Verifiable Homomorphic Encrypted Computations for Cloud Computing

Ruba Awadallah
School of Computer Sciences
Universiti Sains Malaysia
Penang, Malaysia

Azman Samsudin
School of Computer Sciences
Universiti Sains Malaysia
Penang, Malaysia

Mishal Almazrooie
School of Computer Sciences
Universiti Sains Malaysia
Penang, Malaysia

***Abstract*—Cloud computing is becoming an essential part of computing, especially for enterprises. As the need for cloud computing increases, the need for cloud data privacy, confidentially, and integrity are also becoming essential. Among potential solutions, homomorphic encryption can provide the needed privacy and confidentiality. Unlike traditional cryptosystem, homomorphic encryption allows computation delegation to the cloud provider while the data is in its encrypted form. Unfortunately, the solution is still lacking in data integrity. While on the cloud, there is a possibility that valid homomorphically encrypted data beings swapped with other valid homomorphically encrypted data. This paper proposes a verification scheme based on the modular residue to validate homomorphic encryption computation over integer finite field to be used in cloud computing so that data confidentiality, privacy, and data integrity can be enforced during an outsourced computation. The performance of the proposed scheme varied based on the underlying cryptosystems used. However, based on the tested cryptosystems, the scheme has 1.5% storage overhead and a computational overhead that can be configured to work below 1%. Such overhead is an acceptable trade-off for verifying cloud computation which is highly needed in cloud computing.**

***Keywords*—Cloud computing; computation verification; data confidentiality; data integrity; data privacy; distributed processing; homomorphic encryption**

I. Introduction

The demanding needs of modern computing have prompted many enterprises to outsource their data solution to cloud service providers (CSP). CSP provides services that improve performance efficiency and ease of maintenance to the adopters. On top of the improvements, adopting cloud services also offers savings in information technology infrastructure costs, in which most of the infrastructure cost is transferred to the CSPs, and clients only pay for what are used. Thus, enterprises can conveniently store, maintain, and manage data files remotely with reduced operation costs [1]. The CSP market is currently brimming with CSPs and their innovative and competitive products. In general, the current success of cloud services is mostly on cloud storage. However, the market for cloud computing is also building up in momentum. Implementing cloud computing empowers enterprises to become more competitive by having computing platforms that are scalable, agile, and reliable. As a result, the growth in the cloud computing market is projected to reach US 623.3 billion by 2023 [2].

A typical cloud computing ecosystem consists of cloud users (client), CSP, and the network infrastructure that connects the client and the CSP. Models of CSP architecture consist of software as a service, infrastructure as a service, and platform as a service. In addition, there are a few CSP designs that include private, public, hybrid, and community clouds [3].

Even though the outlook for cloud computing is positive, this technology is always being plagued with the trade-offs between cost and security. The issue lies in the principle of cloud computing, where enterprises need to delegate the task of protecting their data to CSP [4]. Subsequently, data sovereignty is lost once the data is stored in a remote CSP. This absence of control for data security presents data protection problems. According to the Cloud Security Alliance (CSA) analysis, for the third time in a row [5], [6], [7], data breaches topped the list of threats in the cloud. Data is considered breached once its information is disclosed, manipulated, or used by unauthorized parties. A data breach may be the primary goal of a targeted attack or merely the result of human error. However, the management of CSP is central, and it cannot guarantee the reliability of its employees [8]. [9] found that the occurrence of internal breaches is more serious and costly than foreign attacks. The reason behind this result is that insiders know the system and attack valuable information, while outsiders steal what they have access to [10].

As part of the security risk assessment, data privacy, confidentiality, and integrity must be considered to mitigate potential risks. Privacy refers to the access control that the clients have over their data. Confidentiality means only authorized parties can access the data. In comparison, data integrity refers to the assurance of data consistency over its entire lifecycle [11].

In order to ensure privacy and confidentiality in cloud computing, researchers have indicated that homomorphic encryption (HE) is one of the promising methods for remote manipulations over encrypted data [12], [13]. However, although HE makes computation delegation possible, it has security flaws that can affect the validity of outsourced calculations. Specifically, HE is malleable in nature, which makes it noncompliance to the indistinguishability under adaptive chosen-ciphertext attack (IND-CCA2) security notation [14]. Therefore, data integrity is at stake with only HE itself versus centralized cloud data management. For cloud computing, the threats to data integrity can be numerous and varied. This paper addresses the problem of data integrity verification (DIV) of CSP computations over homomorphically encrypted data. The focus of this paper is on the CSP behavior that stores and computes sensitive data. Specifically, the threats from the CSP can be enumerated as follows:

1) An attacker (CSP) violates the data integrity by directly substituting the given ciphertext with another valid ciphertext.
2) An attacker (CSP) maliciously substitutes a given computation query with another valid query.

Integer-based HE has been extensively researched and used. Therefore, this research aims to achieve the computational integrity of HE over an integer finite field. That contributes to strengthening the security of HE against data tampering and thus achieving privacy, confidentiality and integrity of the processed data.

The rest of the paper is organized as follows. Section II presents the current DIV methods with their limitations. Section III illustrates the candidate HE cryptosystems and presents the proposal scheme. The results and discussion are shown in Section IV. Finally, Section V provides the conclusion.

II. Related Work

Researchers are still looking for a comprehensive security solution that can bring cloud computing to the mainstream. HE with different approaches have been utilized to address the DIV problem. Classical auditing methods on single data copy had a broad resonance in addressing this problem, which represented by provable data possession (PDP) techniques [15], [16], [17], [18], [19], [20] and proving the possibility of retrieval (POR) techniques [21], [22], [23], [24], [25], [26], [27]. However, these methods are ineffective in the case of data loss or corruption on the servers. Another alternative is to archive multiple replicas of each file to use if the original copy has compromised, this model is known as data integrity auditing under distributed such as [28], [29], [30], [31].

As the previous schemes by their nature allow for a limited number of queries, there are proposing solutions [32], [33], [34], [35] that assign auditing tasks to a single third-party auditor (TPA) that independently manages the data audits. There are also works [36], [37], [38] where the auditing task is assigned to multiple TPAs to benefit from simultaneous synchronous audit sessions. Nevertheless, all the aforementioned schemes focus on checking the data integrity stored in cloud computing servers without verifying the validity and efficiency of CSP computations over the data.

In the field of verifiable computing, a variety of methods, including such [39], [40], [41], [42], [43], [44], [45], [46], [47], [48], [49], [50], [51]. [52] the basic of many variants of the verifiable computing proposal afterward, that developed based on fully homomorphic encryption (FHE). Although FHE appears theoretically ideal, it is inefficient for practical implementations due to the difficulty of the requirements of high storage usage and heavy overhead computing, therefore it is not very powerful for use in many power-limited devices. Likewise, these schemes [53], [51], [54], [55], [56], [57] proposed impractical methods based on FHE.

Otherwise, [46] presented a verifiable scheme that implements a commitment utilizing probabilistically checkable proofs. At the same time, [44] extended the scope of verifiable computation in two essential directions: public delegation and public verifiability. While [47] used a quadratic arithmetic program and Elliptic curve encryption to obtain public verification commitment with a constant size however the number of executed operations. Also, [40] suggested a verifiable method of computations of quadratic polynomials over a large number of variables. Meanwhile, [45] tried to solve possible collusion attacks in the El-gamal scheme by re-encrypting the ciphertext using the receiver's public key. After that, [41] present a general Incremental verifiable database system by integrating the primitive vector commitment and the encrypted-then-incremental MAC encryption mode. And [48] suggested a framework using a hash function over ciphertext and dual-CSPs to check data duplication. [43] scheme promised an improved deduplication system in a hybrid cloud architecture. Furthermore [58] introduced the IKGSR scheme to improve the RSA key generation function based on the use of four giant prime numbers to generate encryption and decryption keys. In short, all of these proposals have the same framework idea of bounding the CSP generate a commitment, and accordingly, the client used this commitment to verify the CSP performance over his ciphertexts.

Whereas [50] proposed a public evaluation verification scheme over ciphertexts by interacting with the trusted authenticator (TA) and a public auditor proxy (PAP). Although they reduce the overload of both the cloud users and the verifier, it inefficient for practical applications due to using FHE's complicated scheme.

While numerous attempts have been made to overcome the CSP's computation cheating attack problems, they remain subject to certain fundamental flaws. First of all, most of the previous works are constructed for particular structures and cannot be included in other environments. This means that even a minor alteration can cause the schemes to fail due to their particular layouts. Furthermore, all the proposed models believe in the centralization of the CSP's authority or any third party over the data. As CSP can manipulate the computations applied to the data, it can generate the commitment value to match the applied computations. Thus, the client still receives an adequate commitment to computed ciphertexts, while the CSP perform the computation fraud attack.

