ExpSOS: Secure and Verifiable Outsourcing of Exponentiation Operations for Mobile Cloud Computing

Kai Zhou, Student Member, IEEE, M. H. Afifi, and Jian Ren, Senior Member, IEEE

Abstract—Discrete exponential operation, such as modular exponentiation and scalar multiplication on elliptic curves, is a basic operation of many public-key cryptosystems. However, the exponential operations are considered prohibitively expensive for resource-constrained mobile devices. In this paper, we address the problem of secure outsourcing of exponentiation operations to one single untrusted server. Our proposed secure outsourcing scheme for general exponential (ExpSOS) only requires a very limited number of modular multiplications at local mobile environment, and thus it can achieve significant computational performance gain. ExpSOS also provides a secure verification scheme with probability approximately 1 to ensure that the mobile end users can always receive valid results. The comprehensive analysis as well as the simulation results in real mobile device demonstrates that our proposed ExpSOS can significantly improve the existing schemes in efficiency, security, and result verifiability. We apply ExpSOS to securely outsource several cryptographic protocols to show that ExpSOS can be widely applied to many computation-intensive applications and achieve significant performance improvement.

Index Terms—Mobile cloud computing, secure outsourcing, modular exponentiation, scalar multiplication, result verification.

I. INTRODUCTION

CLOUD computing provides end-users the capability to securely access the shared pool of resources such as computational power and storage. It enables end-users to utilize those resources in a pay-per-use manner. Among all types of computations, exponential operation in a finite group is almost ubiquitous in public-key cryptosystems. However, due to large integers involved, exponentiation operation is considered prohibitively expensive for resource-constrained devices such as mobile phones. Thus, outsourcing exponentiation operation to the cloud servers becomes an appealing choice.

However, when sensitive data is outsourced to the untrusted cloud, security of the data as well as the result is at risk. Moreover, many cryptographic applications, such as digital signatures, require to verify the validity of the results of modular exponentiation. Thus result verification is also a crucial issue. In contrast, the cloud cannot be fully trusted for at least three reasons. First, the cloud could be curious. That is, it may try to “mine” as much information as possible from the outsourced data. Second, the computational resource is commodity. The cloud has the motivation to cheat in the computation process in order to save computational resources to support more business. Third, the cloud is a shared environment. It is hard to secure individual data using just regular data processors. Thus, security and verifiability are two major concerns for computation outsourcing.

To address these two issues, various computation outsourcing mechanisms have been proposed for various computational problems. Atallah et al. [1], Atallah and Frikken [2], Wang et al. [3], Zhou and Ren [4], and Liu et al. [5] focused on secure outsourcing of scientific computations such as matrix multiplications, solving system of linear equations and floating point computation. Other works [6]–[13] focused more on upper-layer algorithms such as association rule mining, sequence comparison, feature extraction of images and searching over encrypted data. The outsourcing of cryptographic computations [14]–[19] has been a popular research topic among the community. Especially, outsourcing of the modular exponentiation has been extensively studied due to its significance in cryptographic computations. In [17], the authors considered outsourcing of modular exponentiation to two servers assuming that they would not collude. The basic idea of the proposed scheme in [17] is to split the base and exponent of the modular exponentiation into two random looking pieces and then separately outsource them to two servers. Then the end-user can combine the results returned by the servers to recover the desired result. Under this scheme, the end-user can check the validity of the returned results with probability $\frac{1}{2}$. Following [17], Chen et al. [14] proposed a similar scheme and improved the performance by reducing one query to the servers and increasing the verifiability to $\frac{2}{3}$. In order to eliminate the assumption that the two servers would not collude, Wang et al. [20] proposed a scheme to outsource modular exponentiation to one single server. However, at local side, the end-user still needs to carry out one modular exponentiation $\chi^2$, where $\chi$ is a security parameter. As a result, the computational gain is limited for the end-user. Moreover, all these three schemes rely on pre-computation of modular exponentiation of some random integers. This will cause extra overhead to the end-user’s limited computational...
power or storage space depending on the method by which pre-computation is implemented. In our recent work [21], we proposed an efficient scheme to securely outsource scalar multiplication on elliptic curves. However, it did not provide algorithms for result verification.

The existing research in this area can be classified into three categories based on the model of the cloud \( S \): the Honest-but-Curious Single-server (HCS) model [22], the Malicious Multiple-servers (MM) model [14], [17], [23] and Malicious Single-server (MS) model [20], [24]. In particular, in the honest but curious model, the cloud will honestly fulfill its advertised functionality. However, \( S \) could be curious. It may try to exploit any key information from the outsourced task, which may include the input, the output as well as the intermediate computational results. When the outsourced data is sensitive, this could cause severe security and privacy issues. In the malicious model, the cloud \( S \) may not carry out the desired computation truthfully. This can happen for various reasons. A simple scenario could be that the cloud simply returns some trivial results since the computational resource is a commodity for the cloud server. As a result, the end-user \( E \) is unable to receive a valid result from the cloud server \( S \).

The models are hierarchical in the sense that a secure outsourcing scheme designed for single-server model can be extended to multiple-server model and a scheme for malicious cloud can be extended to honest but curious cloud. Specifically, these models can be organized into three layers: at the bottom layer is the HCM (Honest-but-Curious Multiple-servers) model, in the middle are the MM and HCS and on the top is MS. A secure outsourcing scheme designed for a model in an upper layer is also suitable for that in a lower layer. Thus, a secure outsourcing scheme for MS is most widely applicable and achieves the highest security standard. In this paper, we first propose a secure outsourcing scheme for the HCS model. Then a verification scheme is proposed for the MS model.

Apparently, it is much more desirable and secure to outsource exponentiation operations to one single server instead of multiple servers with security based on the assumption that the servers would not collude. The secure outsourcing scheme should not impose expensive computational overhead at local side. Otherwise, the performance gain from outsourcing would diminish. The scheme should also provide high verifiability. Ideally, the end-user should be able to verify the validity of the returned result with probability 1.

In this paper, we extend the notion of modular exponentiation to general exponential operations in a finite group, including scalar multiplication on elliptic curves. In general, each exponential operation consists of a series of basic group operations. The number of such operations varies with the exponent. In this sense, modular exponentiation and scalar multiplication can both be regarded as exponentiation operations. Thus, we propose a Secure Outsourcing Scheme for general Exponential (ExpSOS) operations. The proposed ExpSOS is based on a secure disguising procedure that maps the integers in the group \( \mathbb{R}_N \) to the larger group \( \mathbb{R}_L \) so that the cloud will carry out the computation in \( \mathbb{R}_L \) while still keeps \( N \) secure. From the result returned by the cloud, the end-user can recover the result back to \( \mathbb{R}_N \) efficiently.

The main contributions of this paper can be summarized as follows:

- We formally define a secure outsourcing scheme and four outsourcing models. The proposed ExpSOS is shown to be effective under all four different models.
- We develop schemes to securely outsource exponentiation operations in a general finite group, including modular exponentiation and scalar multiplication on elliptic curves.
- We outsource exponential operations to one single untrusted server eliminating the non-collusion assumption between multiple servers.
- Our proposed ExpSOS is efficient in that it requires only a small number of modular multiplications at local side.
- We propose a verification scheme such that the end-user can verify the validity of the result with probability approximately 1.

The rest of this paper is organized as follows. In Section II, we introduce our assumptions and formal definitions of secure computation outsourcing. In Section III, we present the design of ExpSOS for both modular exponentiation and scalar multiplication. We propose the verification scheme in Section IV. The security and complexity analysis of ExpSOS are given in Section V. Then we apply ExpSOS to outsource several cryptographic protocols in Section VI. In Section VII, we compare the performance of ExpSOS with several existing works and give some numeric results. We conclude in Section VIII.

II. SECURE COMPUTATION OUTSOURCING MODEL

A. System Model and Threat Model

In the general settings of computation outsourcing, the system consists of two entities: an end-user \( E \) and the cloud \( S \). The end-user \( E \) is resource-constrained. It has limited computational power and storage space. The cloud \( S \) is regarded as possessing abundant resources and is able to carry out computational intensive operations. The cloud can be further modeled as the single-server model and the multiple-server model. In the single-server model, the cloud is viewed as one unit. In contrast, in the multiple-server model, the cloud is divided into two or more distinct units. Each unit carries out the computational tasks independently. While communication between different units is allowed, key information is only limited to individual unit since otherwise security of the whole system maybe in jeopardy.

In addition, the cloud \( S \) can be either honest (however, could be curious) or malicious. Our scheme can be applied to all four threat models combined from these two assumptions.

Suppose the end-user \( E \) wishes to accomplish a computationally intensive task \( F(x) = \omega \), where \( x \) is the input and \( \omega \) is the output of the task. However, due to the limited resources, \( E \) may not be able to finish the task using the locally available resources. The computational task \( F \) could be outsourced to \( S \). Unfortunately, the cloud is only a shared server and cannot be fully trusted. Therefore, we have to make sure that it is...
B. Definition of Secure Outsourcing Scheme

We follow the security model defined in [17], which is also employed in works [14], [20], [23], and [24]. In [17], the authors consider splitting a cryptographic algorithm $\text{Alg}$ to a trusted component $T$ and an untrusted component $C$ that $T$ can make queries to. It is also proposed that an adversary $A$ consists of two parts: the adversarial environment $E$ that submits adversarially chosen inputs to $\text{Alg}$ and a malicious component $C'$ that operates in place of $C$. Based on this model, the algorithm with outsource-IO is defined in [17] as an algorithm $\text{Alg}(x_{hs}, y_{hp}, x_{hu}, x_{ap}, x_{au}) \rightarrow (y_{s}, y_{p}, y_{u})$ that takes in five inputs and produces three outputs, where $x_{hs}$ is the honest secret input which is honestly generated and is only known to $T$; $y_{hp}$ is the honest and protected input that is known to both $T$ and $E$, but not to $C$; $x_{hu}$ is the honest and unprotected input that is known to $T$, $E$ and $C$; $x_{ap}$ is the adversarial protected input that is adversarially generated and is known to both $T$ and $E$, but not to $C$; $x_{au}$ is the adversarial input that is known to $T$, $E$ and $C$. $y_{s}$ is the secret output that is only known to $T$; $y_{p}$ is the protected output that is known to both $T$ and $E$, but not to $C$; $y_{u}$ is the unprotected output that is known to $T$, $E$ and $C$.

