computes $[\![-r_2 x]\!] = [\![x]\!]^{-r_2}$ and $[\![-r_1 y]\!] = [\![y]\!]^{-r_1}$. Subsequently, S_0 computes $[\![(x+r_1)\cdot(y+r_2)]\!]\cdot[\![-r_2 x]\!]\cdot[\![-r_1 y]\!]\cdot[\![-r_1 r_2]\!] = [\![(x+r_1)\cdot(y+r_2)-r_2 x-r_1 y-r_1 r_2]\!] = [\![x\cdot y]\!]$. □

Theorem 3. *Given $[\![x]\!]$ and $[\![y]\!]$ as input, where $x, y \in [-2^l, 2^l]$, the proposed* SCMP *protocol correctly outputs $[\![\mu]\!]$. If $\mu = 0$, $x \geq y$, otherwise, $x < y$.*

Proof. According to Fig. 4, there are two possible values for D that can be obtained, i.e., $r_1\cdot(x-y+1)+r_2$ and $r_1\cdot(y-x)+r_2$. In Fig. 4, r_1 and r_2 are derived from r_3 and r_4 in $tuple_{S_0}$, hence we have $r_1 \leftarrow \{0,1\}^\sigma \backslash \{0\}$, $r_2 \leq \frac{N}{2}$ and $r_1 + r_2 > \frac{N}{2}$. Since $r_1 \leftarrow \{0,1\}^\sigma \backslash \{0\}$, $r_2 \leq \frac{N}{2}$, $r_1 + r_2 > \frac{N}{2}$ and $x, y \in [-2^l, 2^l]$, we can easily obtain $0 < r_1 \cdot (x - y + 1) + r_2 < N$ and $0 < r_1 \cdot (y - x) + r_2 < N$.

When $0 < r_1 \cdot (x - y + 1) + r_2 \leq \frac{N}{2}$, it implies that $x - y + 1 \leq 0$. Consequently, we have $x < y$, and SCMP outputs $[\![1]\!]$. When $\frac{N}{2} < r_1 \cdot (x - y + 1) + r_2 \leq N$, it implies that $x - y + 1 \geq 1$. In this case, we have $x \geq y$, and SCMP outputs $[\![0]\!]$.

When $0 < r_1 \cdot (y - x) + r_2 \leq \frac{N}{2}$, it implies that $y - x \leq 0$. In this case, we have $x \geq y$, and SCMP outputs $[\![0]\!]$. When $\frac{N}{2} < r_1 \cdot (y - x) + r_2 < N$, it implies that $y - x \geq 1$. In this case, we have $x < y$, and SCMP outputs $[\![1]\!]$.

Therefore, the proposed SCMP correctly compares x and y. □

Theorem 4. *Given $[\![x]\!]$ as input, where $x \in [-2^l, 2^l]$, the proposed* SSBA *protocol correctly outputs $[\![s_x]\!]$ and $[\![x^*]\!]$. s_x is the sign bit of x, and x^* represents the magnitude of x. If $x \geq 0$, $s_x = 0$ and $x^* = x$, otherwise, $s_x = 1$ and $x^* = -x$.*

Proof. Given $[\![x]\!]$ and $[\![0]\!]$ as input, where $x \in [-2^l, 2^l]$, according to Theorem 3, we can easily observe that the $\texttt{SCMP}([\![x]\!], [\![0]\!])$ outputs $[\![1]\!]$ when $x \in [-2^l, 0)$ and outputs $[\![0]\!]$ when $x \in [0, 2^l]$. Therefore, if $x \in [0, 2^l]$, $s_x = 0$, otherwise, $s_x = 1$.

According to Theorem 2, we can correctly obtain $[\![(1-2\cdot s_x)\cdot x]\!] \leftarrow \texttt{SMUL}([\![1-2\cdot s_x]\!], [\![x]\!])$. When $x \geq 0$, $(1-2\cdot s_x)\cdot x = x$ as $(1 - 2 \cdot s_x) = 1$. When $x < 0$, $(1 - 2 \cdot s_x) \cdot x = -x$ as $(1 - 2 \cdot s_x) = -1$.

Therefore, the proposed SSBA correctly outputs $[\![s_x]\!]$ and $[\![x^*]\!]$. □

Theorem 5. *Given $[\![x]\!]$ and $[\![y]\!]$ as input, where $x \in [0, 2^l]$ and $y \in (0, 2^l]$, the proposed* SDIV *protocol correctly outputs $[\![q]\!]$ and $[\![e]\!]$. q is the quotient of division, and e is the remainder of division, i.e., $x = q \cdot y + e$.*

Proof. It is widely recognized that any quotient q satisfying $q \in [0, 2^l]$ and $0 \leq x - q \cdot y < y$ can be represented as $\sum_0^{i=l} q_i \cdot 2^i$, where $q_i \in \{0, 1\}$. As shown in the loop of Fig. 6, for any $i \in \{l, l-1, ..., 1, 0\}$, if $x \geq 2^i \cdot y$, then we have $\mu' = 1$, $q_i = 1 \cdot 2^i$ and $x = x - 1 \cdot 2^i \cdot y$, otherwise, $\mu' = 0$, $q_i = 0 \cdot 2^i$ and $x = x - 0 \cdot 2^i \cdot y$. We observe that $\sum_0^{i=l} q_i \cdot 2^i = \sum_0^{i=l} \mu' \cdot 2^i$, where $\mu' \in \{0, 1\}$. Therefore, any $q \in [0, 2^l]$ can be represented by $\sum_0^{i=l} \mu' \cdot 2^i$. Since $\sum_0^{i=l} q_i \cdot 2^i = q$, $e = x - y \cdot \sum_0^{i=l} q_i \cdot 2^i$. Therefore, the proposed SDIV protocol correctly outputs $[\![q]\!]$ and $[\![e]\!]$. □

B. Security Analysis

Liu *et al.* [9] has proven the semantic security of their Paillier cryptosystem with partial decryption (PCPD) in POCF. In this paper, we adopt the same method used in POCF [9] to prove the security of FastPaiTD. In SOCI+, we assume that S_0 is not colluding with S_1. Following the approach of POCF [9], we define the semantic security model for FastPaiTD.