In different ways, some researchers, such as [11] and [4], sought to use blockchain technology to prove the work of cloud service providers on cloud data. [11] relied on fiat cryptocurrencies such as Bitcoin and Ethereum to store the hash of the database issued by at least four cloud service providers and compare all the issued results. [4] used the proof of work consensus to delay the creation of a new record in the database to 6 minutes to create a single record as a minimum.

Despite the security effectiveness of the proposed schemes, they are pretty expensive; i.e. in addition to the cost of the required computations, blockchain implementation costs will be added as additional costs that clients will have to pay. Moreover, adopting the Byzantines Fault Tolerance consensus for both proposals would at least quadruple both costs. Also, using proof of work consensus in scheme [4] will impact cloud computing business performance. Therefore, our proposal will be based on modular arithmetic to provide a verification mechanism for the processes applied to the data at the lowest costs. Furthermore, the use of modular arithmetic dramatically increases digital signal processing performance in algorithms with extensive use of addition and multiplication. Thus, it

provides speed and low energy consumption and promises high reliability and fault tolerance [59], [60], [61]. The analysis of the latest scientific papers [62], [63], [64], [65], [66] confirms that the use of modular computation is continuously expanding. They ascribe that to the modular arithmetic's ability to increase the reliability of monitoring systems and their tolerance of errors significantly by increasing the resources used while preserving the operating time. As a result, many major companies such as Cisco and Kabushiki Kaisha Toshiba are rushing to research and apply modular arithmetic [67].

III. Verification Scheme Design

The migration of sensitive data into CSP is a source of security issues. If sensitive data are migrated into CSP, the client must be assured that proper data security measurements are in place. In order to ensure data privacy and confidentiality, this paper assumes the use of HE. Subsequently, this section presents the proposed scheme which enables the client to verify the integrity of the applied computations over the encrypted data. Fig. 1 shows the flow diagram of the proposed scheme.

A. Scheme Preliminaries

This paper proposes the use of HE over $\mathbb{Z}_p^*$ in encrypting the data before sending them to CSP, thus allows CSP to perform operations on the encrypted data at the client's request, without disclosing its content. In the context of this paper, the HE is briefly defined as follows. An HE over operation '$\diamond$' in a finite field $\mathbb{Z}_p^*$ is an encryption scheme that supports the following equation:

$$Enc_{k_e}(m_1) \diamond Enc_{k_e}(m_2) = Enc_{k_e}(m_1 \diamond m_2), \quad \forall\, m_1, m_2 \in \mathbb{Z}p^*, \tag{1}$$

where $Enc(.)$ is an encryption algorithm, k_e is the encryption key and $(m_1,\ m_2)$ are plaintexts. An HE scheme is primarily characterized by four operations: KeyGen, Enc, Dec, and Eval. Eval is an HE-specific operation, which takes ciphertexts as input and outputs a ciphertext corresponding to the plaintexts [68]. The Eval function in this paper supports both addition and multiplication operations over $\mathbb{Z}_p^*$. Table I summarizes the math notations used in this article. Depending on the supported homomorphism features, HE schemes can perform different type of operations. At any given time, the Partial Homomorphic Encryption (PHE) scheme can only perform one type of computation operation. It can be either a multiplicative homomorphism; e.g. RSA [69], and ElGamal [70], or additive homomorphism; e.g. Benaloh [71], Paillier [72], and Okamoto-Uchiyama (OU) [73]. While Somewhat Homomorphic Encryption (SWHE) scheme is a cryptosystem which supports both properties but for limited number of operations. Such as Boneh-Goh-Nissim (BGN) [74] which allowing unlimited number of additions, but only one multiplication. In this paper six HE schemes over Z_p are benchmarks against the propose scheme. The following subsections introduce the six cryptosystems.

1) RSA Cryptosystem: RSA is a block cipher algorithm over integer finite field which support evaluation function for only homomorphic multiplication computations over ciphertext [69]. In RSA, the plaintext and the ciphertext (which are represented as positive integers) are bounded by n, where n is defined as $n < 2^{4096}$ for practical purposes. Following are the four main operations governing the RSA multiplicative-PHE cryptosystem:

TABLE I. Mathematical Notation

Notation	Explanation
c_i	a ciphertext
m_i	a plaintext
p	a large prime number
q	a large prime number
n	a modulus
λ	an encryption security parameter
$\mathbb{Z}_n^*$	a set of integer modulo n
$\mathbb{Z}_p^*$	a set of integers modulo p
$\mathbb{Z}_r^*$	a set of integer modulo r
$\mathbb{Z}_T^*$	a set of integer modulo T
Prk	a private key
Puk	a public key
$KeyGen$	a homomorphic Prk and Puk generation
$Dec_{Prk}(c_i)$	a homomorphic decryption of c_i using Prk
$Enc_{Puk}(m_i)$	a homomorphic encryption of m_i using Puk
$Eval$	a homomorphic function of m
$\diamond$	a computation function
c_r	a homomorphic computed result in ciphertext
G	a cyclic group
$\mathbb{G}$	a multiplicative group

- KeyGen: The public key $Pk = \{e, n\}$ and the private key $Prk = \{d, n\}$ are built upon two large prime numbers p, q such that $p \neq q$, and $n = p \times q$. The integer e is randomly selected such that $\gcd(\phi(n), e) = 1,\ 1 < e < \phi(n)$ and $d \equiv e^{-1} \bmod \phi(n)$, where $\phi(n) = (p-1)(q-1)$.
- Enc: The public key $Pk = \{e, n\}$ is used to encrypt plaintext $m \in \{0,1\}^*$ as shown by Equation (2).

$$c = Enc_{Pk}(m) = m^e \bmod n. \tag{2}$$

- Dec: The ciphertext c can be decrypted by using private key $Prk = \{d, n\}$ as shown in Equation (3).

$$m = Dec_{Prk}(c) = c^d \bmod n. \tag{3}$$

- Eval: RSA cryptosystem satisfies multiplicative homomorphism as shown in Equation (4).

$$\begin{aligned} Enc(m_1) \times Enc(m_2) &= (m_1^e \bmod n) \times (m_2^e \bmod n), \\ &= (m_1 \times m_2)^e \bmod n, \\ &= Enc(m_1 \times m_2). \end{aligned} \tag{4}$$

2) ElGamal Cryptosystem: ElGamal proposed a probabilistic cryptography scheme based on public key cryptosystem in 1985 [70]. The scheme is based on Diffie–Hellman key exchange. The security of the scheme is based on the security of the discrete logarithm problem. A simple ElGamal's scheme is as follows:

- KeyGen: Key generation process required a cyclic group G with order n using generator g. $(h = g^y)$ is calculated based on a randomly chosen $y \in \mathbb{Z}_n^*$. The public key and the private keys are defined as $Pk = \{G, n, g, h\}$ and $Prk = \{y, n\}$, respectively.

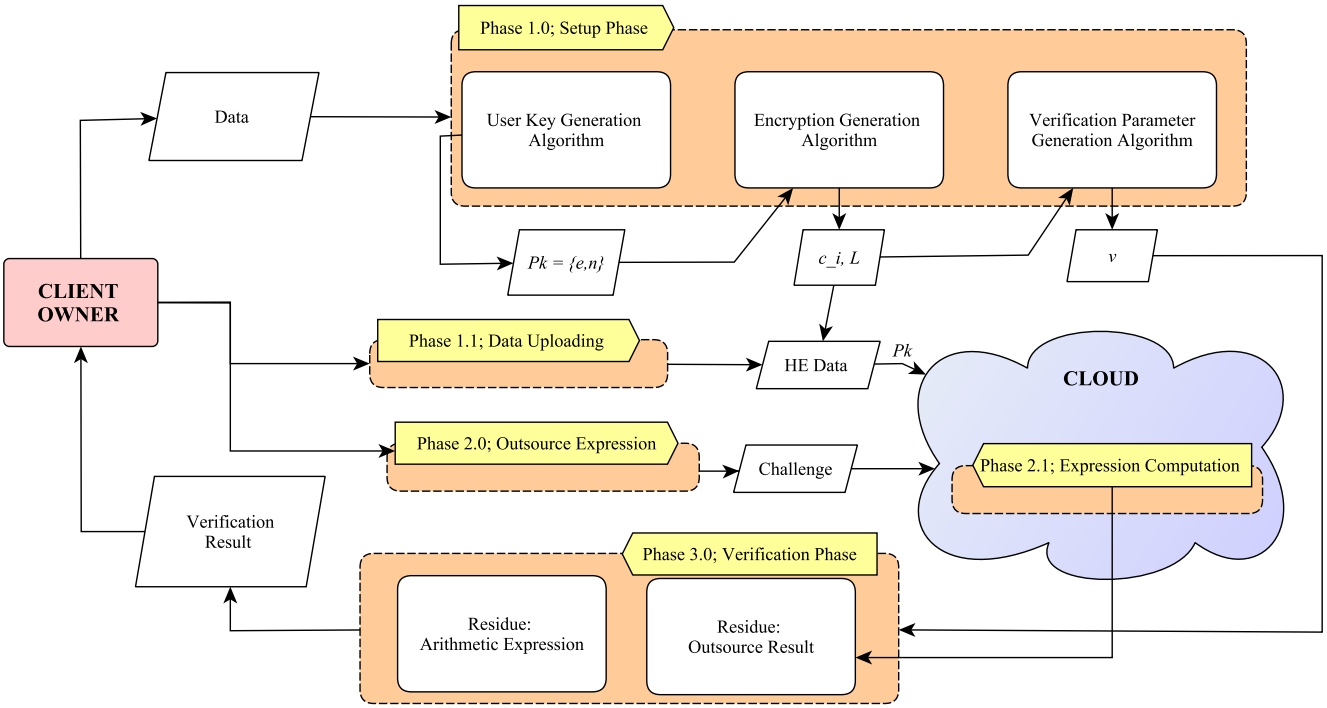

Fig. 1. Proposed Scheme Flow Diagram.