Based on this outsource-IO, the notion of outsource security is further defined in [17]. The security definition requires that for any adversary $A = (E, C')$, there exists a pair of polynomial-time simulators $(S_1, S_2)$ that can simulate the views of $E$ and $C'$, respectively. Moreover, the two pairs of views are computationally indistinguishable. In the following, we propose the formal security definition for our outsourcing model. In particular, our definition can be considered as a tailored version of that in [17]. This is because we do not explicitly consider an adversary environment $E$ since there exist no adversarially generated inputs in our settings. Accordingly, we define an algorithm with outsource-IO as follows.

Definition 1 (Algorithm With Outsource-IO): An algorithm $\text{Alg}$ obeys the outsource input/output specification if it takes two inputs and produces two outputs, i.e., $\text{Alg}(x_p, x_u) \rightarrow (y_p, y_u)$. Especially,
- $x_p$ is the protected input known only to $T$.
- $x_u$ is the unprotected input, which is known to $T$ and $C$.
- $y_p$ is the protected output known only to $T$.
- $y_u$ is the unprotected output, which is known to $T$ and $C$.

Definition 2 (Outsource-Security): Let $\text{Alg}$ be an algorithm with outsource-IO. A pair of algorithms $(T, C)$ is said to be an outsource-secure implementation of $\text{Alg}$ if:

Correctness $T^C$ is a correct implementation of $\text{Alg}$.

Security For any probabilistic polynomial-time adversary $C'$, there exists probabilistic expected polynomial-time simulator $S$ such that the following pair of random variables is computationally indistinguishable.

$\text{VIEW}_{\text{real}} \sim \text{VIEW}_{\text{ideal}}$ (The adversary $C'$ learns nothing.)

- The view that the adversary $C'$ obtains by participating in the following REAL process:

$\text{VIEW}_{\text{real}} = \{(\text{istate}, x_p, x_u) \leftarrow I(1^\alpha);$
\hspace{1cm}$(\text{tstate}, \text{cstate}, y_p, y_u) \leftarrow T^C(x_p, x_u) : \text{cstate}\}.$

In this real process, the state variable (e.g., $\text{istate}$, $\text{tstate}$ and $\text{cstate}$) keeps records of the view of the corresponding party. First, the protected input $x_p$ and unprotected input $x_u$ are picked using an honest, stateful process $I$ to which the adversary does not have access. Next, the algorithm $T^C$ is run on the inputs $(x_p, x_u)$ and produces output $(y_p, y_u)$ as well as $\text{tstate}$ and $\text{cstate}$ for $T$ and $C'$, respectively. The view of $C'$ in the real process is $\text{cstate}$.

- The IDEAL process:

$\text{VIEW}_{\text{ideal}} = \{(\text{istate}, x_p, x_u) \leftarrow I(1^\alpha);$
\hspace{1cm}$(\text{astate}, y_p, y_u) \leftarrow \text{Alg}(x_p, x_u);$  
\hspace{1cm}$(\text{sstate}, \text{cstate}) \leftarrow S^C(x_u) : \text{cstate}\}.$

In the ideal process, a simulator $S$ is utilized to simulate the view of $C'$. Note that $S$ is given only the unprotected input and can make queries to the adversary $C'$.

Definition 3 (a-Efficient): Suppose the running time for a task $F$ to be processed locally by $E$ is $t_0$. Under a Secure Outsourcing Scheme (SOS), the running time of local processing for $E$ is $t_p$. Then the SOS is $a$-efficient if $\frac{t_p}{t_0} \geq a$.

Definition 4 (β-Verifiable): Given the returned output $\Omega$ and the proof $\Phi$, denote the probability that $E$ is able to verify the validity of the result $\omega$ as $\kappa$. Then an SOS is $\beta$-verifiable if $\kappa \geq \beta$.

From the definition above, we can see that a larger $\alpha$ indicates a better performance of a secure outsourcing scheme, while a larger $\beta$ means a better verifiability.

III. ExpSOS: SECURE OUTSOURCING OF EXPONENTIATION OPERATIONS

A. General Framework

The general framework of an SOS consists of four different functions $(T, C, R, V)$.

1) Problem Transformation $T : F(x) \rightarrow G(y)$. The end-user $U$ locally transforms the problem $F(x)$ to a new form $G(y)$, where $y$ is the new input and $G$ is the new problem description. $E$ then outsources $G(y)$ to the cloud server $S$.

2) Cloud Computation $C : G(y) \rightarrow (\Omega, \Phi)$. The cloud $S$ solves the transformed problem $G(y)$ to obtain the corresponding result $\Omega$. At the same time, $S$ returns a proof $\Phi$ of the validity of the result.

3) Result Recovery $R : \Omega \rightarrow \omega$. Based on the returned result $\Omega$, the end-user $U$ recovers the result $\omega$ of the original problem $F(x)$.

4) Result Verification $V : (\Omega, \Phi, \omega) \rightarrow (\text{True} \text{ or False})$. Based on $\omega$, $\Omega$ and the proof $\Phi$, the end-user $U$ verifies the validity of the result.
B. Secure Disguising Procedure

In this paper, we assume that for the end-user, exponentiation operations are operated in the integer ring modular $N$, denoted as $\mathbb{Z}_N$. We note that $N$ is not necessarily a prime number. It can also be a product of large prime numbers. To outsource modular exponentiation to the shared cloud, we first need to conceal the modular $N$. To achieve this, we multiply $N$ by a randomly selected large prime $p$ and define $L = pN$. We define a secure disguising procedure to map $x \in \mathbb{Z}_N$ to $y \in \mathbb{Z}_L$ as follows:

1) Select a random $k$, $1 \leq k \leq p - 1$.
2) Compute $y = x + kN \pmod{L}$.

Without the knowledge of $k$, it is hard to determine which point $x$ is mapped to. To recover $x$ from $y$, the end-user only needs to compute $y = (x + kN) = x \pmod{N}$. Therefore, regardless of which $k$ is selected to outsource $x$, we will always have $y = x \pmod{N}$.

Now, we explore the properties of the computation outsourcing and recovery functions. They are key to our proposed secure outsourcing scheme.

**Theorem 1:** For any $x_1, x_2 \in \mathbb{Z}_N$ and their corresponding disguised form $y_1 = x_1 + k_1N$ and $y_2 = x_2 + k_2N$, where $k_1$ and $k_2$ are randomly selected integers, $1 \leq k_1, k_2 \leq p - 1$, we have

$$x_1 + x_2 = (y_1 + y_2) \pmod{N},$$
$$x_1 x_2 = (y_1 y_2) \pmod{N}.$$  

**Proof:** We can verify that

$$(y_1 + y_2) \pmod{N} = ((x_1 + k_1N) + (x_2 + k_2N)) \pmod{L} \pmod{N} = (x_1 + k_1N + x_2 + k_2N) \pmod{N} = (x_1 + x_2) \pmod{N}.$$  

Similarly, it can be verified that $x_1 x_2 = y_1 y_2 \pmod{N}$.  

**Corollary 1:** Suppose $x = (x_1, x_2, \ldots, x_n) \in \mathbb{Z}_N^n$ and $y = (y_1, y_2, \ldots, y_n) = (x_1 + k_1N, x_2 + k_2N, \ldots, x_n + k_nN)$ and let $\phi: \mathbb{R}_n \rightarrow \mathbb{R}$ be an $n$-variable polynomial function with coefficients in $\mathbb{R}$, where $\mathbb{R}$ can be $\mathbb{Z}_N$ or $\mathbb{Z}_L$. Then we have

$$\phi(x) = \phi(y) \pmod{N}.$$  

Theorem 1 enables us to transform the addition and multiplication in a ring $\mathbb{Z}_N$ into the corresponding operations in another large ring $\mathbb{Z}_L$. Since polynomial evaluation consists of addition and multiplication, Corollary 1 states that we can transform polynomial evaluation in $\mathbb{Z}_N$ into corresponding operations in $\mathbb{Z}_L$.

C. Secure Outsourcing of Modular Exponentiation Under HCS Model

1) Conceal the Base in Modular Exponentiation Outsourcing: Consider modular exponentiation $R = u^a \pmod{N}$. We assume that $N$ is either a large prime or a product of large prime numbers, which is the typical situation in cryptosystems.

**Theorem 1** states that the result of multiplication in the ring $\mathbb{Z}_N$ can be obtained from the multiplication in $\mathbb{Z}_L$ through the transformation function and the inverse function. If we take $x_1 = x_2 = u$, we can get

$$((u + rN) \pmod{L})^2 = u^2 \pmod{N}.$$  

If we repeat the multiplication in $\mathbb{Z}_N$ for a times, we have the following corollary.