Definition 1. *Let ζ = (*NGen*,* keygen*,* Enc*,* Dec*,* PDec*,* TDec*) be a public key cryptosystem that supports partial decryption (*PDec*) and threshold decryption (*TDec*). Assuming a polynomial adversary $\mathcal{A}$, if $\mathcal{A}$ has negligible advantage in the challenger-adversary game, then ζ is semantically secure. The challenger-adversary game is defined as follows.*

- *The challenger obtains the public key pk and private key sk of ζ by calling* keygen*. Subsequently, the challenger splits sk into sk_1 and sk_2, and sends pk and one of sk_1 and sk_2 to $\mathcal{A}$.*
- *$\mathcal{A}$ randomly selects two plaintexts m_0 and m_1 with equal bit-length, and sends them to the challenger through a secure communication channel.*
- *The challenger flips a coin to randomly choose a bit $b \in \{0, 1\}$, then adopts* Enc *to encrypt m_b into ciphertext c and sends c to $\mathcal{A}$.*
- *The $\mathcal{A}$ outputs a bit b'. If $b' = b$, $\mathcal{A}$ succeeds, otherwise, $\mathcal{A}$ fails.*

The advantage of $\mathcal{A}$ in this game is defined as $\mathsf{Adv}_\xi(\kappa) = \left|\Pr[b = b'] - \frac{1}{2}\right|$, where κ is a secure parameter.

We now formally adopt the method presented in [9] to prove the semantic security of our novel $(2, 2)$-threshold Paillier cryptosystem (FastPaiTD).

Theorem 6. *Assuming FastPai is semantically secure, the FastPaiTD in V-A is also semantically secure.*

Proof. The semantic security of FastPai has been proven in [10]. As same as [9], we assume a probabilistic polynomial-time adversary $\mathcal{A}$ who breaks the semantic security of FastPaiTD with an advantage at most ϵ. We also construct a simulator $\mathcal{S}$ with the same time complexity as $\mathcal{A}$. Then, $\mathcal{S}$, $\mathcal{A}$ and the challenger perform the following operations.

- The challenger obtains the public key $pk = (N, h)$ of FastPai.
- $\mathcal{S}$ randomly chooses sk_1, where $sk_1 \in [0, N(N-1)]$.
- $\mathcal{A}$ receives pk and sk_1 from $\mathcal{S}$, then randomly chooses two plaintexts m_0 and m_1 with same bit-length, and sends m_0 and m_1 to $\mathcal{S}$.
- After receiving m_0 and m_1 from $\mathcal{A}$, $\mathcal{S}$ sends them to the challenger of FastPai.
- The challenger randomly chooses a bit b, and encrypts m_b into a ciphertext c by calling `Enc`. Subsequently, the challenger sends c to $\mathcal{S}$.
- $\mathcal{S}$ sends c to $\mathcal{A}$.
- $\mathcal{A}$ finally outputs a bit b' as the guess of $\mathcal{S}$ and sends it to $\mathcal{S}$.

From the view of $\mathcal{A}$, excepting sk_1, the distributions of pk and challenger's ciphertexts are as same as in the real semantic security experiment. According to V-A, the real $sk_1 \in [0, 2\alpha N]$. Therefore, given $X \in [0, 2\alpha N]$ and $Y \in [0, N(N-1)]$, we can calculate that X and Y have at most $\frac{2\alpha}{N-1}$ statistical distance. Hence, $\mathcal{S}$ breaks the semantic security of FastPaiTD with advantage at least $\epsilon - \frac{2\alpha}{N-1}$. □

The offline and online mechanism consists of two parts. The first part involves computing the encryption of random numbers and some constants. The second part involves constructing a pre-computation table to speed up `Enc`. We now prove that the offline and online mechanism is secure, meaning that it does not reveal any information about the plaintext.

Theorem 7. *The offline and online mechanism in SOCI+ does not leak any information when performing secure outsourced computations.*

Proof. The security of the first part is proven as follow. Each time S_0 and S_1 extract a ciphertext $[\![m]\!]$ from $tuple_{S_0}$ and $tuple_{S_1}$, respectively, they immediately adopt the $[\![0]\!]$ in their tuple to compute $[\![m']\!] = [\![m]\!] \cdot [\![0]\!]$. Subsequently, they replace $[\![m]\!]$ in their tuple with $[\![m']\!]$. Although $[\![m']\!]$ and $[\![m]\!]$ are the encrypted values of the same number, $[\![m']\!]$ is not identical as $[\![m]\!]$. Therefore, after refreshing a ciphertext, it appears as if S_0 and S_1 encrypt a message each time. Consequently, the first part of the offline and online mechanism does not disclose any plaintext information.