- Enc: The encryption of plaintext m requires a random integer r to be selected and kept hidden. The result of encrypting plaintext m is a ciphertext pair $c = (c_1, c_2)$ which is defined as follows:

$$(c_1, c_2) = Enc_{Pk}(m) = (g^x, mh^x) = (g^x, mg^{xy}) \tag{5}$$

- Dec: The decryption is performed by using the private key $\{y, n\}$ to compute $s = {c_1}^y$, followed by the decryption process itself as shown in the following equation:

$$Dec(c) = c_1 \times s^{-1} = mg^{xy} \times g^{-xy} = m. \tag{6}$$

- Eval: ElGamal cryptosystem satisfies multiplicative homomorphism as shown in Equation (7).

$$\begin{aligned} Enc(m_1) \times Enc(m_2) &= (g^{x_1}, m_1 h^{x_1}) \times (g^{x_2}, m_2 h^{x_2}) \\ &= (g^{x_1+x_2}, m_1 \times m_2 h^{x_1+x_2}) \\ &= E(m_1 \times m_2). \end{aligned} \tag{7}$$

3) Benaloh Cryptosystem: Benaloh scheme is based on the Goldwasser-Micali (GM) public key cryptosystem [71]. Benaloh scheme enhances the GM scheme by encrypting in blocks of bits rather than encrypting bit by bit. Security assumption of Benaloh scheme is based on the higher residuosity problem which is the generalization of quadratic residuosity problems (x^2). Following is the description of the Benaloh additive-PHE cryptosystem:

- KeyGen: For a given block size r, two large primes p and q are selected such that $\gcd(r, (p-1)/r) = 1$ and $\gcd(r, (q-1)) = 1$. Subsequently n and $\phi(n)$ are calculated as $n = pq$ and $\phi(n) = (p-1)(q-1)$, respectively. $y \in \mathbb{Z}_n^*$ is selected such that $(y^{\frac{\phi}{r}}) \equiv 1 \bmod n$, where $\mathbb{Z}_n^*$ is the multiplicative subgroup of integers modulo n which includes all the numbers smaller than r and relatively prime to r. The public key is published as (y, n), while (p, q) represents the private key.

- Enc: To encrypt a plaintext $m \in \mathbb{Z}_r$, where $\mathbb{Z}_r = \{0, 1, ..., r-1\}$, a random $u \in \mathbb{Z}_r^*$ is selected. The encryption equation is as shown below:

$$c = Enc_{Pk}(m) = (y^m u^r) \bmod n. \tag{8}$$

- Dec: The decryption process is done through an exhaustive search for $i \in \mathbb{Z}_r$, in which the plaintext m can be recovered by using Equation (9).

$$m = (y^{-i} c)^{\phi/r} \bmod n. \tag{9}$$

- Eval: Benaloh cryptosystem satisfies additive homomorphism as shown in Equation (10).

$$\begin{aligned} Enc(m_1) \times Enc(m_2) &= ((y^{m_1} {u_1}^r) \bmod n)) \\ &\quad \times ((y^{m_2} {u_2}^r) \bmod n), \\ &= (y^{m_1+m_2} (u_1 \times u_2)^r) \bmod n, \\ &= E((m_1 + m_2) \bmod n). \end{aligned} \tag{10}$$

4) Okamoto-Uchiyama Cryptosystem: [73] proposed a new additive cryptosystem which improve the computational performance by defining $n = p^2q$ within the same domain of $\mathbb{Z}_n^*$. The security assumption of OU cryptosystem is based on the p-subgroup that makes it equivalent to the factorization of n. Following is the OU cryptosystem:

- KeyGen: After determining the value of n, a random number $g \in \{2, \ldots, n-1\}$ is selected such that $g^{p-1} \not\equiv 1 \mod p^2$. Subsequently h can be calculated as $h = g^n \mod n$. The public key and the private key are $\{n, g, h\}$ and $\{p, q\}$, respectively.
- Enc: A plaintext $m < p$ can be encrypted with the public key $Pk = \{n, g, h\}$ as shown in Equation (11). $r \in \{1, \ldots, n-1\}$ is randomly selected.

$$c = Enc_{Pk}(m) = g^m h^r \mod n. \tag{11}$$

- Dec: To recover the plaintext, the private key $Prk = \{p, q\}$ is used with Equation (12).

$$\begin{aligned} a &= \frac{(c^{p-1} \mod p^2) - 1}{p}, \\ b &= \frac{(g^{p-1} \mod p^2) - 1}{p}, \\ b' &= b^{-1} \mod p, \\ Dec_{Prk}(c) &= ab' \mod p. \end{aligned} \tag{12}$$

- Eval: OU cryptosystem satisfies additive homomorphism as shown in Equation (18).

$$\begin{aligned} Enc(m_1) \times Enc(m_2) &= ((g^{m_1} h^{r_1}) \mod n) \\ &\quad \times ((g^{m_2} h^{r_2}) \mod n), \\ &= (g^{m_1+m_2} h^{r_1+r_2}) \mod n, \\ &= Enc(m_1 + m_2). \end{aligned} \tag{13}$$

5) Paillier Cryptosystem: Paillier cryptosystem is a probabilistic public key cryptosystem based on higher-order residual classes which support only additive homomorphism computations [72]. Following are the four main operations governing the Paillier additive-PHE cryptosystem:

- KeyGen: Paillier cryptosystem has a set of keys. $p \in \mathbb{Z}_n^*$, $q \in \mathbb{Z}_n^*$, $g \in \mathbb{Z}_{n^2}^*$ are randomly selected such that $\gcd\big(L(g^\lambda \mod n^2), n\big) = 1$, where p, q are two large primes, $n = p \times q$ and functions L and λ are defined as follows:

$$L(u) = (u-1)/n. \tag{14}$$

$$\lambda = lcm\big((p-1)(q-1)\big). \tag{15}$$

- Enc: The encryption process utilizes the public key $Pk = \{n, g\}$ to encrypt an arbitrary plaintext $m \in \mathbb{Z}_n^*$ with a randomly selected integer $r \in \mathbb{Z}_n^*$ to produce ciphertext c.

$$c = Enc_{Pk}(m) = g^m r^n \mod n^2. \tag{16}$$

- Dec: The decryption process uses the private key $Prk = \lambda$ in the decrypting process as shown by Equation (17).

$$Dec_{Prk}(c) = \Big(\frac{L(c^\lambda \mod n^2)}{L(g^\lambda \mod n^2)}\Big) \mod n = m. \tag{17}$$

- Eval: Paillier cryptosystem satisfies additive homomorphism as shown in Equation (18).

$$\begin{aligned} Enc(m_1) \times Enc(m_2) &= (g^{m_1} r_1^n \mod n^2) \\ &\quad \times (g^{m_2} r_2^n \mod n^2), \\ &= \big(g^{(m_1+m_2)} (r_1 \times r_2)^n\big) \mod n^2, \\ &= Enc(m_1 + m_2). \end{aligned} \tag{18}$$

6) Boneh-Goh-Nissim Cryptosystem: BGN defined a Paillier-like cryptosystem with an unlimited number of homomorphic additions and a single multiplication on the plaintext [74]. BGN cryptosystem is described as follows:

- KeyGen: A two large prime numbers q and r are chosen to produce the value of $n = qr$ and a positive integer $T < q$ which is selected randomly. Subsequently, two multiplicative groups $\mathbb{G}$, $\mathbb{G}_1$ of order n that support a bilinear pairing $e\colon (\mathbb{G} \times \mathbb{G}) \to \mathbb{G}_1$ are selected. Random generators g, u are chosen where $g, u \in \mathbb{G}$, and $h = uq$ where h is a generator of the subgroup of order p. The public key is composed of $Pk = \{n, g, h, \mathbb{G}, \mathbb{G}_1, e\}$, and the private key is $Prk = \{p, n\}$.
- Enc: For a plaintext $m \in \mathbb{Z}_T$ a random $r \in \mathbb{Z}_n$ is selected. The encryption process is as shown by Equation (19).