**Corollary 2:** For $u, a, r \in \mathbb{Z}_N$, we have

$$(u + rN) \pmod{L} \equiv u^a \pmod{N}.$$  

Corollary 2 gives us a way to conceal the base when outsourcing modular exponentiation. That is, we can first transform the original base $u$ to $U = (u + rN) \pmod{L}$, where $r \in \mathbb{Z}_N$ is a random integer. Then the cloud can compute $U^a \pmod{L}$ based on which the original result can be recovered by computing $U^a = u^a \pmod{N}$.

2) Conceal the Exponent in Modular Exponentiation Outsourcing: The remaining task is to conceal the exponent $a$. We have the following theorem.

**Theorem 2:** For $N = p_1p_2 \cdots p_m$, where $p_1, p_2, \ldots, p_m$ are distinct prime numbers, we have

$$u^{a+k\varphi(N)} = u^a \pmod{N},$$  

where $k$ is a random integer and $\varphi(\cdot)$ is the Euler’s totient function.

**Proof:** We first prove that $u^{1+k\varphi(N)} = u \pmod{N}$. For each prime factor $p_i$ of $N$, $i = 1, 2, \ldots, m$. There are two possible cases:

- Case 1: $\gcd(u, p_i) \neq 1$, that is $u$ and $p_i$ are not relatively prime. Since $p_i$ is prime, we have $p_i \mid u$. Thus $u^{1+k\varphi(N)} - u = 0 \pmod{p_i}$, which means that $p_i \mid (u^{1+k\varphi(N)} - u)$.
- Case 2: $\gcd(u, p_i) = 1$, that is $u$ and $p_i$ are relatively prime. Then, by the Euler’s Theorem, we have $u^{\varphi(p_i)} = 1 \pmod{p_i}$. From the multiplicative property of the Euler’s totient function, we have $\varphi(N) = \varphi(p_1)\varphi(p_2) \cdots \varphi(p_m)$. Let $\theta(p_i) = \varphi(N)/\varphi(p_i)$. Then,

$$u^{1+k\varphi(N)} \pmod{p_i} = u \cdot u^{k\varphi(p_i)\varphi(p_2) \cdots \varphi(p_m)} \pmod{p_i} = u \cdot (u^{\theta(p_i)})^{k\theta(p_i)} \pmod{p_i} = u \cdot (1)^{k\theta(p_i)} \pmod{p_i} = u \pmod{p_i}.$$  

That is $(u^{1+k\varphi(N)} - u) = 0 \pmod{p_i}$. Therefore, in both cases, we have proved that $p_i \mid (u^{1+k\varphi(N)} - u)$. Since $p_i$ is arbitrarily selected and $p_1, p_2, \ldots, p_m$ are distinct prime numbers, we have

$$N \mid (u^{1+k\varphi(N)} - u).$$  

Hence, $u^{1+k\varphi(N)} = u \pmod{N}$. Multiplying both sides of the equation by $u^{a-1}$, we can obtain

$$u^{a+k\varphi(N)} = u^a \pmod{N}.$$  

In Theorem 2, we do not require that $u$ and $N$ to be co-prime as required in the Euler’s theorem. Instead,
we assume that \( N \) is the product of distinct prime numbers. For instance, in RSA, the modulus \( N = pq \) is the product of two distinct prime numbers.

Theorem 2 introduces a way to conceal the exponent \( a \). That is, by transforming the original exponent \( a \) to \( A = a + k\phi(N) \), where \( k \) is a random integer, we can conceal \( a \) due to the randomness of \( k \). Now, based on Theorem 1 and Theorem 2, we can construct our secure outsourcing scheme for modular exponentiation. In the secure outsourcing scheme, we utilize a function \( C(U, A, L) \) to denote the computation of a modular exponentiation for the cloud as \( C(U, A, L) = U^A \mod L \). The result recovery function is \( R(R, N) = R \mod N \).


**Protocol 1 Secure Outsourcing of Modular Exponentiation Under HCS Model**

**Input:** \( N, u, a \in \mathbb{R}_N \).

**Output:** \( R = u^a \mod N \).

**Key Generation KeyGen(1^k, N) \rightarrow (p, L):**

1. \( E \) generates a large prime \( p \) and calculates \( L \leftarrow pN \).
2. The public key is \( K_p = (L) \), and the private key is \( K_s = \{p, N\} \).

**Problem Transformation T(a, u) \rightarrow (A, U):**

1. \( E \) selects random integers \( r, k \in \mathbb{R}_N \) as the temporary key.
2. \( E \) calculates \( A \leftarrow a + k\phi(N), U \leftarrow (u + rN) \mod L \).
3. \( E \) outsources \((U, A, L)\) to the cloud.

**Cloud Computation C(A, U, L) \rightarrow R1:**

1. \( S \) computes \( R_1 \leftarrow C(U, A, L) = U^A \mod L \).
2. \( S \) returns \( R_1 \) to \( E \).

**Result Recovery R(R1, N) \rightarrow R:**

1. \( E \) recovers the result as \( R \leftarrow R(R_1, N) = R \mod N \).

The soundness of the outsourcing scheme is guaranteed by the following theorem:

**Theorem 3:** The secure outsourcing scheme for modular exponentiation is sound. That is \( R = R_1 = u^a \mod N \).

The proof of Theorem 3 is straightforward based on Theorem 1 and Theorem 2. Specifically, by transforming the original problem of modular exponentiation to a disguised form, our proposed ExpSOS under HCS model is sound.

**D. Secure Outsourcing of Scalar Multiplication Under HCS Model**

In this section, we consider secure outsourcing of scalar multiplication \( sP \) on an elliptic curve \( E(\mathbb{F}_p) \) described by the following short Weierstrass equation:

\[
E : \ y^2 = x^3 + bx + c,
\]

where the coefficients \( b, c \) and the coordinates of the points are all in a finite field \( \mathbb{F}_p \). Furthermore, for cryptographic applications, we usually work with points in a set of \( m \)-torsion points \( E(\mathbb{F}_p)[m] \) defined as \( E(\mathbb{F}_p)[m] = \{ P \in E(\mathbb{F}_p) : [m]P = \mathcal{O}\} \), where \( \mathcal{O} \) is the point at infinity. Thus, we assume \( P \in E(\mathbb{F}_p)[m] \) and \( s \in \mathbb{Z}_m \).

Secure outsourcing of scalar multiplication relies on two basic operations, point additions and point doublings. They play a similar role as modular multiplication in the outsourcing of modular exponentiation. Specifically, the “double-and-add” algorithm to calculate scalar multiplication on elliptic curves consists of a series of point addition and point doubling. Thus intuitively, we can regard secure outsourcing of point addition and point doubling as two building blocks to implement scalar multiplication.

We utilize projective coordinate to represent a point \( P = (x, y, z) \) corresponding to the point \( Q = (\frac{x}{z}, \frac{y}{z}) \) in the affine coordinates. As a result, the computation of point addition and point doubling consists of only modular addition and multiplication. Specifically, given two points \( P = (x_1, y_1, z_1) \) and \( Q = (x_2, y_2, z_2) \) such that \( P \neq \pm Q \), the point addition \( P + Q = (x_3, y_3, z_3) \) can be calculated as follows:

\[
x_3 = \sigma \tau, \quad y_3 = \rho(\sigma^2 y_1 z_2 - \tau^3 - z_1 y_1 z_2), \quad z_3 = \sigma^3 z_1 z_2,
\]

where \( \rho = y_2 z_1 - y_1 z_2, \quad \sigma = x_2 z_1 - x_1 z_2, \quad \tau = \rho^2 z_1 z_2 - \sigma^3 - 2\sigma^2 x_1 z_2 \).

The point doubling \( 2P = (x_4, y_4, z_4) \) can be calculated as follows:

\[
x_4 = 2\sigma \mu, \quad y_4 = \rho(4\tau - \mu) - 8y_1^2 \sigma^2, \quad z_4 = 8\sigma^3,
\]

where \( \rho = 3z_1^2 + 3x_1^2, \quad \sigma = y_1 z_1, \quad \tau = x_1 y_1 \sigma, \quad \mu = \rho^2 - 8\tau \).

In projective coordinates, one point addition and doubling take 14 multiplications and 12 multiplications, respectively.

Corollary 1 states that by mapping the variables of a polynomial from a finite field to variables in a ring, we can evaluate the polynomial in the ring and recover the result in the finite field. This gives us the insight of our proposed scheme since essentially, point addition and point doubling are both the process of evaluating polynomials on the coordinates of the points. Thus, we can construct the secure computation scheme for point addition and point doubling as in Protocol 2.

**Theorem 4:** The proposed secure point addition and point doubling protocol is sound.

The proof of Theorem 4 is straightforward from the Corollary 1.

The above theorem enables us to conceal the points as well as the parameters of the elliptic curve from the cloud. To outsource scalar multiplication \( sP \), the remaining part is to conceal the multiplier \( s \). We utilize the property of the order \( m \) of the torsion group that is \( rmP = \mathcal{O} \), for an arbitrary point \( P \in E[m](\mathbb{F}_p) \) and any integer \( r \). As a result, we can conceal \( s \) by adding it to a multiple of \( m \) as \( s' = s + rm \), where \( r \) is a random integer. Now, we can summarize the secure outsourcing scheme of scalar multiplication as in Protocol 3.
Protocol 2 Secure Point Addition and Point Doubling

Input: \( E = \{b, c, p\} \) and \( P = (x_1, y_1, z_1) \), \( Q = (x_2, y_2, z_2) \).

Output: point \( R = P + Q = (x_3, y_3, z_3) \).