In the second part, the pre-computation table is constructed from the public key, allowing anyone to create it. Its sole function is to accelerate the `Enc`. Therefore, the second part of the offline and online mechanism does not disclose any plaintext information. □

SOCI adopts the simulation paradigm [30], also known as the real/ideal model, to prove the security of its `SMUL` protocol. Therefore, we employ the same method to prove the security of `SMUL` in SOCI+. Similar to SOCI, SOCI+ assumes that S_0 and S_1 are semi-honest and non-colluding, which means that S_0 and S_1 may act as adversaries. We use $\mathcal{A}_{S_0}$ and $\mathcal{A}_{S_1}$ to denote S_0 and S_1 as polynomial-time adversaries, respectively. We now adopt the same method in SOCI to prove the security of `SMUL` in SOCI+.

Theorem 8. *Given ciphertexts $[\![x]\!]$ and $[\![y]\!]$, where $x, y \in [-2^l, 2^l]$, in the case of semi-honest attackers $\mathcal{A}_{S_0}$ and $\mathcal{A}_{S_1}$, the proposed `SMUL` protocol in SOCI+ is able to compute $[\![x \cdot y]\!]$ securely.*

Proof. To simulate S_0 and S_1, we construct independent simulators $\mathcal{S}_{S_0}$ and $\mathcal{S}_{S_1}$, respectively.

$\mathcal{S}_{S_0}$ simulates the view of $\mathcal{A}_{S_0}$ as follows.

- $\mathcal{S}_{S_0}$ takes $[\![x]\!]$, $[\![y]\!]$, $[\![(x + r_1) \cdot (y + r_2)]\!]$ as input, and randomly chooses $\tilde{x}$ and $\tilde{y}$, where $\tilde{x}, \tilde{y} \in [-2^l, 2^l]$. Besides, $\mathcal{S}_{S_0}$ also randomly chooses $\tilde{r_1}$, $\tilde{r_2}$, and $\tilde{sk_1}$, where $\tilde{r_1}, \tilde{r_2}, \tilde{sk_1} \leftarrow \{0,1\}^{\sigma}$. Subsequently, $\mathcal{S}_{S_0}$ randomly chooses $\tilde{L}$ with (σ+2) bits.
- $\mathcal{S}_{S_0}$ obtains $[\![\tilde{x}]\!]$, $[\![\tilde{y}]\!]$, $\tilde{X}$, $\tilde{Y}$, $[\![-\tilde{r_2}\tilde{x}]\!]$, $[\![-\tilde{r_1}\tilde{y}]\!]$ and $[\![-\tilde{r_1}\tilde{r_2}]\!]$ by calling `Enc` to encrypt $\tilde{x}$, $\tilde{y}$, $\tilde{x}+\tilde{r_1}$, $\tilde{y}+\tilde{r_2}$, $-\tilde{r_2}\tilde{x}$, $-\tilde{r_1}\tilde{y}$ and $-\tilde{r_1}\tilde{r_2}$, respectively. Subsequently, $\mathcal{S}_{S_0}$ computes $\tilde{C} = \tilde{X}^{\tilde{L}} \cdot \tilde{Y}$, and gets $\tilde{C_1} \leftarrow \texttt{PDec}(\tilde{sk_1}, \tilde{C})$.
- $\mathcal{S}_{S_0}$ computes $[\![\tilde{x} \cdot \tilde{y}]\!] = [\![(x + r_1) \cdot (y + r_2)]\!] \cdot [\![-\tilde{r_2}\tilde{x}]\!] \cdot [\![-\tilde{r_1}\tilde{y}]\!] \cdot [\![-\tilde{r_1}\tilde{r_2}]\!]$.
- Finally, $\mathcal{S}_{S_0}$ outputs the simulation of $\mathcal{A}_{S_0}$'s entire view, consisting of $[\![\tilde{x}]\!]$, $[\![\tilde{y}]\!]$, $\tilde{X}$, $\tilde{Y}$, $\tilde{C}$, $\tilde{C_1}$, $[\![-\tilde{r_2}\tilde{x}]\!]$, $[\![-\tilde{r_1}\tilde{y}]\!]$, $[\![-\tilde{r_1}\tilde{r_2}]\!]$ and $[\![\tilde{x} \cdot \tilde{y}]\!]$.

We conclude that $\mathcal{S}_{S_0}$'s view in the ideal world and $\mathcal{A}_{S_0}$'s view in the real word are computationally indistinguishable since the Paillier cryptosystem in [10] is semantically secure.

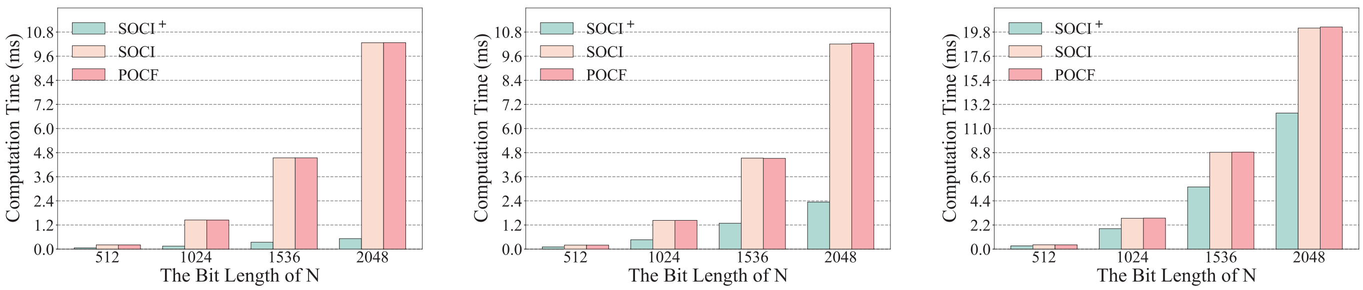
$\mathcal{S}_{S_1}$ simulates the view of $\mathcal{A}_{S_1}$ as follows.