$$Enc_{Pk}(m) = c = g^m h^r \mod n \tag{19}$$

- Dec: Decrypting ciphertext $c \in \mathbb{G}$ by using private key $Prk = \{p, n\}$ is shown in Equation (20). Message m can be recovered in time $O(\sqrt{(T)})$ since the message is bounded by T.

$$\begin{aligned} c^p &\equiv (g^m h^r)^p \mod n \\ &\equiv (g^m)^p \mod n \\ &\equiv (g^p)^m \mod n \end{aligned} \tag{20}$$

- Eval: BGN satisfies unlimited additive homomorphism as shown in Equation (21) and a single multiplicative homomorphism as represented in Equation (22).

$$\begin{aligned} Enc(m_1) \times Enc(m_2) &= ((g^{m_1} h^{r_1}) \mod n) \\ &\quad \times ((g^{m_2} h^{r_2}) \mod n) \\ &= (g^{m_1+m_2} h^{r_1+r_2}) \mod n \\ &= E(m_1 + m_2) \end{aligned} \tag{21}$$

$$\begin{aligned} Enc(m_1) \times k &= {c_1}^k \mod n \\ &= (g^{m_1} h^{r_1})^k \mod n \\ &= (g^{km_1} h^{r_1 k}) \mod n \\ &= E(m_1) \times k \end{aligned} \tag{22}$$

HE cryptosystem is malleable, and therefore it is not IND-CCA2 secured by design. Data integrity can still be compromised by CSP and can go undetected. For example, the CSP

can implicitly substitute given ciphertext or the cumulative result with other valid ciphertext without the need to know the content of those substituted data. Different from confidentiality and privacy, once integrity is compromised there is no way to restore the original data. Therefore, data integrity needs to be enforced on such outsource computations.

B. Proposed Scheme

The verification scheme has three phases: environment setup, computation outsourcing to the CSP by the client, and computation validation of CSP's work by the client, in which the last two phases can be repeated as required (see Table II). The three phases are thoroughly discussed in the subsequent context.

Phase 1 (Setup): The client initiates the initialization phase by defining system parameters. The propose scheme consists of two different number systems. The first number system is the finite field $\mathbb{Z}_p^*$ where the HE calculations take place. The second number system is an n-bit binary number system, where n is a positive integer such that 2^n is much smaller than the prime p. The HE encryption function takes as input a public key k_e and message m of index i and produces a ciphertext c_i as an output, as shown in Equation (23).

$$c_i = Enc_{k_e}(m_i);\ m_i, c_i \in \mathbb{Z}_p^*. \tag{23}$$

Subsequently, the client identifies a positive integer $v < L$ as the secret value, where L is the largest integer allowed in the implemented system. v will also serves as the verification parameter.

Phase 2 (Outsource): In this phase, the client sends its outsource calculations in a form of an arithmetic equation to the CSP. In return the CSP executes the requested calculations and returns the corresponding result back to the client. From the client's repository, the client sends the arithmetic expression, <expr>, to CSP for evaluation. The expression, <expr>, consists of ciphertexts that had been encrypted by using HE with the corresponding arithmetic operators (e.g. "$(101 + 202) \times 303$"). The syntax of the <expr> follows the following grammar:

```
   <expr> ::= <term> '+' <expr> | <term>
   <term> ::= <factor> '×' <term> | <factor>
<factor> ::= '(' <expr> ')' | <const>
  <const> ::= integer
```
(24)

In return, CSP calculates the requested arithmetic expression before sending the corresponding result, c_r (e.g. "91809"), back to the client.

Phase 3 (Validation): To verify CSP's calculation(s), the client needs to assure that the value received from CSP, c_r, is the result of the arithmetic expression outsourced earlier, <expr>.

The equality of a basic arithmetic expression can be validated by evaluating its modular residue. Let <expr> be the outsourced arithmetic expression send by the client to the CSP and let c_r be the calculation result received by the client from CSP. Subsequently, the client can validate c_r by comparing the modular residues of both c_r and <expr>, as depicted by Equation (25).

$$\begin{aligned} c_r &= \texttt{<expr>}. \\ c_r \bmod v &= \texttt{<expr>} \bmod v. \end{aligned} \tag{25}$$

To simplify the calculation of the right-hand-side of Equation (25), the expression, <expr>, is further expanded by using the following grammar based on modular arithmetic properties.

```
   <expr> ::= <term> '+' <expr> | <term>
   <term> ::= <factor> '×' <term> | <factor>
<factor> ::= '(' <expr> ')' mod v |
             <const> mod v
  <const> ::= integer
```
(26)

TABLE II. WORKING EXAMPLES OF THE PROPOSE VERIFIED SCHEME BASED ON THE ARITHMETIC EXPRESSION $c_r = ((c_1 + c_2) \times c_3)$: (A) SETUP, (B) OUTSOURCING, (C) VALIDATION

(a) Setup: client determines modulus and identifies ciphertexts.

Modulus and Ciphertexts	Example 1	Example 2	Example 3
v	3_{10}	62_{10}	158_{10}
c_1	4_{10}	101_{10}	99999_{10}
c_2	7_{10}	202_{10}	88888_{10}
c_3	8_{10}	303_{10}	77777_{10}

(b) Outsource: CSP computes the outsourced expression.

Outsourced Computation	Example 1	Example 2	Example 3
$c_r = ((c_1 + c_2) \times c_3)$	88_{10}	91809_{10}	14691064199_{10}

(c)Validation: client validates result by comparing residues.

Intermediate Values and Residues		Example 1	Example 2	Example 3
Intermediate Values	$c_1 \bmod v$	1_{10}	39_{10}	143_{10}
	$c_2 \bmod v$	1_{10}	16_{10}	92_{10}
	$c_3 \bmod v$	2_{10}	55_{10}	41_{10}
	$((c_1 \bmod v)+ (c_2 \bmod v)) \bmod v$	4_{10}	3025_{10}	9635_{10}
Residue: Arithmetic Expression	$((((c_1 \bmod v)+ (c_2 \bmod v)) \bmod v) \times (c_3 \bmod v)) \bmod v$	1_{10}	49_{10}	55_{10}
Residue: Outsource result	$c_r \bmod v$	1_{10}	49_{10}	155_{10}

IV. RESULT AND DISCUSSION

In this section, the data storage requirement and computation performance are analyzed when implementing the proposed scheme. The client and the CSP are simulated with different machines capacity to reflect the actual setting of the two domains. Cloud computing enterprises offer different computing instances with different performances as shown in Table III. To reflect such capabilities, an Intel® Core(TM) i7-3770 CPU, 3.40GHz CPU, 12GB RAM machine was used

to emulate the CSP computation environment. On the other hand, the client computations are simulated on a machine with Intel® Core(TM) i5-7500U, 2.70GHz CPU, and 4GB RAM.

On the software side, the proposed verified scheme that is running at CSP was implemented using Numpy [75], a compiled library which is efficient in manipulating big integer calculations for Python. For the client implementation which does not require big integer calculations, a basic C++ compiler was used when implementing the proposed scheme. This paper further assumes the use of the two HE properties; PHE represented by multiplicative homomorphic (RSA, ElGamal) and additive homomorphic (Benaloh, OU, Paillier) and SWHE represented by the BGN's method.

TABLE III. GENERIC CLOUD SERVICE PROVIDER CONFIGURATIONS

CSP	Configuration	CPU	Virtual CPUs	Memory (GiB)
AWS Elastic Cloud Computing™ [76]	Low	Intel Xeon E5-2666 v3	2	3.75
	High	Intel Xeon Platinum	72	192
Azure Virtual Machine™ [77]	Low	Intel Xeon Platinum 8168	2	4
	High	Intel Xeon Platinum 8168	72	144
Google Cloud Compute Engine™ [78]	Low	Intel Xeon Scalable	4	16
	High	Intel Xeon Scalable	60	240

In the following subsections, we present the results of applying the verification scheme to the different ciphertext sizes generated from candidate cryptosystems. The purpose of these calculations is to assess the implementation costs and performance of the proposed scheme for all candidate cryptosystems.

A. Storage Analysis

For the storage analysis, it is important to analyzes the storage requirement to store the residues of all the encrypted data at the client side against the actual encrypted data stored at the CSP. Storing the residues at the client side is an overhead to the proposed verified scheme, which does not exist in a normal HE implementation. To gauge the CSP's storage requirement, a few assumptions are made. Among the assumptions is the modulus size. NIST recommends 2048-bit as the minimum size for the factoring modulus. While for a more secured applications, factoring modulus of at least 3072-bit is recommended [79]. To simplify calculation while adopting highest modulus value, this paper assumes 4096-bit as the factoring modulus.