1: Select a large prime \( q \) and compute \( N = pq \).
2: For a coordinate \( x_i \), select a random integer \( k_i \) and compute \( x'_i = (x_i + k_i \cdot p) \mod N \).
3: Transform the points \( P, Q \) and the elliptic curve \( E \) to \( P' = (x'_1, y'_1, z'_1) \), \( Q' = (x'_2, y'_2, z'_2) \) and \( E' = \{b', c', N\} \) respectively as described in Step 2.
4: Outsource \( P', Q' \) and \( E' \) to the cloud.
5: Cloud computes \( R' = P' + Q' \) following the point doubling or point addition procedure.
6: On receiving \( R' = (x'_3, y'_3, z'_3) \), recover \( R \) as \( R = (x'_3, y'_3, z'_3) = (x_3, y_3, z_3) \mod p \).

Protocol 3 Secure Outsourcing of Scalar Multiplication Under HCS Model

Input: \( E = \{b, c, p\} \), \( P = (x_1, y_1, z_1) \), \( s, m \)

Output: point \( R = sP \).

Key Generation KeyGen(\( 1^k, p \)) \( \rightarrow N \):

1: End-user selects a large prime \( q \) and compute \( N \leftarrow pq \).

Problem Transformation \( T(P, E) \rightarrow (P', E') \):

1: End-user generates random integers \( k_1, k_2, k_3, k_4, k_6, r \).
2: Computes \( x'_1 = (x_1 + k_1 \cdot p) \mod N \), \( y'_1 = (y_1 + k_2 \cdot p) \mod N \), \( z'_1 = (z_1 + k_3 \cdot p) \mod N \), \( b' = (b + k_4 \cdot p) \mod N \), \( c' = (c + k_6 \cdot p) \mod N \), \( s' \leftarrow s + rm \).
3: End-user outsources \( P' = (x'_1, y'_1, z'_1) \), \( E' = \{b', c', N\} \) and \( s' \).

Cloud Computation \( C(s', P') \rightarrow R' \):

1: The cloud computes \( R' = s'P' = (x'_3, y'_3, z'_3) \) utilizing the double-and-add algorithm.

Result Recovery \( R(R', p) \rightarrow R \):

1: The end-user recovers the result \( R \) as \( R \leftarrow R' = (x'_3, y'_3, z'_3) \mod p \).

Theorem 5: The secure outsourcing scheme for scalar multiplication is sound in \( \mathbb{F}_q \). That is \( R = sP \).

Proof: From Theorem 4, we know that the secure computation scheme for point addition and point doubling is sound. Since the double-and-add algorithm to compute scalar multiplication consists of a sequence of point addition and point doubling, we have \( R = s'P = (s + rm)P = sP + rmP = sP + P = sP \).

In the next section, we propose a verification scheme to ensure that ExpSOS is secure under the MS model.

IV. RESULT VERIFICATION

In this section, we first analyze the necessary properties of a result verification scheme through some counter examples. We then propose a result verification scheme for outsourcing of modular exponentiation under MS model. We show that the verification scheme can also be applied to outsourcing of scalar multiplication.

For the HCS model discussed in the previous section, we assume that the cloud will honestly conduct its advertised functionality. That is, to compute the function \( C(U, A, L) \) and return the correct result \( U^A \mod L \). However, in the MS model, the cloud may manipulate the result in order to save computational resources. Thus, to verify the soundness of the result returned by the cloud is a critical issue.

A natural way to verify the result, as utilized in many previous works [14], [15], [17], is to outsource the problem multiple times and verify whether the returned results satisfy certain criteria. However, this methodology may cause potential security problems if it is not carefully designed. This is because outsourcing multiple times essentially gives more information about the original problem to the cloud, which may increase the probability for the cloud to recover the original problem. Moreover, the cloud may manipulate the results in order to satisfy the criteria, thus passing the verification. Therefore, we believe that an effective verification scheme should at least have the following two properties:

- **Security**: The verification process should not reveal any key information about the original problem to the cloud.
- **Anti-manipulation**: It is infeasible for the cloud to manipulate the result and pass the verification process.

We utilize two counter-examples in verifying modular exponentiation to illustrate the significance of the above properties and emphasize the key issues in designing a verification scheme.

Counter-Example 1: Transform the exponent \( a \) to \( A_1 = a + k_1\phi(N) \) and \( A_2 = a + k_2\phi(N) \). The cloud returns results \( R_1 = U^{A_1} \mod L \) and \( R_2 = U^{A_2} \mod L \). The end-user checks whether the condition \( R_1 = R_2 \mod N \) holds.

Unfortunately, the above example violates the security property. When the cloud possesses \( A_1 \) and \( A_2 \), it can calculate \( A_1 - A_2 = (k_1 - k_2)\phi(N) \), which is a multiple of the Euler’s totient function \( \phi(N) \). In this case, the cloud can factorize \((k_1 - k_2)\phi(N)\) based on which, the cloud may be able to check the primality of \( N \). Since \( N \) is a product of large prime numbers, the consequence is that the cloud can limit the valid value of \( N \) to a short list. This means that the cloud can derive some key information from the outsourced problem thus making outsourcing insecure. Similarly, some variances of this type of method (e.g., \( A_1 = a + k_1\phi(N) \) and \( A_2 = ca + k_2\phi(N) \), where \( c \) is a known constant) may also have security problems.

Counter-Example 2: Transform the exponent \( a \) to \( A_1 = a + k_1\phi(N) \) and \( A_2 = a + t + k_2\phi(N) \), where \( t \) is a relatively small integer, which makes computing \( U^t \mod N \) within the end-user’s computational ability. The cloud returns results \( R_1 = U^{A_1} \mod L \) and \( R_2 = U^{A_2} \mod L \). The end-user checks whether the condition \( (R_1 \cdot U^t) = R_2 \mod N \) holds.

Due to the randomness of \( t \), the cloud is not able to obtain a multiple of \( \phi(N) \). However, from the equality condition \((R_1 \cdot U^t) = R_2 \mod N\), we have \( U^{A_1 \cdot U^t} = U^{A_2} \mod N \), which is equivalent to \( U^t = U^{A_2 - A_1} \mod N \).

In this case, the cloud can manipulate the two integers \( A_1' \) and \( A_2' \). As long as \( A_2' - A_1' = A_2 - A_1 \), the results will pass the verification but the recovered result \( R = U^{A_1'} \mod N \)
is incorrect. This means that the cloud can manipulate a false result while passing the verification process.

From the above two counter examples, we can see that security and anti-manipulation are two critical issues in result verification schemes. In the following Protocol 4, we propose a verification scheme for modular exponentiation.

**Protocol 4 ExpSOS Under MS Model**

**Input:** $N, u, a \in \mathbb{R}_n$.

**Output:** $R_0 = u^a \pmod{N}$, $\Lambda = \text{True} \text{ or False}$.

**Key Generation** $\text{KeyGen}(1^k, N) \rightarrow (p, L)$:

1. $E$ generates a large prime $p$ and calculate $L \leftarrow pN$.
2. The public key is $K_p = \{L\}$, and the private key is $K_s = \{p, N\}$.

**Problem Transformation** $T(a, u) \rightarrow (A_1, A_2, U)$:

1. $E$ selects random integers $r, k_1, k_2, t_1, t_2$ as the ephemeral key with the constraint that $t_1, t_2 \leq b$.
2. $E$ calculates $A_1 = a + k_1\phi(N)$, $A_2 = t_1a + t_2 + k_2\phi(N)$ and $U = (u + rN) \pmod{L}$.
3. $E$ outsources $(U, A_1, A_2, L)$ to the cloud.

**Cloud Computation** $C(A_1, A_2, U, L)$:

1. $S$ computes $R_1 \leftarrow U^{A_1} \pmod{L}$ and $R_2 \leftarrow U^{A_2} \pmod{L}$.
2. $S$ returns $R_1$ and $R_2$ to $E$.

**Result Verification** $V(R_1, R_2) \rightarrow \Lambda$:

1. $E$ checks whether $(R_1)^1 \cdot u^2 = R_2 \pmod{N}$.
2. If the equality holds, set $\Lambda \leftarrow \text{True}$. Otherwise, set $\Lambda \leftarrow \text{False}$.

**Result Recovery** $\mathcal{R}(R_1, N) \rightarrow R_0$:

1. $E$ recovers the result as $R_0 \leftarrow \mathcal{R}(R_1, N) = R_1 \pmod{N}$.

Now, we utilize an example to illustrate our proposed ExpSOS under MS model.

**Example 1:** Suppose the end-user $U$ wants to calculate $u^a \pmod{N}$, where $N = 431$ is a prime, $u = 189$ and $a = 346$. $E$ can outsource $u^a \pmod{N}$ as follows:

1. **Key Generation:** $E$ selects a prime number $p = 397$ and calculate $L = pN = 171107$. Then $E$ selects random integers $r = 146, k_1 = 332, k_2 = 68$ and $t_1 = 4, t_2 = 12$ with $t_1, t_2 < b = 16$.
2. **Problem Transformation:** $E$ calculates $A_1 = a + k_1\phi(N) = 143106$, $A_2 = t_1a + t_2 + k_2\phi(N) = 30636$ and $U = (u + rN) = 63115 \pmod{L}$. $E$ then queries $C(U, A_1, L)$ and $C(U, A_2, L)$ to the cloud $S$.
3. **Cloud computation:** $S$ computes $R_1 = U^{A_1} \pmod{L} = 63115^{143106} \pmod{171107} = 81281, R_2 = U^{A_2} \pmod{L} = 63115^{30636} \pmod{171107} = 55473$ and returns $R_1$ and $R_2$ to $E$.
4. **Result Verification:** $E$ calculates $R_1^1 \cdot u^2 \pmod{N} = (190^4 \cdot 189^{12}) \pmod{431} = 305$ and $R_2 \pmod{N} = 55473 \pmod{431} = 305$ that satisfy $R_1^1 \cdot u^2 = R_2 \pmod{N}$. Thus, the returned results are correct.
5. **Result Recovery:** $E$ recovers the result as $R = R_1 \pmod{N} = 81281 \pmod{431} = 190$ that is equal to $u^a = 190 \pmod{N}$.