- $\mathcal{S}_{S_1}$ takes $[\![x + r_1]\!]^L \cdot [\![y + r_2]\!]$, $([\![x + r_1]\!]^L \cdot [\![y + r_2]\!])^{sk_1}$ as input, and randomly chooses $\tilde{x}$ and $\tilde{y}$, where $\tilde{x}, \tilde{y} \in [-2^l, 2^l]$. Subsequently, $\mathcal{S}_{S_1}$ also randomly chooses $\tilde{r_1}$, $\tilde{r_2}$, $\tilde{sk_1}$ and $\tilde{sk_2}$, where $\tilde{r_1}, \tilde{r_2}, \tilde{sk_1}, \tilde{sk_2} \leftarrow \{0,1\}^{\sigma}$. Next, $\mathcal{S}_{S_1}$ randomly chooses $\tilde{L}$ with (σ+2) bits.
- $\mathcal{S}_{S_1}$ obtains $\tilde{X}$, $\tilde{Y}$ and $[\![(\tilde{x}+\tilde{r_1}) \cdot (\tilde{y}+\tilde{r_2})]\!]$ by calling `Enc` to encrypt $\tilde{x}+\tilde{r_1}$, $\tilde{y}+\tilde{r_2}$ and $(\tilde{x}+\tilde{r_1}) \cdot (\tilde{y}+\tilde{r_2})$. Moreover, $\mathcal{S}_{S_1}$ computes $\tilde{C} = \tilde{X}^{\tilde{L}} \cdot \tilde{Y}$, $\tilde{C_1} \leftarrow \texttt{PDec}(\tilde{sk_1}, \tilde{C})$ and $\tilde{C_2} \leftarrow \texttt{PDec}(\tilde{sk_2}, \tilde{C})$.
- Finally, $\mathcal{S}_{S_1}$ outputs the simulation of $\mathcal{A}_{S_1}$'s entire view, consisting of $\tilde{C}$, $\tilde{C_1}$, $\tilde{C_2}$, $\tilde{x}+\tilde{r_1}$, $\tilde{y}+\tilde{r_2}$, $(\tilde{x}+\tilde{r_1}) \cdot (\tilde{y}+\tilde{r_2})$ and $[\![(\tilde{x}+\tilde{r_1}) \cdot (\tilde{y}+\tilde{r_2})]\!]$.

We conclude that $\mathcal{S}_{S_1}$'s view in the ideal world and $\mathcal{A}_{S_1}$'s view in the real word are computationally indistinguishable, since the Paillier cryptosystem is semantically secure and SOCI [1] has proven that the one-time key encryption scheme $x + r$ is able to securely hide x. □

Theorem 9. *Given ciphertexts $[\![x]\!]$ and $[\![y]\!]$, where $x, y \in [-2^l, 2^l]$, in the case of semi-honest attackers $\mathcal{A}_{S_0}$ and $\mathcal{A}_{S_1}$, the proposed `SCMP` protocol in SOCI+ is able to compare x and y securely.*

Proof. The proposed `SCMP` in SOCI+ roots in the `SCMP` in SOCI [1]. Since SOCI [1] has proven the security of its `SCMP` and our building blocks (i.e., FastPaiTD and the offline and



(a) Comparison of Enc in Different Schemes

(b) Comparison of Dec in Different Schemes

(c) Comparison of PDec (using sk_2) in Different Schemes

Fig. 7. Comparison of Different Threshold Paillier Cryptosystems with a Varying Bit-Length of N

online mechanism) are secure, the proposed `SCMP` protocol is able to compare x and y in a secure manner. □

Theorem 10. *Given ciphertext $[\![x]\!]$, where $x \in [-2^l, 2^l]$, in the case of semi-honest attackers $\mathcal{A}_{\mathcal{S}_0}$ and $\mathcal{A}_{\mathcal{S}_1}$, the proposed `SSBA` protocol in SOCI+ is able to obtain $[\![s_x]\!]$ and $[\![x^*]\!]$ securely.*

Proof. The proposed `SSBA` is constructed by calling `SCMP` and `SMUL`. Since these protocols and the building blocks (i.e., FastPaiTD and the offline and online mechanism) are secure, the proposed `SSBA` protocol is able to obtain $[\![s_x]\!]$ and $[\![x^*]\!]$ in a secure manner. □

Theorem 11. *Given ciphertexts $[\![x]\!]$ and $[\![y]\!]$, where $x \in [0, 2^l]$ and $y \in (0, 2^l]$, in the case of semi-honest attackers $\mathcal{A}_{\mathcal{S}_0}$ and $\mathcal{A}_{\mathcal{S}_1}$, the proposed `SDIV` protocol in SOCI+ is able to obtain $[\![q]\!]$ and $[\![e]\!]$ securely.*

Proof. The proposed `SDIV` is constructed by calling `SCMP` and `SMUL`. Since these protocols and the building blocks (i.e., FastPaiTD and the offline and online mechanism) are secure, the proposed `SDIV` protocol is able to obtain $[\![q]\!]$ and $[\![e]\!]$ in a secure manner. □

VII. Experimental Evaluation

SOCI+ has protocols similar to the privacy preserving integer calculation protocols in POCF [9]. In the rest of this paper, for simplicity, we denote the privacy preserving integer calculation protocols in POCF as POCF. To evaluate the computation and communication costs, we implement SOCI+ (which is open source[1]), SOCI, and POCF using gmpy2-2.1.0a1 in Python 3.6.8 on two identical servers (CPU: AMD EPYC 7402 24-Core Processor; Memory: 128 GB). In our experiments, we set $l = 32$ and $\sigma = 128$. Specially, we set $l = 10$ when evaluating the `SDIV` protocol. When constructing a pre-computation table to speed up `Enc` in SOCI+, we set $b = 5$, and the parameter len is equal to the bit-length of sk. It should be noted that POCF fails to support `SSBA` and `SDIV`. Therefore, we adopt the system architecture of POCF to implement `SSBA` and `SDIV` proposed by [31], and regard them as components of POCF. We repeat all experiments for 500 times with a single thread and take the average as experimental results. In the rest of this paper, we adopt $|N|$ to denote the bit length of N. When presenting the experimental results in the form of table, we highlight all the best results in bold.