Another assumption is the machine word-size. Current CPUs typically operate on 32- or 64-bit data, stacked of 4-bits or 8-bits based on ISO/IEC 2382:2015 standard [80]. The storage requirement for the verified scheme depends on the size of the verifying parameter, v. In order to simulate primitive encryption, we assume the client operates on 64-bit data which is typical in most modern desktop computers. Thus, the storage requirement at the client side is less than 1.5% of the CSP full storage. To put this result into perspective, for a client who owns petabytes of data stored at the CSP, this scheme will require the client to store only terabytes of the corresponding data at the client side which is feasible on current modern desktop.

TABLE IV. MULTIPLICATIVE HOMOMORPHIC CALCULATIONS: VERIFICATION COST AGAINST THE COST OF PERFORMING THE ACTUAL HOMOMORPHIC CALCULATION

Number of Operations per Verification	CSP: RSA Actual Multiplicative Homomorphic Calculation* (μsec)	Client: RSA-Verified Method Verification Calculation* (μsec)	 Overhead (%)
1	63.83332	0.477777	0.748
10	745.9444	0.833333	0.111
100	88345.34	0.922222	< 0.000
1000	7071374.8	2.5001	< 0.000
10000	1120000000	17.254	< 0.000
Number of Operations per Verification	**CSP: ElGamal** **Actual Multiplicative Homomorphic Calculation* (μsec)**	**Client: ElGamal-Verified Scheme** **Verification Calculation* (μsec)**	 **Overhead (%)**
1	115.0222	0.566666	0.49265
10	1592.375	1.622222	0.1018
100	299578.7	2.9785	< 0.000
1000	37347510.18	7.222222	< 0.000
10000	2749672997	33.98888	< 0.000

* Average of 10 readings.

B. Performance Analysis

It is crucial to analyze the calculation overhead of the proposed verification scheme, that is, it is important to gauge the acceptable number of calculations per verification in order to reduce the calculation overhead in the proposed scheme. The analysis in this section is therefore qualitative in nature and based on how the proposed scheme works in terms of homomorphic operations. In general, the computations performed by the client's machine is slightly faster than the computations performed by the CSP when processing single outsource expression (one expression, one verification). However, a series of expressions (many expressions per verification) can reduce the calculation overhead extremely.

In case of multiplicative-PHE, on verifying one RSA multiplication calculation the client is required to perform one 64-bit multiplication and two 64-bit modular operations on the residues of the two corresponding ciphertexts, while the CSP homomorphically performs one big integer multiplication (4096-bit) operation on the two ciphertexts. Whereas ElGamal cryptosystem needs to double up the RSA computations for one multiplication operation because the nature of the scheme in producing ciphertext pair for each single plaintext. Table IV shows the simulation results in multiplicative-PHE over x operations. Where Fig. 2 and Fig. 3 show the application of the proposed scheme to RSA and ElGamal cryptosystems, respectively. Both figures demonstrate the requested time variance between the client verification process and the CSP processing the data. Fig. 4 illustrates the corresponding overhead in processing multiplicative-PHE expressions per verification. Both encryption systems converge in the overhead percentage. In which a performing of one verification process per each computation process is less than 0.01%, but it quickly drops further to less than $1.00E^{-7}\%$ in one verification process per 10000 computations.

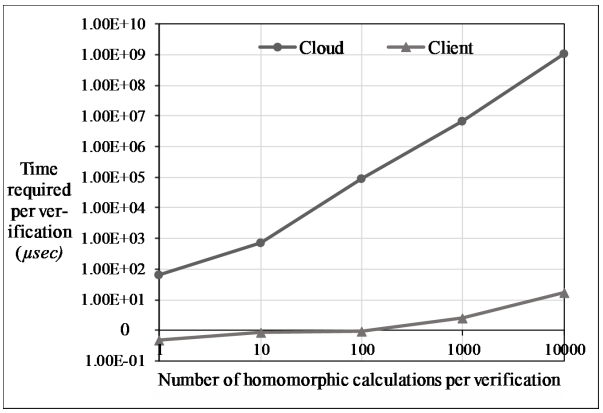

Fig. 2. Verified Scheme over RSA.

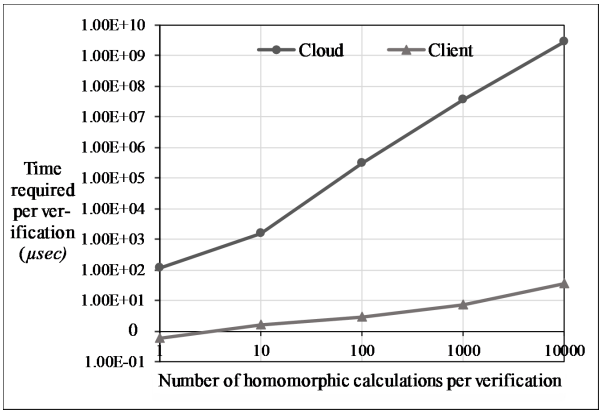

Fig. 3. Verified Scheme over ElGamal.

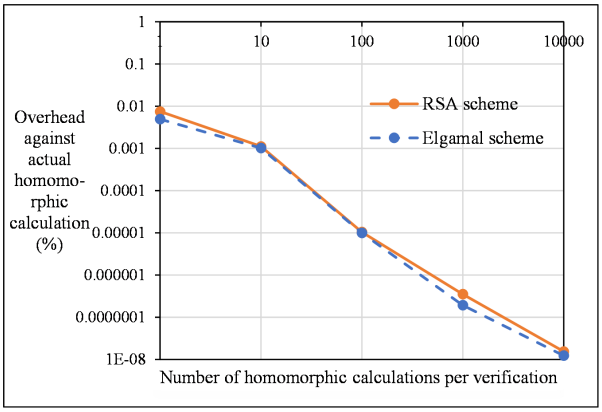

Fig. 4. Cost Overhead for Verifying Multiplicative-PHE Calculation.

TABLE V. ADDITIVE HOMOMORPHIC CALCULATIONS: VERIFICATION COST AGAINST THE COST OF PERFORMING THE ACTUAL HOMOMORPHIC CALCULATION

Number of Operations per Verification	CSP: Benaloh Actual Additive Homomorphic Calculation* (μsec)	Client: Benaloh-Verified Scheme Verification Calculation* (μsec)	Overhead (%)
1	70.2354	0.5324	0.75802
10	821.267	0.9924	0.12083
100	96514.354	1.352	< 0.000
1000	8075412.25	3.1231	< 0.000
10000	1750000000	27.564	< 0.000
Number of Operations per Verification	**CSP: OU** **Actual Additive Homomorphic Calculation* (μsec)**	**Client: OU-Verified Scheme** **Verification Calculation* (μsec)**	**Overhead (%)**
1	121.733	0.78889	0.6480
10	1663.42	1.02222	0.06145
100	154769.59	1.5	< 0.000
1000	14875365	3.12222	< 0.000
10000	2394117441	28.9556	< 0.000
Number of Operations per Verification	**CSP: Paillier** **Actual Additive Homomorphic Calculation* (μsec)**	**Client: Paillier-Verified Scheme** **Verification Calculation* (μsec)**	**Overhead (%)**
1	86.48889	0.6964	0.80519
10	1082.333	0.96667	0.08931
100	139625.84	1.1	< 0.000
1000	9175365.1	2.88889	< 0.000
10000	1921808022	29.874	< 0.000

* Average of 10 readings.

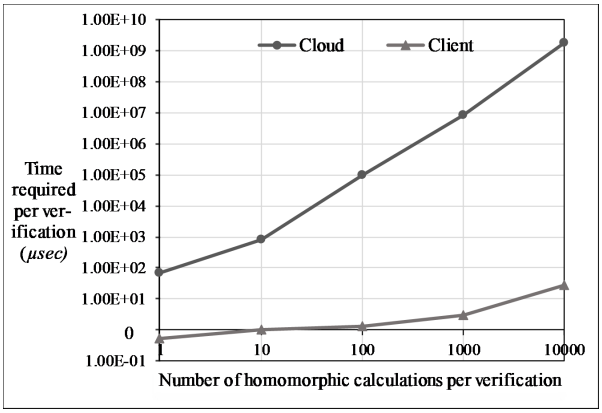

Fig. 5. Verified Scheme over Benaloh.

The schematic for additive-PHE verification is similar to the multiplicative-PHE. For verifying x additive Benaloh or OU calculations, the client performs x multiplications and two modular operations. Whereas in verifying x additive Pailier calculations, the client performs x multiplications and two modular n^2 operations, as seen in Section III-A. On the other hand, the CSP homomorphically performs x big integer multiplications (4096-bit). Table V shows the CSP and client average execution time for the verified scheme in additive-PHE. Fig. 5, Fig. 6 and Fig. 7 display the results of applying the verification scheme over the selected additive-PHE cryptosystems. Also, Fig. 8 shows the overhead results in processing additive-PHE expressions per verification for the selected cryptosystems. They are at rates less than 0.01% and rapidly decline to be approximately $1.00E^{-5}\%$ in only 100 applied computations.