In Protocol 4, the two outsourced exponential operations are related through an affine function. As a result, the cloud is unable to derive a multiple of $\phi(N)$ only based on $A_1$ and $A_2$. Moreover, the cloud cannot manipulate the results to create a verifiable equality.

This verification scheme can also be applied to outsourcing of scalar multiplications. The base point $P$ can be transformed to $P'$ as described in Protocol 3. The exponent $s$ can be transformed to $s_1 = s + r_1m$ and $s_2 = t_1s + t_2 + r_2m$, where $r_1, r_2, t_1, t_2$ are random integers and $t_1, t_2 \leq b$. Then the end-user can check the condition $Q_2 = t_1Q_1 + t_2P$, where $Q_1 = s_1P'$ and $Q_2 = s_2P'$.

**V. SECURITY AND COMPLEXITY ANALYSIS**

In this section, we analyze the security and the computational complexity of ExpSOS. We utilize the secure outsourcing of modular exponentiation as a representative to perform the analysis. The analysis of outsourcing scalar multiplication can be conducted in a similar way. We show that ExpSOS is secure under both HCS and MS model. Specifically, under the HCS model, the ExpSOS is $\frac{1}{2}$-log$_2$ $a$-efficient. Under the MS model, the ExpSOS is $\frac{1}{2}$ log$_b$ $a$-efficient and $(1 - \frac{1}{2b})$-verifiable, where $a$ is the exponent and $b$ is the security parameter.

**A. Security Analysis**

In ExpSOS, we conceal the base $u$ through the expansion function $(a + rN) \pmod{L}$ and the exponent $a$ is mapped to $a + k\phi(N)$. In our analysis, we show that given the public information $(L, U, A_1, A_2)$, the cloud cannot derive any key information about the input $(u, a, N)$ and the output $R = u^a \pmod{N}$.

**Theorem 6:** Under the MS model, ExpSOS is an outsource-secure implementation of modular exponentiation, where the inputs $(u, a, N)$ are protected.

**Proof:** The correctness of ExpSOS is guaranteed by Theorem 3. In the following, we prove the security of ExpSOS.

In our setting, the protected input is $x_P = (u, a, N)$ and the unprotected input is set to be $x_u = L$. The protected output is $y_P = R = u^a \pmod{N}$ while the unprotected output is empty. In the ideal process, the simulator $S$ acts in the following manner. $S$ will ignore any input that it received. Then, it makes queries $(U^*, A_1^*, A_2^*)$ to malicious $C'$. Then $S$ saves both its state and $C'$’s state. We note that in the real process, the inputs are computationally disguised in a random manner by the end-user before querying to $C'$. In the ideal process, the simulator $S$ always creates random and independent queries to $C'$. In this sense, the inputs to $C'$ are computationally indistinguishable. Thus, we have $\text{VIEW}_{\text{real}} \sim \text{VIEW}_{\text{ideal}}$.

We show that the proposed verification scheme has the security and effectiveness properties as described previously. The verifiability is based on the likelihood of finding two integers $R_1$ and $R_2$ so that $R_1^1 \cdot u^2 = R_2 \pmod{N}$ holds true.

**Lemma 1:** Given $(A_1, A_2, U, L)$, an adversary can generate $R_1$ and $R_2$ with probability $1/b$ such that $R_1^1 \cdot u^2 = R_2 \pmod{N}$. 
Proof: Given $A_1$ and $A_2$, an adversary can always select $B_1 = A_1 + \theta_1$ and $B_2 = A_2 + \theta_2$. Then,

$$U^{b_2-t_1}B_1 \pmod{N} = U^{A_2+\theta_2-t_1}A_1 \pmod{N} = u^{t_2+k_2\varphi(N)+t_1-t_2} \pmod{N} = u^{t_2}U^{b_2-t_1} \pmod{N}. $$

It is obvious that as long as $\theta_2 - t_1\theta_1 = 0$, the equality will hold. If the value of $t_1$ is correctly guessed, an adversary can select $\theta_1$ and $\theta_2$ such that $\theta_2 - t_1\theta_1 = 0$. Since the probability to correctly guess $t_1$ is $1/b$, an adversary can manipulate the result with probability $1/b$.

Theorem 7: Under MS model, the ExpSOS is $(1 - \frac{1}{2^\pi})$-verifiable.

This theorem indicates that if the cloud wants to manipulate the result, it has to guess the random integers, the probability to succeed is only $1/b$. In fact, if we outsource $C(U, A_1, L)$ and $C(U, A_2, L)$ in a random order, we can further reduce the probability for the cloud to guess the correct randoms to $1/2b$. According to Definition 4, ExpSOS is at least $(1 - 1/2b)$-verifiable.

The upper bound $b$ is a security parameter that measures the confidence of the end-user about the returned result. In practical computation outsourcing systems, the cloud would be severely punished if cloud manipulation is detected. Therefore, the benefit for the cloud to cheat would be hardly justifiable in this setting.

B. Complexity Analysis

We utilize outsourcing of modular exponentiation as a representative to analysis complexity. The analysis can be applied to scalar multiplication similarly. The essence of ExpSOS is to limit the number of modular multiplications for the end-user to compute modular exponentiation with the aid of the cloud. In our analysis, we utilize the number of modular multiplications, denoted as $\pi$, as a measurement. To calculate $u^a \pmod{N}$, the number of multiplications is $\pi = \frac{3}{2}l_a$, where $l_a$ is the bit length of $a$. Therefore, in calculating the modular exponentiation $u^a \pmod{N}$, $l_a \approx \log_2 a$ and $\pi \approx \frac{3}{2}\log_2 a$.

In ExpSOS, under the HCS model, to calculate $U$, $A$, and $L$, the end-user needs 3 multiplications. We notice that when the end-user knows the factors of $N$, it is computationally easy to calculate $\varphi(N)$. For example, when $N$ is a prime, $\varphi(N) = N - 1$. Moreover, the calculation of $\varphi(N)$ is a one-time process. The computational overhead for calculating $\varphi(N)$ is negligible especially when the end-user outsources modular exponentiation multiple times. Thus, under HCS model, we have $\pi_{\text{HCS}} = 3$. Hence, the computational gain from outsourcing is $\alpha_{\text{HCS}} = \pi/\pi_{\text{HCS}} = \frac{3}{4}\log_2 a$. From Definition 3, ExpSOS is $\frac{3}{4}\log_2 a$-efficient under the HCS model.

Under the MS model, the calculation of $L, U, A_1, A_2$ will take 4 multiplications. In the verification scheme, the end-user has to calculate $R_1^{t_1} \pmod{N}$ and $u^{t_2} \pmod{N}$. Thus, $\pi_{\text{MS}} = 4 + \frac{3}{2}\log_2 t_1 + \frac{3}{2}\log_2 t_2 + 1$. Since $t_1$ and $t_2$ are upper-bounded by $b$, we have $\log_2 t_1 + \log_2 t_2 \leq 2 \log_2 b$. Hence the computational gain from outsourcing is

$$\alpha = \frac{\pi}{\pi_{\text{MS}}} = \frac{3}{5} \log_2 a = \frac{3}{5} \log_2 a + 1 + \frac{3}{2} \log_2 t_1 + \frac{3}{2} \log_2 t_2 \geq \frac{3}{7} \log_2 a + 1 + \frac{3}{2} \log_2 t_1 + \frac{3}{2} \log_2 t_2 \geq \frac{3}{7} \log_2 a.$$ 

Thus under the MS model, ExpSOS is at least $\frac{3}{7}\log_2 a$-efficient.

C. Trade-Off Between Computation and Security

The above security and complexity analysis reveal the trade-off between computational overhead and security. In the MS model, ExpSOS is at least $\frac{3}{7}\log_2 a$-efficient and $(1 - 1/2b)$-verifiable. Both measurements relate to the same parameter $b$. On one hand, $b$ is the upper bound of the computational overhead that the end-user can tolerate. On the other hand, $b$ reveals the confidence of the end-user about the returned result which is also regarded as the security level of the result. When $b$ increases, the end-user has to carry out more computation. However, the probability that the end-user can verify the validity of the result also increases.

Thus, the proposed ExpSOS is cost-aware in the sense that it enables the end-user to have the flexibility to choose the most suitable outsourcing scheme according to its computational constraint and security demand. This is important especially when the end-users vary in computational power and security demands. It also makes ExpSOS widely applicable.

VI. APPLICATIONS

The proposed ExpSOS is able to conceal the base, the exponent and the module of the modular exponentiation $u^a \pmod{N}$. It can also be used to conceal the base point $P$ and multiplier $s$ of the scalar multiplication $sP$. With this feature, the parameters (private or public) within the cryptosystem are totally concealed from the outside, especially the cloud. Thus, the cryptosystem is isolated from the outsourced system. In this sense, ExpSOS can be regarded as a black box that takes as input $\{u, a, N, b\}$ and creates the output $u^a \pmod{N}$ as ExpSOS($u, a, N, b$) $\rightarrow u^a \pmod{N}$, where $b$ is security parameter selected by the end-user. The end-user will have a performance gain of at least $\frac{3}{7}\log_2 a$ and can verify the validity of the result with probability $1 - \frac{1}{2^\pi}$.