A. Performance of Different Threshold Paillier Cryptosystem

In this subsection, we evaluate the performance of different threshold Paillier cryptosystems, which form the foundation of SOCI+, SOCI and POCF.

In our experiments, the size of private key in SOCI+ is 448 bits when $|N| = 2048$, thus the size of private key in SOCI+ is about 0.055 KB. A smaller private key in SOCI+ leads to faster `PDec`. Figs. 7(a) and 7(b) compare the computation costs of `Enc` and `Dec` with different bit-length of N among SOCI+, SOCI and POCF. The results show that SOCI+ has fastest encryption and decryption.

In SOCI+ and SOCI, we can compute $M_1 \leftarrow \texttt{PDec}(c, sk_1)$ and $M_2 \leftarrow \texttt{PDec}(c, sk_2)$ to partially decrypt a ciphertext c, respectively. After obtaining M_1 and M_2, the corresponding plaintext m can be obtained by computing $\texttt{TDec}(M_1, M_2)$. In [9], POCF has the operations of `PDec1` and `PDec2`, where `PDec1` is equivalent to $\texttt{PDec}(c, sk_1)$, and `PDec2` integrates $\texttt{PDec}(c, sk_2)$ and $\texttt{TDec}(M_1, M_2)$. For convenience, when describing POCF, we adopt `PDec` and `TDec` instead of `PDec1` and `PDec2`. For SOCI+, SOCI, and POCF, sk_1 is set to be the same number with σ bits. For SOCI and POCF, we set $sk_2 = \lambda \cdot (\lambda^{-1} \bmod N) - sk_1$, where λ is the private key of SOCI and POCF, and we set $sk_2 = ((2\alpha)^{-1} \bmod N) \cdot (2\alpha) - sk_1$ for SOCI+. Table I presents the computation costs comparison of `PDec` and `TDec` among SOCI+, SOCI and POCF. The computation costs of `PDec` (using sk_1) and `TDec` are almost the same in all schemes, but SOCI+ achives best performance in `PDec` (using sk_2). Fig. 7(c) presents an intuitive comparison of `PDec` (using sk_2), demonstrating that SOCI+ outperforms the other two schemes and improves the computation costs by approximately 1.6 times compared to SOCI when $|N| = 2048$.

B. Evaluations for Secure Outsourced Computation Protocols

In this subsection, we compare the computation costs, communication costs, and running time of `SMUL`, `SCMP`, `SSBA` and `SDIV` to evaluate their performance. We define the running time as the sum of computation time and communication time, and assuming a bandwidth with 100 Mbps.

[1]https://github.com/W-Q-Deng/SOCI-plus

TABLE I
COMPARISON OF BASICALLY CRYPTOGRAPHIC OPERATIONS AND STORAGE COSTS ASSUMING THE BIT-LENGTH OF N IS 2048 (112-BIT SECURITY)

Scheme	Keygen	Enc	Dec	PDec (sk_1) [1]	PDec (sk_2) [1]	TDec	Addition	Scalar-mul[2]	Subtraction	PK[3]	SK[3]	Ciphertext[3]
SOCI+	**148.036** ms	**0.522** ms	**2.340** ms	**0.704** ms	**12.412** ms	**0.007** ms	**0.006** ms	**0.066** ms	**0.058** ms	0.500 KB	**0.055** KB	**0.500** KB
SOCI	1036.621 ms	10.269 ms	10.207 ms	0.705 ms	20.168 ms	0.007 ms	0.006 ms	0.066 ms	0.058 ms	**0.250** KB	0.250 KB	0.500 KB
POCF	1043.951 ms	10.273 ms	10.246 ms	0.706 ms	20.264 ms	0.007 ms	0.006 ms	0.066 ms	0.058 ms	0.250 KB	0.250 KB	0.500 KB

[1] PDec (sk_1) and PDec (sk_2) stand for performing PDec with sk_1 and sk_2, respectively.
[2] Scalar-mul stands for scalar-multiplication.
[3] PK, SK and Ciphertext stand for size of public key, size of private key and size of ciphertext, respectively.