The verification for a BGN calculation consists of verifying additive homomorphic and multiplicative homomorphic at the same time. For x BGN computation, the verification process at the client side involves x 64-bit addition, one 64-bit multiplication and two 64-bit binary operations on the residues of the two respective ciphertexts, while the CSP homomorphically conducts x big integer addition and one

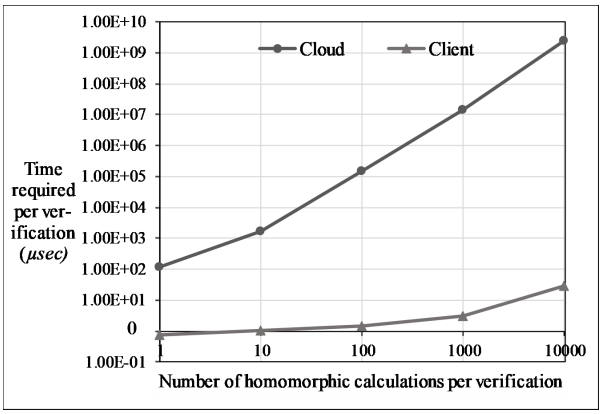

Fig. 6. Verified Scheme over OU.

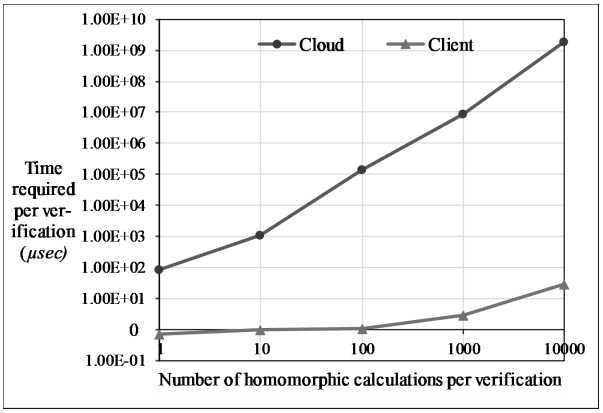

Fig. 7. Verified Scheme over Pailler.

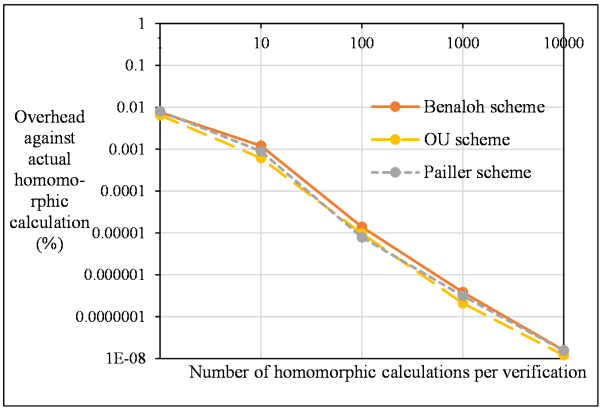

Fig. 8. Cost Overhead for Verifying Additive-PHE Calculation.

big integer multiplication (4096-bit) operations on the two ciphertexts. Simulation result shown in Table VI and Fig. 9. Where Fig. 10 indicates that the verification overhead which is performed by the client is about 2.9% of the computation time needed by the CSP to process the real BGN calculation, moreover, it shows the overhead at the client side decreasing fast, as the number of calculations per verification increases.

TABLE VI. SOMEWHAT HOMOMORPHIC CALCULATIONS: VERIFICATION COST AGAINST THE COST OF PERFORMING THE ACTUAL HOMOMORPHIC CALCULATION

Number of Operations per Verification	CSP: BGN Actual Additive & One Multiplicative Homomorphic Calculation* (μsec)	Client: Verified Scheme	
		Verification Calculation* (μsec)	Overhead (%)
1	128.3111	3.7	2.8836
10	1768.6644	6.87778	0.03889
100	225805.97	8.18889	$<$ 0.000
1000	41472148	11.9333	$<$ 0.000
10000	3168250731	29.77778	$<$ 0.000

* Average of 10 readings.

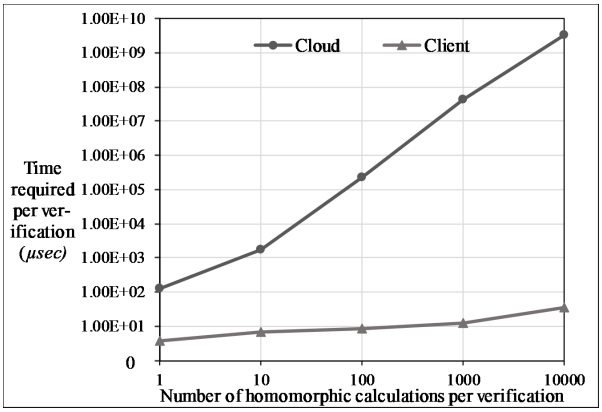

Fig. 9. Verified Scheme over BG.

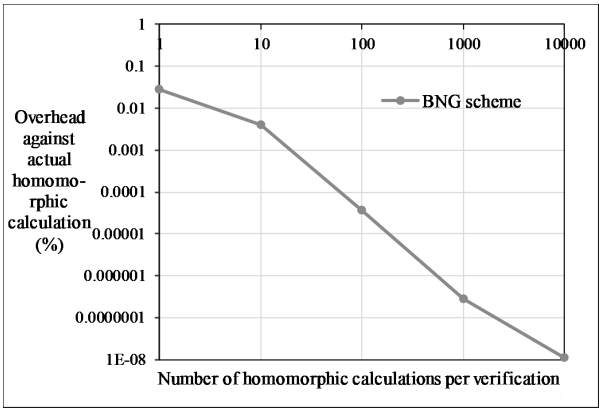

Fig. 10. Cost Overhead for Verifying SWHE Calculation.

C. Security Analysis

The proposed verification mechanism enhances the security of HE against data tampering. That is, HE cryptosystems with this mechanism are able to provide data integrity, not only confidentiality and privacy. Now the client can infer if any data breach occurred from substitution in ciphertext or change in the query process. In which the verification result does not match the results sent to the client.

D. Discussion

In general, the overhead of the propose scheme does not varied too much against the homomorphic cryptosystems.

This is because all the mentioned cryptosystems are based on the integer finite field, in which both the multiplicative-PHE and additive-PHE are being designed by manipulating only the multiplication operation on the cihpertexts. Across the board, the overhead is high since the propose scheme is verifying by invoking calculations within the integer finite field. However, as shown in the previous section, amortization plays an important role in reducing the overhead, that is, one verification calculation is used to verify a batch of outsourced calculations. This is attributed to the increase in the execution time discrepancy between the CSP and the client; that is, the increase in the number of multiplications that CSP applies to the encrypted data against the execution time for verification which changes very slightly on the client machine.

It is also important to note that the increase in the ciphertext size due to the different cryptosystems and the size of the finite field do affect the overall performance of the propose scheme. It is also important to note that the propose scheme is less efficient on BGN with SWHE feature. This is due, partly to the high cost of exponential operation that was used to represent a single multiplication operation.

V. Conclusion

This paper addresses the problem of DIV of outsource computation. In the context of outsourcing computation to CSP, HE over $\mathbb{Z}_p^*$ does provide data confidentiality but lacks in data integrity. This paper presents an efficient DIV scheme for HE over $\mathbb{Z}_p^*$ by evaluating the modular residue of the outsource calculation. The propose scheme is flexible and extensible in design, in which the number of modulus that can be used is limited only by the word size of the client's machine. With a 64-bit machine, there are technically 2^{64} possible modules. Subsequently, base on a 64-bit machine the storage requirement on the client's machine is less than 1.5% of the data size stored at the CSP. Across different cryptosystems tested, the worst computational overhead performed by the client is less than 3% of the actual homomorphic calculation perform by the CSP, that is, if one verification is applied to one homomorphic calculation. The worst computational overhead reduces to less than 0.1%, if one verification is performed for every 10 homomorphic calculations. It is also worth noting that the cryptosystems tested are slightly varied in their performances. In general, the proposed verified scheme can be implemented on any homomorphic cryptosystem that operates over the integer finite field $\mathbb{Z}_p^*$ without much restriction. Although the scheme solves the problem of verifying the integrity of the data computations, it may constitute a burden on the client to provide storage and extra work to achieve the verification phase. Therefore, we aim to shift the verification process to decentralized fog nodes, which communicate through a consensus in future work.

Acknowledgment

This work was supported by MOHE under the Fundamental Research Grant Scheme with grant number: FRGS/1/2020/ICT07/USM/01/1.