In this section, we will explore efficient outsourcing of exponential operations in some typical cryptographic protocols to the cloud.

A. Outsourcing Inner Product Encryption for Biometric Authentication

A practical Inner Product Encryption (IPE) scheme was recently introduced in [26]. In IPE, given the master secret
key msk, the encryption function Encrypt(msk, y) encrypts a vector y as
\[ \mathbf{c}_y = (C_1, C_2) = (g_2^{-\beta} \cdot s_k^y, B^*), \]
where \( g_2 \) is a generator of the underlying bilinear group \( \mathbb{G}_2 \), \( B \leftarrow \mathbb{G}_m(\mathbb{Z}_q) \) is a randomly generated matrix and \( B^* = \det(B) \cdot (B^{-1})^T \). Similarly, a key generation function KeyGen(msk, x) will generate a key skx associated with a vector x. Given skx, cy, and the decryption key, the decryption process will produce the inner product \( z = x \cdot y \). If x and y are binary vectors, the inner product z is actually the Hamming distance between x and y. Thus, IPE provides a way to evaluate the distance between two vectors from the ciphertexts.

Due to this feature, IPE is a promising technique to protect biometric templates in biometric authentication systems [26].

In biometric authentication systems, each user is identified by some biometric template (e.g., iris) represented by a vector x. During enrollment, an authority encrypts such template x as skx and stores it in a database together with the user’s identity as a record \((ID, skx)\). When another user with template y wants to authenticate himself, the authority encrypts the template as cy and sends it to the database server. The decryption obtains the inner product \( z = x \cdot y \), which is generally the hamming distance. If z is within the pre-defined threshold, the user is authenticated. In the following, we show that our ExpSOS scheme can be well applied to biometric authentication system to outsource the encryption process.

Normally, the KeyGen and Encrypt functions are executed in a resource-constrained device such as a scanner to collect and encrypt users’ biometric data. In the following, we take the Encrypt function as an example to illustrate the outsourcing procedure, which can also be applied to the function KeyGen.

Note that bilinear operation is typically implemented based on elliptic curves. Thus the underlying group \( \mathbb{G}_1 \) and \( \mathbb{G}_2 \) are sets of points on elliptic curves. The exponentiations are actually scalar multiplications of the base point \( g_2 \). As a result, we can utilize Protocol 3 as the basic scheme to outsource Encrypt. To simplify notations, we use a function \( T(\mathcal{P}, E) \to P' \) to represent the process in Protocol 3 to transform a point \( P = (x_1, y_1, z_1) \) to its disguised form \( P' = (x_2, y_2, z_2) \). Similarly, a function \( T^{-1}(P') \to P \) will recover P from P'. Let m be the order of group \( \mathbb{G}_2 \). The main computation involved in Encrypt are exponentiations of the same base point \( g_2 \). Let \( \beta \cdot y \cdot B^* = (\beta_1, \beta_2, \ldots, \beta_n) \). We can transform \( g_2 \) to its disguised form \( g_2' \) according to Protocol 3. For each \( \beta_i \), we transform \( \beta_i \) to \( \beta'_i = \beta_i + k_i m \), where \( k_i \) is a randomly generated integer. Then, \( g_2' \) and \( \beta'_i \) are outsourced to the cloud for computation. The result can be easily recovered by applying the function \( T^{-1} \). For result verification, we can jointly verify the exponentiations since they share the same base. To be specific, we can randomly select a set \( I \subset \{1, 2, \ldots, n\} \) and calculate \( \sum_{j \in I} \beta_j \). Then a new exponent for verification is computed as \( \delta = \prod_{i \in I} \beta_i \). The result verification is done by locally calculating exponentiations with small exponents \( t_1 \) and \( t_2 \). The protocol to outsource Encrypt is presented in Protocol 5.

\[ x = \sum_{j \in I} \beta_j \]

It is obvious that in the original Encrypt function, the major computation burden lies in the computation of \((n + 1)\) exponentiations, where \( n \) is the length of the vector. However, with ExpSOS, the end-user only needs to transform each entry in the vector as well as the base to the disguised form. The transformation for each entry only takes one multiplication. In result verification, the end-user only needs to conduct two exponentiations with small exponents and limited number of multiplications. In this sense, ExpSOS can achieve significant performance gain for the end-user.

**Protocol 5 Outsourcing Encryption of IPE**

**Input:** \( E, B = (\beta_1, \beta_2, \ldots, \beta_n), g_2 \)

**Output:** \( c = g_2^{\delta}, c_1 = g_2^{\beta_1}, C = (c, c_1, \ldots, c_n), \Lambda = \text{True or False} \)

**Key Generation KenGen(1^k, m) → L:**
1. \( E \) generates a large prime \( p \) and calculate \( L = pm \).
2. The public key is \( K_p = \{L\} \), and the private key is \( K_s = \{p, m\} \).

**Problem Transformation** \( T(g_2, B, E) \to G_2^*: \)
1. \( E \) calls function \( T \) in Protocol 3 to transform point \( g_2 \) to \( G_2^* \).
2. \( E \) selects random integers \( t_1, t_2, k \) and \( (k_1, k_2, \ldots, k_n) \) as the ephemeral key with the constraint that \( t_1, t_2 \leq b \).
3. \( E \) calculates \( \beta' = \beta + km \) and \( \beta'_i = \beta_i + k_i m \) for \( i = 1, 2, \ldots, n \).
4. \( E \) randomly selects a set \( I \subset \{1, 2, \ldots, n\} \) and calculate \( \delta = \prod_{i \in I} \beta_i + t_2 \).
5. \( E \) outsources \( G_2^*, \delta, B' = (\beta', \beta'_1, \beta'_2, \ldots, \beta'_n) \) to the cloud.

**Cloud Computation** \( C(G_2^*, \delta, B') \to C^* \):
1. \( S \) computes \( C' = (c'_1 = g_2^{\delta}, c'_2 = g_2^{\beta'_1}, c'_1 = g_2^{\beta'_1}, \ldots, c'_n = g_2^{\beta'_n}) \).
2. \( S \) returns \( C^* \) to \( E \).

**Result Verification** \( V(C^*) \to \Lambda \):
1. \( E \) checks \( T^{-1}(c'_i) = (\prod_{j \in I} c'_j)^{t_1} \cdot g_2^{t_2} \).
2. If the equality holds, set \( \Lambda \leftarrow \text{True} \). Otherwise, set \( \Lambda \leftarrow \text{False} \).

**Result Recovery** \( R(C^* \to C \):
1. \( E \) recovers the result as \( C = (c = T^{-1}(c'_1), c_1 = T^{-1}(c'_2), \ldots, c_n = T^{-1}(c'_n)) \).

### B. Outsourcing Identity Based Encryption

Identity Based Encryption (IBE) system is proposed to alleviate the process of public key certification in traditional public key cryptosystems. In IBE system, a user can utilize his identity such as his email address as the public key. Then a trusted authority will generate and distribute private key to the message receiver. The idea of IBE was initialized by Shamir in [27]. A practical IBE system was proposed in [28] based on bilinear pairing on elliptic curves.

In an implementation of IBE system [16, Ch. 5], the public parameters are an elliptic curve \( E(F_p)[m] \) and a base point \( P \in E(F_p)[m] \). Also, the trusted authority will publish his own public key \( P_I \in E(F_p)[m] \). The parameters are known...
to the authenticated users in the system. We assume that a user Alice uses the hash of her own identity to generate the public key which is a point on the elliptic curve, that is \( P_A \in E(\mathbb{F}_p)[m] \). For any other user Bob who desires to send a message \( M \) to Alice, he will conduct the following encryption process:

1) Bob selects a random integer \( r \in \mathbb{Z}_m \);
2) Bob computes \( C_1 = rP; \)
3) Bob computes \( C_2 = M \oplus H(e(P_A, P_T))^r; \)
4) Bob sets the ciphertext as \( C = (C_1, C_2) \).

In the above encryption algorithm, \( e(P_A, P_T) \) denotes the pairing between public points \( P_A \) and \( P_T \) and \( H(\cdot) \) is a hash function. We note that both the input and output of the pairing \( e(P_A, P_T) \) are public. Thus, the end-user Bob can obtain the pairing result denoted as \( g = e(P_A, P_T) \).

At the end, we can see that the computational burden for Bob lies in the scalar multiplication \( rP \) and the modular exponentiation \( g^r \pmod{p} \). We summarize the outsourcing of IBE as in Protocol 6.

**Protocol 6 Secure Outsourcing of Identity Based Encryption**

**Input:** \( E, P = (x, y, z), r, g = e(P_A, P_T) \)

**Output:** \( C_1 = rP, C_2 = H(g)^r \)

**Key Generation KeyGen(1^k) \( \rightarrow \) L:**

1. Bob selects a large prime \( q \) and calculates \( L \leftarrow pq \).

**Problem Transformation** \( T(p, r, L, P) \rightarrow (E', r_1, r_2, P', H(g), L) \):

1. Bob generates temporary key \( k_1, k_2, k_3, k_4, k_5, r_1, r_2 \) with \( t_1, t_2 < b \).
2. Bob calculates \( r_1 \leftarrow (r + k_1 p) \pmod{L}, r_2 \leftarrow (t_1 r + t_2 + k_2 p) \pmod{L}, x' \leftarrow (x + k_3 p) \pmod{L}, y' \leftarrow (y + k_4 p) \pmod{L} \).
3. Bob sets \( P' \leftarrow (x', y', z') \).