TABLE II
COMPARISON OF COMPUTATION COSTS AND COMMUNICATION COSTS ASSUMING 112-BIT SECURITY

Algorithm	Computation Costs			Communication Costs			Running Time		
	SOCI+	SOCI	POCF	SOCI+	SOCI	POCF	SOCI+	SOCI	POCF
`SMUL`	**15.698** ms	84.098 ms	92.104 ms	**1.498** KB	2.498 KB	2.498 KB	**15.821** ms	84.303 ms	92.309 ms
`SCMP`	**19.037** ms	52.342 ms	53.015 ms	**1.498** KB	1.499 KB	1.499 KB	**19.160** ms	52.465 ms	53.138 ms
`SSBA`	**34.773** ms	157.054 ms	155.460 ms	**2.997** KB	3.996 KB	3.997 KB	**35.019** ms	157.381 ms	155.787 ms
`SDIV`	**382.624** ms	1524.189 ms	8524.647 ms	**32.965** KB	43.959 KB	244.314 KB	**385.324** ms	1527.790 ms	8544.661 ms

TABLE III
THEORETICAL COMPARISON OF COMMUNICATION COSTS

Scheme	`SMUL`	`SCMP`	`SSBA`	`SDIV`
SOCI+	**3**$\|N^2\|$ bits	**3**$\|N^2\|$ bits	**6**$\|N^2\|$ bits	**6**$(l+1)\|N^2\|$ bits
SOCI	5$\|N^2\|$ bits	3$\|N^2\|$ bits	8$\|N^2\|$ bits	8$(l+1)\|N^2\|$ bits
POCF	5$\|N^2\|$ bits	3$\|N^2\|$ bits	8$\|N^2\|$ bits	$(3l^2+13l+59)\|N^2\|$ bits

Table II presents the comparison of computation costs, communication costs, and running time among SOCI+, SOCI and POCF. The experimental results demonstrate that SOCI+ outperforms the other two schemes. Specifically, experimental results indicate that SOCI+ improves the computation costs by $2.7-5.4$ times compared to SOCI. Figs. 8(a), 8(b), 8(c) and 8(d) present the computation costs comparison of `SMUL`, `SCMP`, `SSBA` and `SDIV`, respectively. The results indicate that SOCI+ outperforms both SOCI and POCF in terms of computation costs, and the advantage of SOCI+ increase with $|N|$.

Table II demonstrates that SOCI+ generally reduces communication costs by approximately $25\%-40\%$ compared to SOCI, except for `SCMP`. The experimental results for communication costs of `SMUL`, `SCMP`, `SSBA` and `SDIV` are presented at Figs. 9(a), 9(b), 9(c) and 9(d), respectively. While the three shcemes has almost the same communication costs for `SCMP`, SOCI+ exhibits significant advantage in other protocols when N is large. To better understand the differences in communication costs among the three schemes, we present a theoretical analysis of communication costs for SOCI+, SOCI and POCF in Table III. The experimental results align with theoretical analysis of communication costs.

The running time of the proposed protocols is affected by computation power and bandwidth. As previously mentioned, the running time is the sum of computation time and communication time, and we assume the bandwidth is 100 Mbps. The experimental results for running time are presented in the right-hand side of Table II. The results indicate that SOCI+ improves $2.7-5.3$ times in terms of running time compared to SOCI. Figs. 10(a), 10(b), 10(c) and 10(d) present the experimental results for running time with a varying $|N|$. The results show that SOCI+ outperforms both SOCI and POCF in terms of running time with different $|N|$.

VIII. CONCLUSION

In this paper, we proposed SOCI+, an enhanced toolkit for secure outsourced computation on integers. Specifically, we designed a novel $(2,2)$-threshold Paillier cryptosystem (FastPaiTD) falling in the twin-server architecture based on the scheme of Ma *et al.* [10] (FastPai). Additionally, we proposed an offline and online mechanism for SOCI+. Our FastPaiTD and offline and online mechanism significantly improve the performance of secure outsourced computation protocols. SOCI+ strictly outperforms the state-of-the-art in terms of computation costs and communication costs and is correct and secure. In the future work, we will upgrade SOCI+ to support floating point arithmetic and more types of secure outsourced computations.

REFERENCES

[1] 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, 2022.

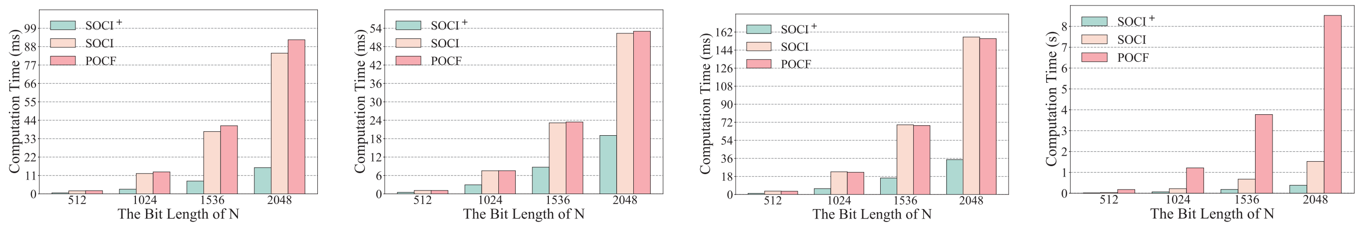
[2] Z. Shan, K. Ren, M. Blanton, and C. Wang, "Practical secure computation outsourcing: A survey," *ACM Computing Surveys (CSUR)*, vol. 51, no. 2, pp. 1–40, 2018.

[3] P. Muncaster. (2019) Data leak exposes 267 million facebook users. [Online]. Available: https://www.infosecurity-magazine.com/news/data-leak-exposes-267-million/

[4] 2023. [Online]. Available: https://privacyrights.org/data-breaches

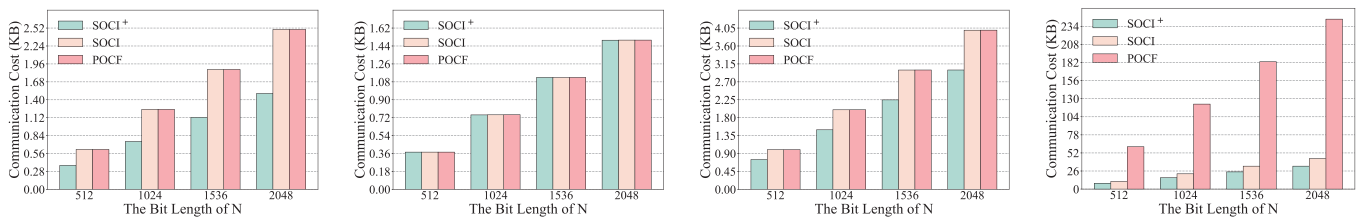
[5] P. Li, J. Li, Z. Huang, C.-Z. Gao, W.-B. Chen, and K. Chen, "Privacy-preserving outsourced classification in cloud computing," *Cluster Computing*, vol. 21, pp. 277–286, 2018.