References

[1] M. Mohammed and F. Abed, "A symmetric-based framework for securing cloud data at rest," *Turkish Journal of Electrical Engineering & Computer Sciences*, vol. 28, no. 1, pp. 347–361, 2020.

[2] M. R. Report, "Cloud computing market," https://www.marketsandmarkets.com/Market-Reports/cloud-computing-market-234.html, note = Accessed: 2020-05-12, 2019.

[3] P. K. Senyo, E. Addae, and R. Boateng, "Cloud computing research: A review of research themes, frameworks, methods and future research directions," *International Journal of Information Management*, vol. 38, no. 1, pp. 128–139, 2018.

[4] R. Awadallah and A. Samsudin, "Using blockchain in cloud computing to enhance relational database security," *IEEE Access*, 2021.

[5] C. S. Alliance, "Practices for secure development of cloud applications," https://safecode.org/practices-for-secure-development-of-cloud-applications/, 2013, accessed: 2021-01-14.

[6] ——, "Top threats research," https://cloudsecurityalliance.org/group/top-threats/, 2016, accessed: 2021-01-14.

[7] ——, "Top threats to cloud computing: Egregious eleven," https://cloudsecurityalliance.org/artifacts/top-threats-to-cloud-computing-egregious-eleven/, 2020, accessed: 2021-01-14.

[8] N. Anciaux, M. Benzine, L. Bouganim, P. Pucheral, and D. Shasha, "Ghostdb: querying visible and hidden data without leaks," in *Proceedings of the 2007 ACM SIGMOD international conference on Management of data*, 2007, pp. 677–688.

[9] S. Pearson and A. Benameur, "Privacy, security and trust issues arising from cloud computing," in *2010 IEEE Second International Conference on Cloud Computing Technology and Science*. IEEE, 2010, pp. 693–702.

[10] R. Richardson and C. Director, "Csi computer crime and security survey," *Computer security institute*, vol. 1, pp. 1–30, 2008.

[11] R. Awadallah, A. Samsudin, J. S. Teh, and M. Almazrooie, "An integrated architecture for maintaining security in cloud computing based on blockchain," *IEEE Access*, vol. 9, pp. 69 513–69 526, 2021.

[12] M. Ibtihal, N. Hassan *et al.*, "Homomorphic encryption as a service for outsourced images in mobile cloud computing environment," in *Cryptography: Breakthroughs in Research and Practice*. IGI Global, 2020, pp. 316–330.

[13] P. Awasthi, S. Mittal, S. Mukherjee, and T. Limbasiya, "A protected cloud computation algorithm using homomorphic encryption for preserving data integrity," in *Recent Findings in Intelligent Computing Techniques*. Springer, 2019, pp. 509–517.

[14] R. Awadallah and A. Samsudin, "Homomorphic encryption for cloud computing and its challenges," in *2020 IEEE 7th International Conference on Engineering Technologies and Applied Sciences (ICETAS)*. IEEE, 2020, pp. 1–6.

[15] Y. Deswarte, J.-J. Quisquater, and A. Saïdane, "Remote integrity checking," in *Working conference on integrity and internal control in information systems*. Springer, 2003, pp. 1–11.

[16] D. L. Gazzoni Filho and P. S. L. M. Barreto, "Demonstrating data possession and uncheatable data transfer." *IACR Cryptol. ePrint Arch.*, vol. 2006, p. 150, 2006.

[17] G. Ateniese, S. Kamara, and J. Katz, "Proofs of storage from homomorphic identification protocols," in *International conference on the theory and application of cryptology and information security*. Springer, 2009, pp. 319–333.

[18] C. Wang, S. S. Chow, Q. Wang, K. Ren, and W. Lou, "Privacy-preserving public auditing for secure cloud storage," *IEEE Computer Architecture Letters*, vol. 62, no. 02, pp. 362–375, 2013.

[19] L. Li, Y. Yang, and Z. Wu, "Fmr-pdp: flexible multiple-replica provable data possession in cloud storage," in *2017 IEEE Symposium on Computers and Communications (ISCC)*. IEEE, 2017, pp. 1115–1121.

[20] J. Ni, K. Zhang, Y. Yu, and T. Yang, "Identity-based provable data possession from rsa assumption for secure cloud storage," *IEEE Transactions on Dependable and Secure Computing*, 2020.

[21] J. Zhang, B. Wang, M. R. Ogiela, X. A. Wang, and A. K. Sangaiah, "New public auditing protocol based on homomorphic tags for secure cloud storage," *Concurrency and Computation: Practice and Experience*, vol. 32, no. 18, p. e5600, 2020.

[22] S. Hiremath and R. S. Kunte, "Homomorphic authentication scheme for proof of retrievability with public verifiability," in *2020 4th International Conference on Intelligent Computing and Control Systems (ICICCS)*. IEEE, 2020, pp. 1017–1022.

[23] D. Vasilopoulos, M. Önen, K. Elkhiyaoui, and R. Molva, "Message-locked proofs of retrievability with secure deduplication," in *Proceedings of the 2016 ACM on Cloud Computing Security Workshop*, 2016, pp. 73–83.

[24] C. B. Tan, M. H. A. Hijazi, Y. Lim, and A. Gani, "A survey on proof of retrievability for cloud data integrity and availability: Cloud storage state-of-the-art, issues, solutions and future trends," *Journal of Network and Computer Applications*, vol. 110, pp. 75–86, 2018.

[25] J. Yuan and S. Yu, "Proofs of retrievability with public verifiability and constant communication cost in cloud," in *Proceedings of the 2013 international workshop on Security in cloud computing*, 2013, pp. 19–26.

[26] E. Shi, E. Stefanov, and C. Papamanthou, "Practical dynamic proofs of retrievability," in *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, 2013, pp. 325–336.

[27] K. Omote and T. P. Thao, "Md-por: multisource and direct repair for network coding-based proof of retrievability," *International Journal of Distributed Sensor Networks*, vol. 11, no. 6, p. 586720, 2015.

[28] Y. Zhu, H. Wang, Z. Hu, G.-J. Ahn, H. Hu, and S. S. Yau, "Efficient provable data possession for hybrid clouds," in *Proceedings of the 17th ACM conference on Computer and communications security*, 2010, pp. 756–758.

[29] Z. Hao and N. Yu, "A multiple-replica remote data possession checking protocol with public verifiability," in *2010 second international symposium on data, privacy, and E-commerce*. IEEE, 2010, pp. 84–89.

[30] C. Wang, Q. Wang, K. Ren, N. Cao, and W. Lou, "Toward secure and dependable storage services in cloud computing," *IEEE transactions on Services Computing*, vol. 5, no. 2, pp. 220–232, 2011.

[31] B. Rakesh, K. Lalitha, M. Ismail, and H. P. Sultana, "Distributed scheme to authenticate data storage security in cloud computing."

[32] R. Saxena and S. Dey, "Cloud audit: A data integrity verification approach for cloud computing," *Procedia Computer Science*, vol. 89, pp. 142–151, 2016.

[33] Q. Wang, C. Wang, K. Ren, W. Lou, and J. Li, "Enabling public auditability and data dynamics for storage security in cloud computing," *IEEE transactions on parallel and distributed systems*, vol. 22, no. 5, pp. 847–859, 2010.

[34] C. Wang, Q. Wang, K. Ren, and W. Lou, "Privacy-preserving public auditing for data storage security in cloud computing," in *2010 proceedings ieee infocom*. Ieee, 2010, pp. 1–9.

[35] S. E. Arasu, B. Gowri, and S. Ananthi, "Privacy-preserving public auditing in cloud using hmac algorithm," *International Journal of Recent Technology and Engineering*, vol. 2, no. 1, pp. 149–152, 2013.

[36] M. Kolhar, M. M. Abu-Alhaj, and S. M. Abd El-atty, "Cloud data auditing techniques with a focus on privacy and security," *IEEE Security & Privacy*, vol. 15, no. 1, pp. 42–51, 2017.

[37] S. H. Abbdal, H. Jin, A. A. Yassin, Z. A. Abduljabbar, M. A. Hussain, Z. A. Hussien, and D. Zou, "An efficient public verifiability and data integrity using multiple tpas in cloud data storage," in *2016 IEEE 2nd International Conference on Big Data Security on Cloud (BigDataSecurity), IEEE International Conference on High Performance and Smart Computing (HPSC), and IEEE International Conference on Intelligent Data and Security (IDS)*. IEEE, 2016, pp. 412–417.

[38] A. Razaque and S. S. Rizvi, "Triangular data privacy-preserving model for authenticating all key stakeholders in a cloud environment," *Computers & Security*, vol. 62, pp. 328–347, 2016.

[39] C. Asmuth and J. Bloom, "A modular approach to key safeguarding," *IEEE transactions on information theory*, vol. 29, no. 2, pp. 208–210, 1983.