**Cloud Computation** \( C(E', r_1, r_2, P', H(g), L) \rightarrow (Q_1, Q_2, R_1, R_2) \):

1. S calculates \( Q_1 \leftarrow r_1 P', Q_2 \leftarrow r_2 P', R_1 \leftarrow H(g)^{r_1} \) and \( R_2 \leftarrow H(g)^{r_2} \).
2. S returns the results \( (Q_1, Q_2, R_1, R_2) \) to Bob.

**Result Verification** \( V(t_1, t_2, H(g), p, Q_1, Q_2, R_1, R_2) \rightarrow \Lambda; \)

1. Bob verifies the results by checking \( R_1^{t_1} \cdot H(g)^{t_2} = R_2 \pmod{p} \) and \( t_1 Q_1 + t_2 P = Q_2 \pmod{p} \), where the modular is applied coordinate-wise.

**Result Recovery** \( R(Q_2, M, R_2, p) \rightarrow (C_1, C_2) \):

1. Bob recovers the results \( C_1 \leftarrow Q_2 \pmod{p} \) and \( C_2 \leftarrow M \oplus R_2 \pmod{p} \).

From the above two applications, we can summarize some techniques in designing secure outsourcing scheme utilizing the outsourcing of exponential operation as a building block:

- It is more efficient and secure to share some common parameters in different subroutines of the outsourcing process. In this way, we can not only reduce the computational overhead, but also expose less information to the cloud.
- When outsourcing modular exponentiation with the same base, the computational overhead can be reduced by jointly verifying the result. This is demonstrated in the outsourcing of IPE.
- When making multiple queries to the cloud, the end-user can randomize the order of queries to increase verifiability.

**VII. Performance Evaluation**

In this section, we first compare ExpSOS with three existing works on secure outsourcing of modular exponentiation. Then we give some numerical results to show the efficiency of ExpSOS.

**A. Performance Comparison**

Secure outsourcing of cryptographic computations, especially modular exponentiation, has been a popular research topic [14], [17], [20], [22]–[24], [29]–[33]. For instance, Van Dijk et al. [33] proposed a secure outsourcing scheme for modular exponentiation with variable-exponent fixed base and fixed-exponent variable-base under single untrusted server model. However, the base is known to the server. In [17], the authors considered outsourcing variable-base variable-exponent modular exponentiation to two untrusted servers. Following this work, Chen et al. [14] improved the scheme in [17] in both efficiency and verifiability. Then, Wang et al. [20] made further improvement by reducing the two-server model to one single untrusted server model. The existing research in this area can be classified into three categories based on the supporting models, i.e., HCS model ([22]), MM model ([14], [17], [23]), and MS model ([20], [24]). In the following, we compare our ExpSOS with the six typical schemes in these three categories.

1) **HCS Model**: The most representative example in this category is the scheme introduced in [22]. It considers outsourcing \( u^a \) to a single honest-but-curious server, where \( u \in \mathbb{G} \) and \( a \in \mathbb{Z}_p \). It does not provide the ability to verify the result. The outsourcing process includes three steps. First, the end-user utilizes a \texttt{Rand} function to generate two pairs \((k_1, g^{k_1})\) and \((k_2, g^{k_2})\), where \( k_1 \) and \( k_2 \) are selected randomly. The base \( u \) is disguised as \( v = u \cdot g^{k_1} \) and the exponent \( a \) is divided into two parts \((a_0, a_1)\) such that \( a = a_1 \cdot T + a_0 \), where \( T = \lceil \sqrt{p} \rceil \). In the second step the end-user makes two queries to the server as \( S(v, T) = v^T \rightarrow h \) and \( S(g^{-ak_1-k_2}(mod\ p)) = g^{-ak_1-k_2} \rightarrow h_2 \). The end-user then utilizes Protocol 1 in [22] to calculate \( h_1 = v^{a_0} \cdot h^{a_1} \). In the third step the result is recovered as \( h_1h_2g^{k_2} = u^a \).

It is clear that the computational bottleneck lies in the computation of \( h_1 = v^{a_0} \cdot h^{a_1} \), where the bit length of \( a_0 \) and \( a_1 \) can be half of that of \( a \). Analysis shows that it takes \( l \) multiplications to determine \( h_1 \), where \( l \) is the maximum of \( \log a_0 \) and \( \log a_1 \). In addition, the scheme in [22] also requires the end-user to call \texttt{Rand} function twice and then conduct Euclidean division to obtain \( a_0 \) and \( a_1 \).
2) MM Model: The most representative examples for MM model include [14], [17], and [23]. MM model considers outsourcing modular exponentiation to two untrusted servers \( S_1 \) and \( S_2 \) and it is assumed that the two servers do not collude which corresponds to our MM model. In these schemes, a subroutine \( \text{Rand} \) is utilized to generate random modular exponentiations pairs. Specifically, on input a base \( g \in \mathbb{Z}_p^* \), the subroutine \( \text{Rand} \) will generate random pairs in the form of \((\theta, g^\theta \mod p)\), where \( \theta \) is a random number in \( \mathbb{Z}_p^* \). Then the end-user can make queries to \( \text{Rand} \) and each query will return a random pair to the end-user. Typically, the subroutine \( \text{Rand} \) is implemented via two different methods. One method is that a table of random pairs is pre-computed from a trusted server and stored at the end-user. Whenever the end-user needs to make a query to \( \text{Rand} \), it just randomly draws a pair from the table. The critical problem of this method is that it will take a lot of storage space from the end-user. Specifically, a random pair will take \( 2\ell \) space, where \( \ell \) is the bit length of \( p \). In addition, to make the generator of the pairs look random, the table size should be large. As a result, the storage overhead becomes unacceptable for the resource-constrained end-users. The other method is to utilize some pre-processing techniques such as the BPV generator [31] and the EBPV generator [32]. To generate one random pair, the EBPV generator takes \( O(\log^2 \ell_a) \) modular multiplications, where \( \ell_a \) is the bit length of the exponent.

The scheme proposed in [17] can be briefly summarized as follows. First, the end-user runs \( \text{Rand} \) 6 times to obtain random pairs \((a, g^a), (\beta, g^\beta), (t_1, g^{t_1}), (t_2, g^{t_2}), (r_1, g^{r_1}), (r_2, g^{r_2})\). Then \( a^u \) can be written as

\[
u^a = \nu^{b \cdot f - b \cdot \left(\frac{\nu}{f}\right)^{-a} - d \cdot \left(\frac{\nu}{d}\right)^{-a - d}}
\]

where \( \nu = g^a, b = \frac{\ell}{f}, f \) and \( d \) are random integers.

The end-user then makes queries, in random order to the cloud server \( S_1 \) to derive \( Q_1^1 = \left(\frac{\nu}{f}\right)^d, Q_1^2 = f^{-b}, Q_1^3 = g^{t_1}, Q_1^4 = g^{r_1} \). Similarly, the end-user makes queries to the second cloud server \( S_2 \) to derive \( Q_2^1 = \left(\frac{\nu}{d}\right)^{-d}, Q_2^2 = \left(\frac{\nu}{f}\right)^{-a}, Q_2^3 = (g^\beta)^{f}, Q_2^4 = (g^\alpha)^{f, \omega} \). The result can be recovered as \( a^u = g^{\beta^1} \cdot Q_1^1 \cdot Q_1^2 \cdot Q_2^3 \cdot Q_2^4 \). The result verification is carried out by checking whether \( Q_1^1 \cdot Q_2^1 = g^{t_1} \) and \( Q_1^2 \cdot Q_2^2 = g^{r_2} \). We note that the end-user needs to make queries to each server \( S_1 \) and \( S_2 \) for four times, among which the first two are computation queries and the other two are test queries. Since the test queries and the computation queries are independent, the servers can potentially compute the test queries honestly but cheat in the computation queries. The authors address this problem by sending out the queries in random order. The verifiability of this scheme is \( \frac{1}{4} \). In the outsourcing process, \( E \) has to run the subroutine \( \text{Rand} \) 6 times, make 9 modular multiplications (MMul) and 5 modular inversions (MInv), where \( \text{Rand} \) has a complexity of \( O(\log^2 n) \) MMul and \( n \) is the bit length of the exponent.

Based on [17], Chen et al. [14] made some improvement by reducing the computational overhead to 5 \( \text{Rand} \), 7 MMul and 3 MInv and the queries to the two-server are reduced to 6 times in total. Moreover, the verifiability is improved to \( \frac{1}{2} \).

Unlike [14] and [17], the scheme in [23] utilized the Chinese Remainder Theorem to disguise the secret values. To be specific, the group \( \mathbb{Z}_p \) is extended to \( \mathbb{Z}_n \), where \( n = pr_1r_2 \) and \( r_1 \) and \( r_2 \) are large prime numbers. Together with some randomly generated values, the secret values \( u \) and \( a \) are transformed to the corresponding elements in \( \mathbb{Z}_n \). The scheme also employs the function \( \text{Rand} \) to generate random exponentiation pairs \((t_i, g^{t_i})\). Result verification is achieved by comparing the results returned from two independent servers. Compared with [14] and [17], the advantage of the scheme is that it can achieve checkability 1 for every query result. Complexity analysis shows that to outsource one modular exponentiation, the end-user has to carry out 7 multiplications in addition to executing the \( \text{Rand} \) function 4 times.