[6] C. Wang, A. Wang, J. Xu, Q. Wang, and F. Zhou, "Outsourced privacy-preserving decision tree classification service over encrypted data," *Journal of Information Security and Applications*, vol. 53, p. 102517, 2020.



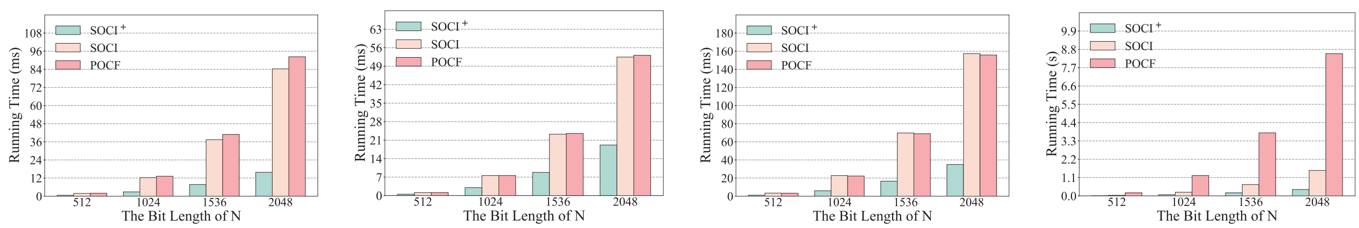
(a) Computation Costs of SMUL in Different Schemes (b) Computation Costs of SCMP in Different Schemes (c) Computation Costs of SSBA in Different Schemes (d) Computation Costs of SDIV in Different Schemes

Fig. 8. Computation Costs Comparison of Different Schemes with a Varying Bit-Length of N



(a) Communication Costs of SMUL in Different Schemes (b) Communication Costs of SCMP in Different Schemes (c) Communication Costs of SSBA in Different Schemes (d) Communication Costs of SDIV in Different Schemes

Fig. 9. Communication Costs Comparison of Different Schemes with a Varying Bit-Length of N



(a) Running Time of SMUL in Different Schemes (b) Running Time of SCMP in Different Schemes (c) Running Time of SSBA in Different Schemes (d) Running Time of SDIV in Different Schemes

Fig. 10. Running Time Comparison of Different Schemes with a Varying Bit-Length of N, assuming the Bandwidth is 100 Mbps

[7] B. Zhao, W.-N. Chen, F.-F. Wei, X. Liu, Q. Pei, and J. Zhang, "Evolution as a service: A privacy-preserving genetic algorithm for combinatorial optimization," *arXiv preprint arXiv:2205.13948*, 2022.

[8] B. Zhao, Y. Li, X. Liu, X. Li, H. H. Pang, and R. H. Deng, "Identifiable, but not visible: A privacy-preserving person reidentification scheme," *IEEE Transactions on Reliability*, 2023.

[9] 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.

[10] H. Ma, S. Han, and H. Lei, "Optimized paillier's cryptosystem with fast encryption and decryption," in *Annual Computer Security Applications Conference*, 2021, pp. 106–118.

[11] Y. Rahulamathavan, R. C.-W. Phan, S. Veluru, K. Cumanan, and M. Rajarajan, "Privacy-preserving multi-class support vector machine for outsourcing the data classification in cloud," *IEEE Transactions on Dependable and Secure Computing*, vol. 11, no. 5, pp. 467–479, 2013.

[12] P. Paillier, "Public-key cryptosystems based on composite degree residuosity classes," in *Advances in Cryptology—EUROCRYPT'99: International Conference on the Theory and Application of Cryptographic Techniques Prague, Czech Republic, May 2–6, 1999 Proceedings 18*. Springer, 1999, pp. 223–238.

[13] X. D. Zhu, H. Li, and F. H. Li, "Privacy-preserving logistic regression outsourcing in cloud computing," *International Journal of Grid and Utility Computing*, vol. 4, no. 2-3, pp. 144–150, 2013.

[14] 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.

[15] 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.

[16] H. Chun, Y. Elmehdwi, F. Li, P. Bhattacharya, and W. Jiang, "Outsourceable two-party privacy-preserving biometric authentication," in *Proceedings of the 9th ACM symposium on Information, computer and communications security*, 2014, pp. 401–412.

[17] B. K. Samanthula, W. Jiang, and E. Bertino, "Privacy-preserving complex query evaluation over semantically secure encrypted data," in *Computer Security-ESORICS 2014: 19th European Symposium on Research in Computer Security, Wroclaw, Poland, September 7-11, 2014. Proceedings, Part I 19*. Springer, 2014, pp. 400–418.

[18] B. Wang, M. Li, S. S. Chow, and H. Li, "A tale of two clouds: Computing on data encrypted under multiple keys," in *2014 IEEE Conference on Communications and Network Security*. IEEE, 2014, pp. 337–345.