[40] M. Backes, D. Fiore, and R. M. Reischuk, "Verifiable delegation of computation on outsourced data," in *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, 2013, pp. 863–874.

[41] X. Chen, J. Li, J. Weng, J. Ma, and W. Lou, "Verifiable computation over large database with incremental updates," *IEEE transactions on Computers*, vol. 65, no. 10, pp. 3184–3195, 2015.

[42] S. Goldwasser, Y. T. Kalai, and G. N. Rothblum, "Delegating computation: interactive proofs for muggles," *Journal of the ACM (JACM)*, vol. 62, no. 4, pp. 1–64, 2015.

[43] J. Li, Y. K. Li, X. Chen, P. P. Lee, and W. Lou, "A hybrid cloud approach for secure authorized deduplication," *IEEE Transactions on Parallel and Distributed Systems*, vol. 26, no. 5, pp. 1206–1216, 2014.

[44] B. Parno, M. Raykova, and V. Vaikuntanathan, "How to delegate and verify in public: Verifiable computation from attribute-based encryption," in *Theory of Cryptography Conference*. Springer, 2012, pp. 422–439.

[45] P. Renjith and S. Sabitha, "Verifiable el-gamal re-encryption with authenticity in cloud," in *2013 Fourth International Conference on Computing, Communications and Networking Technologies (ICCCNT)*. IEEE, 2013, pp. 1–5.

[46] S. T. Setty, R. McPherson, A. J. Blumberg, and M. Walfish, "Making argument systems for outsourced computation practical (sometimes)." in *NDSS*, vol. 1, no. 9, 2012, p. 17.

[47] S. Setty, V. Vu, N. Panpalia, B. Braun, A. J. Blumberg, and M. Walfish, "Taking proof-based verified computation a few steps closer to practicality," in *Presented as part of the 21st {USENIX} Security Symposium ({USENIX} Security 12)*, 2012, pp. 253–268.

[48] Z. Wen, J. Luo, H. Chen, J. Meng, X. Li, and J. Li, "A verifiable data deduplication scheme in cloud computing," in *2014 International Conference on Intelligent Networking and Collaborative Systems*. IEEE, 2014, pp. 85–90.

[49] X. Yu, Z. Yan, and A. V. Vasilakos, "A survey of verifiable computation," *Mobile Networks and Applications*, vol. 22, no. 3, pp. 438–453, 2017.

[50] X. Yu, Z. Yan, and R. Zhang, "Verifiable outsourced computation over encrypted data," *Information Sciences*, vol. 479, pp. 372–385, 2019.

[51] D. Fiore, R. Gennaro, and V. Pastro, "Efficiently verifiable computation on encrypted data," in *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, 2014, pp. 844–855.

[52] R. Gennaro, C. Gentry, and B. Parno, "Non-interactive verifiable computing: Outsourcing computation to untrusted workers," in *Annual Cryptology Conference*. Springer, 2010, pp. 465–482.

[53] C. Gentry and S. Halevi, "Implementing gentry's fully-homomorphic encryption scheme," in *Annual international conference on the theory and applications of cryptographic techniques*. Springer, 2011, pp. 129–148.

[54] T. Li, J. Li, Z. Liu, P. Li, and C. Jia, "Differentially private naive bayes learning over multiple data sources," *Information Sciences*, vol. 444, pp. 89–104, 2018.

[55] Y. Ding, B. Han, H. Wang, and X. Li, "Ciphertext retrieval via attribute-based fhe in cloud computing," *Soft Computing*, vol. 22, no. 23, pp. 7753–7761, 2018.

[56] A. Marinho, L. Murta, C. Werner, V. Braganholo, S. M. S. d. Cruz, E. Ogasawara, and M. Mattoso, "Provmanager: a provenance management system for scientific workflows," *Concurrency and Computation: Practice and Experience*, vol. 24, no. 13, pp. 1513–1530, 2012.

[57] R. Canetti, B. Riva, and G. N. Rothblum, "Two 1-round protocols for delegation of computation." *IACR Cryptol. ePrint Arch.*, vol. 2011, p. 518, 2011.

[58] P. Chinnasamy and P. Deepalakshmi, "Improved key generation scheme of rsa (ikgsr) algorithm based on offline storage for cloud," in *Advances in big data and cloud computing*. Springer, 2018, pp. 341–350.

[59] M. Deryabin, M. Babenko, A. Nazarov, N. Kucherov, A. Karachevtsev, A. Glotov, and I. Vashchenko, "Protocol for secure and reliable data transmission in manet based on modular arithmetic," in *2019 International conference on Engineering and Telecommunication (EnT)*. IEEE, 2019, pp. 1–5.

[60] A. Tchernykh, U. Schwiegelsohn, E.-g. Talbi, and M. Babenko, "Towards understanding uncertainty in cloud computing with risks of confidentiality, integrity, and availability," *Journal of Computational Science*, vol. 36, p. 100581, 2019.

[61] M. Deryabin, N. Chervyakov, A. Tchernykh, M. Babenko, N. Kucherov, V. Miranda-López, and A. Avetisyan, "Secure verifiable secret short sharing scheme for multi-cloud storage," in *2018 International Conference on High Performance Computing & Simulation (HPCS)*. IEEE, 2018, pp. 700–706.

[62] N. Chervyakov, M. Babenko, A. Tchernykh, N. Kucherov, V. Miranda-López, and J. M. Cortés-Mendoza, "Ar-rrns: Configurable reliable

distributed data storage systems for internet of things to ensure security," *Future Generation Computer Systems*, vol. 92, pp. 1080–1092, 2019.

[63] B. Varghese and R. Buyya, "Next generation cloud computing: New trends and research directions," *Future Generation Computer Systems*, vol. 79, pp. 849–861, 2018.

[64] W. Wong and K. Blow, "Design and analysis of an all-optical processor for modular arithmetic," *Optics communications*, vol. 265, no. 2, pp. 425–433, 2006.

[65] V. T. Goh and M. U. Siddiqi, "Multiple error detection and correction based on redundant residue number systems," *IEEE Transactions on Communications*, vol. 56, no. 3, pp. 325–330, 2008.

[66] C.-H. Chang, A. S. Molahosseini, A. A. E. Zarandi, and T. F. Tay, "Residue number systems: A new paradigm to datapath optimization for low-power and high-performance digital signal processing applications," *IEEE circuits and systems magazine*, vol. 15, no. 4, pp. 26–44, 2015.

[67] V. Kuchukov, A. Nazarov, and I. Vashchenko, "Cloud-fog-edge computing model for video surveillance based on modular arithmetic," in *2020 IEEE Conference of Russian Young Researchers in Electrical and Electronic Engineering (EIConRus)*. IEEE, 2020, pp. 374–376.

[68] A. Acar, H. Aksu, A. S. Uluagac, and M. Conti, "A survey on homomorphic encryption schemes: Theory and implementation. corr abs/1704.03578 (2017)," *arXiv preprint arXiv:1704.03578*, 2017.

[69] R. L. Rivest, L. Adleman, and M. L. Dertouzos, "On data banks and privacy homomorphisms," *Foundations of secure computation*, vol. 4, no. 11, pp. 169–180, 1978.

[70] T. ElGamal, "A public key cryptosystem and a signature scheme based on discrete logarithms," *IEEE transactions on information theory*, vol. 31, no. 4, pp. 469–472, 1985.

[71] J. Benaloh, "Dense probabilistic encryption," in *Proceedings of the workshop on selected areas of cryptography*, 1994, pp. 120–128.

[72] 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.

[73] T. Okamoto and S. Uchiyama, "A new public-key cryptosystem as secure as factoring," in *International conference on the theory and applications of cryptographic techniques*. Springer, 1998, pp. 308–318.

[74] D. Boneh, E.-J. Goh, and K. Nissim, "Evaluating 2-dnf formulas on ciphertexts," in *Theory of cryptography conference*. Springer, 2005, pp. 325–341.

[75] S. v. d. Walt, S. C. Colbert, and G. Varoquaux, "The numpy array: a structure for efficient numerical computation," *Computing in science & engineering*, vol. 13, no. 2, pp. 22–30, 2011.

[76] Amazon, "Amazon ec2 instance types," https://aws.amazon.com/ec2/instance-types/#instance-details, 2019, accessed: 2020-04-17.

[77] L. Bruno, "Azure cloud computing user guide," vol. 53, pp. 1689—-1699, 2019.

[78] Google, "Google cloud compute products," https://cloud.google.com/compute/docs/disks/?authuser=3&hl=ru#localssds, 2019, accessed: 2020-07-28.

[79] E. Barker, W. Burr, A. Jones, T. Polk, S. Rose, M. Smid, and Q. Dang, "Recommendation for key management part 3: Application-specific key management guidance," *NIST special publication*, vol. 800, p. 57, 2009.

[80] S. Trudel and I. Order, "International standard iso/iec information technology—process assessment—requirements for performing process assessment," 2015.