[19] J. Feng, L. T. Yang, Q. Zhu, and K.-K. R. Choo, "Privacy-preserving tensor decomposition over encrypted data in a federated cloud environment," *IEEE Transactions on Dependable and Secure Computing*, vol. 17, no. 4, pp. 857–868, 2018.

[20] N. Cui, X. Yang, B. Wang, J. Li, and G. Wang, "Svknn: Efficient secure and verifiable k-nearest neighbor query on the cloud platform," in *2020 IEEE 36th International Conference on Data Engineering (ICDE)*. IEEE, 2020, pp. 253–264.

[21] X. Liu, K.-K. R. Choo, R. H. Deng, R. Lu, and J. Weng, "Efficient and privacy-preserving outsourced calculation of rational numbers," *IEEE Transactions on Dependable and Secure Computing*, vol. 15, no. 1, pp.

27–39, 2016.
[22] V. Nikolaenko, U. Weinsberg, S. Ioannidis, M. Joye, D. Boneh, and N. Taft, “Privacy-preserving ridge regression on hundreds of millions of records,” in *2013 IEEE symposium on security and privacy*. IEEE, 2013, pp. 334–348.
[23] P. Mohassel and Y. Zhang, “Secureml: A system for scalable privacy-preserving machine learning,” in *2017 IEEE symposium on security and privacy (SP)*. IEEE, 2017, pp. 19–38.
[24] J. Chen, L. Liu, R. Chen, W. Peng, and X. Huang, “Secrec: a privacy-preserving method for the context-aware recommendation system,” *IEEE Transactions on Dependable and Secure Computing*, vol. 19, no. 5, pp. 3168–3182, 2021.
[25] B. Xie, T. Xiang, X. Liao, and J. Wu, “Achieving privacy-preserving online diagnosis with outsourced svm in internet of medical things environment,” *IEEE Transactions on Dependable and Secure Computing*, vol. 19, no. 6, pp. 4113–4126, 2021.
[26] C. Hu, C. Zhang, D. Lei, T. Wu, X. Liu, and L. Zhu, “Achieving privacy-preserving and verifiable support vector machine training in the cloud,” *IEEE Transactions on Information Forensics and Security*, 2023.
[27] 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.
[28] D. Pei, A. Salomaa, and C. Ding, *Chinese remainder theorem: applications in computing, coding, cryptography*. World Scientific, 1996.
[29] B. Zhao, Y. Li, X. Liu, H. H. Pang, and R. H. Deng, “Freed: An efficient privacy-preserving solution for person re-identification,” in *2022 IEEE Conference on Dependable and Secure Computing (DSC)*. IEEE, 2022, pp. 1–8.
[30] S. Micali, O. Goldreich, and A. Wigderson, “How to play any mental game,” in *Proceedings of the Nineteenth ACM Symp. on Theory of Computing, STOC*. ACM New York, NY, USA, 1987, pp. 218–229.
[31] X. Liu, R. H. Deng, K.-K. R. Choo, and J. Weng, “An efficient privacy-preserving outsourced calculation toolkit with multiple keys,” *IEEE Transactions on Information Forensics and Security*, vol. 11, no. 11, pp. 2401–2414, 2016.

Bowen Zhao (Member, IEEE) received the Ph.D. degree in cyberspace security from South China University of Technology, Guangzhou, 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 Guangzhou Institute of Technology, Xidian University, Guangzhou. His current research interests include privacy-preserving computation and learning and privacy-preserving crowdsensing.

Weiquan Deng received the B.S. degree in Information Security from Guangxi University, Nanning, China, in 2023.

He is currently working toward the M.S. degree in Guangzhou Institute of Technology, Xidian University, Guangzhou. His research interest is privacy-preserving computation.

Xiaoguo Li received his Ph.D. degree in computer science from Chongqing University, China, in 2019.

He worked at Hong Kong Baptist University as a Postdoctoral Research Fellow from 2019-2021. He is currently a Research Fellow at Singapre Management University, Singapore. His current research interests include trusted computing, secure computation, and public-key cryptography.

Ximeng Liu (Senior Member, IEEE) received the B.Sc. degree in electronic engineering and the Ph.D. degree in cryptography from Xidian University, Xi’an, China, in 2010 and 2015, respectively.

He is currently a Full Professor with the College of Computer Science and Data Science, Fuzhou University. He was a Research Fellow with Peng Cheng Laboratory, Shenzhen, China. He has published more than 200 papers on the topics of cloud security and Big Data security including papers in IEEE TOC, IEEE TII, IEEE TDSC, IEEE TSC, IEEE IoT Journal, etc. His research interests include cloud security, applied cryptography and Big Data security.

Dr. Liu received “Minjiang Scholars” Distinguished Professor Award, “Qishan Scholars” at Fuzhou University and ACM SIGSAC China Rising Star Award (2018).

Qingqi Pei (Senior Member, IEEE) received the B.S., M.S., and Ph.D. degrees in computer science and cryptography from Xidian University, Xi’an, China, in 1998, 2005, and 2008, respectively.

He is currently a Professor and Member of the State Key Laboratory of Integrated Services Networks, Xidian University. His research interests include digital content protection and wireless network and security.

Dr. Pei is a Professional Member of the Association for Computing Machinery and a Senior Member of the Chinese Institute of Electronics and the China Computer Federation.

Robert H. Deng (Fellow, IEEE) received the Ph.D. degree from the Illinois Institute of Technology, Chicago, IL, USA, in 1985.

He is AXA Chair Professor of Cybersecurity, Director of the Secure Mobile Centre, School of Computing and Information Systems, Singapore Management University, Singapore. His research interests include applied cryptography, data security and privacy, and network security.