3) MS Model: The most representative examples for MS model include [20] and [24]. MS model employs only one single cloud server. The scheme in [20] can be summarized as follows. First, the end-user runs \( \text{Rand} \) 7 times to obtain random pairs \((a_1, g^{a_1}), (a_2, g^{a_2}), (a_3, g^{a_3}), (a_4, g^{a_4}), (t_1, g^{t_1}), (t_2, g^{t_2}), (t_3, g^{t_3})\). Then it calculates \( c = (a - b\chi) \mod p \), \( \omega = u/\mu_1, h = u/\mu_3 \), \( \theta = (a_1b - a_2)\chi + (a_3c - a_4) \mod p \), where \( \chi, b \) are randomly selected and \( \mu_i = g^{\mu_i} \), for \( i = 1, 2, 3, 4 \). The end-user then queries to a single cloud server \( S \) to compute \( Q^{(i)} \) for \( i = 1, 2, 3, 4 \), where \( Q^{(1)} = (g^{t_1})^p, Q^{(2)} = (g^{t_2})^p, Q^{(3)} = \omega^b, Q^{(4)} = h^c \). The result verification is carried out by checking whether \( \chi^{(1)} \cdot Q^{(2)} = g^{t_3} \) is true. Similarly, the queries can be divided as test queries and computation queries. As a result, the cloud can compute honestly on the test queries and cheat on the computation queries. Thus, due to the random order of the queries, the verifiability of this scheme is \( \frac{1}{2} \). We note that in the result recovery process, the end-user has to compute an exponentiation \( (\mu_2 \cdot \omega^c)^c \) which takes \( \frac{1}{2} \log \chi \) multiplications. The whole scheme will take 7 \( \text{Rand} \), 12 + \( \frac{1}{2} \log \chi \) MMul, 4 MInv and make 4 queries to the cloud server. In comparison, ExpSOS can avoid inversion and only needs \((5 + 3 \log b)\) MMul, where \( b \) is a small integer.

The scheme in [24] also investigated secure outsourcing of modular exponentiation with variable base and variable exponent to a single untrusted server. The main drawback of [24] is its large number of queries \((l + k + 2)\) made to the server, which introduces high computational and communication overhead. In fact, the scheme in [24] first utilizes a sub-protocol SubAlg to outsource exponentiations with fixed base. To support the case of variable base, the base is split into multiple sets of different sizes. The scheme in [24] also employs the \( \text{Rand} \) five times to generate exponentiation pairs \((t_i, g^{t_i})\). Similar to our work, in the result verification phase, [24] also needs to conduct an exponentiation with a small exponent to achieve a checkability of \( \frac{1}{c(c-1)} \). In addition, the end-user has to conduct \( l + k + 8 \log c + 38 \) multiplications and 1 inversion operation. As shown in an example given in [24], when \( l = k = 29 \) and \( c = 4 \), the number of multiplications reaches 100 and the end-user needs to make 60 queries to the server. Also, as
pointed out in [2], one modular inversion is about 100 times slower than a modular multiplication. This further increases the complexity of the scheme in [24].

4) Performance Comparison of ExpSOS With the Existing Schemes: In comparison, ExpSOS under MM model can be modified as in Protocol 7. Since the cloud servers $S_1$ and $S_2$ do not collude, the only way to make the equality condition satisfied is that $S_1$ and $S_2$ both compute honestly. Thus the verifiability is 1. Moreover, in this process, we successfully avoid inversion that is considered much more expensive than multiplication in field operations. The total computational overhead is only 3 MMul.

To measure the communication overhead, we utilize the total length of bits to be transmitted between the end-user and cloud server in order to outsource one exponentiation. Let $\ell$ be the bit length of $N$. Without loss of generality, we assume that it takes $\ell$ bits to transmit one element in $\mathbb{Z}_N$. In our scheme, the elements in $\mathbb{Z}_N$ are transformed to elements in $\mathbb{Z}_L$, where $L = pN$. Since $p$ is comparable to the divisors of $N$, the bit length of $L$ is at most $2\ell$. Thus, we can assume that it takes $2\ell$ bits to transmit one element in $\mathbb{Z}_L$ (worst case). Following this analysis, the communication cost for each scheme is given in Table II. It shows that our scheme can achieve the highest communication efficiency. In terms of security, we have shown that ExpSOS can successfully conceal the base, exponent and the modulus of the modular exponentiation. It is computationally infeasible for the cloud to derive any key information from the disguised problem. In comparison, all the above three schemes [14], [17], [20] can only conceal the exponent and base while the modulus is exposed to the cloud. Thus ExpSOS can provide much improved security. Moreover, the three schemes in [17], [14], and [20] achieve verifiability of $\frac{1}{2}$, $\frac{1}{4}$, and $\frac{1}{4}$ respectively. In comparison, the verifiability of ExpSOS is $1 - \frac{1}{2^N}$ that is close to 1. This means that the end-user is more confident about the results returned by the cloud. Furthermore, the security of the schemes in [14] and [17] relies on the assumption that the two cloud servers will not collude. The scheme [20] and the proposed ExpSOS are applicable to one single untrusted server hence eliminating the non-collusion assumption.

The comparison of ExpSOS and the schemes in [14], [17], and [20] is summarized in Table II. We can see that the proposed ExpSOS outperforms other schemes in both computational complexity and security. ExpSOS also makes the least queries to the cloud that will introduce the least communication overhead. Moreover, ExpSOS is cost-aware in computational overhead and security such that the end-users can select the most suitable outsourcing scheme according to their own constraints and demands. Also, ExpSOS can be modified such that it is applicable to HCS, MM and MS model.

<table>
<thead>
<tr>
<th>$l_N$ (bits)</th>
<th>ExpSOS ($t_s$ (bits))</th>
<th>[14]</th>
<th>[20]</th>
</tr>
</thead>
<tbody>
<tr>
<td>$t_0$</td>
<td>$t_s$</td>
<td>$\tau$</td>
<td>$t_0$</td>
</tr>
<tr>
<td>128</td>
<td>1358</td>
<td>87</td>
<td>15.6</td>
</tr>
<tr>
<td>256</td>
<td>2554</td>
<td>89</td>
<td>28.6</td>
</tr>
<tr>
<td>384</td>
<td>4095</td>
<td>127</td>
<td>32.3</td>
</tr>
<tr>
<td>512</td>
<td>7837</td>
<td>134</td>
<td>58.6</td>
</tr>
<tr>
<td>640</td>
<td>10991</td>
<td>146</td>
<td>75.0</td>
</tr>
<tr>
<td>768</td>
<td>11427</td>
<td>148</td>
<td>77.2</td>
</tr>
<tr>
<td>896</td>
<td>17445</td>
<td>158</td>
<td>110.2</td>
</tr>
<tr>
<td>1024</td>
<td>20235</td>
<td>174</td>
<td>116.2</td>
</tr>
</tbody>
</table>

To measure the communication overhead, we utilize the total length of bits to be transmitted between the end-user and cloud server in order to outsource one exponentiation. Let $\ell$ be the bit length of $N$. Without loss of generality, we assume that it takes $\ell$ bits to transmit one element in $\mathbb{Z}_N$. In our scheme, the elements in $\mathbb{Z}_N$ are transformed to elements in $\mathbb{Z}_L$, where $L = pN$. Since $p$ is comparable to the divisors of $N$, the bit length of $L$ is at most $2\ell$. Thus, we can assume that it takes $2\ell$ bits to transmit one element in $\mathbb{Z}_L$ (worst case). Following this analysis, the communication cost for each scheme is given in Table II. It shows that our scheme can achieve the highest communication efficiency. In terms of security, we have shown that ExpSOS can successfully conceal the base, exponent and the modulus of the modular exponentiation. It is computationally infeasible for the cloud to derive any key information from the disguised problem. In comparison, all the above three schemes [14], [17], [20] can only conceal the exponent and base while the modulus is exposed to the cloud. Thus ExpSOS can provide much improved security. Moreover, the three schemes in [17], [14], and [20] achieve verifiability of $\frac{1}{2}$, $\frac{1}{4}$, and $\frac{1}{4}$ respectively. In comparison, the verifiability of ExpSOS is $1 - \frac{1}{2^N}$ that is close to 1. This means that the end-user is more confident about the results returned by the cloud. Furthermore, the security of the schemes in [14] and [17] relies on the assumption that the two cloud servers will not collude. The scheme [20] and the proposed ExpSOS are applicable to one single untrusted server hence eliminating the non-collusion assumption.

The comparison of ExpSOS and the schemes in [14], [17], and [20] is summarized in Table II. We can see that the proposed ExpSOS outperforms other schemes in both computational complexity and security. ExpSOS also makes the least queries to the cloud that will introduce the least communication overhead. Moreover, ExpSOS is cost-aware in computational overhead and security such that the end-users can select the most suitable outsourcing scheme according to their own constraints and demands. Also, ExpSOS can be modified such that it is applicable to HCS, MM and MS model.

### Protocol 7 ExpSOS Under MM Model

**Input:** $N, u, a \in \mathbb{R}_N$.

**Output:** $R_0 = a^d \pmod{N}$, $\Lambda = \{\text{True}, \text{False}\}$.

**Key Generation** $(1^4, p) \rightarrow L$:

1. $E$ generates a large prime number $p$ and calculate $L \leftarrow pN$. The public key is $K_p = \{L\}$ and the private key is $K_s = \{p, N\}$.
2. $E$ selects random integers $r, k \in \mathbb{Z}_N$ as the temporary key.

**Problem Transformation** $T(a, u, L) \rightarrow (A, U)$:

1. $E$ calculates $A \leftarrow a + k\varphi(N)$ and $U \leftarrow (u + rN) \pmod{L}$.
2. $E$ then outsource $\{U, A, L\}$ to both cloud servers $S_1$ and $S_2$.

**Cloud Computation** $C(A, U, L) \rightarrow (R_1, R_2)$:

1. $S_1$ computes $R_1 \leftarrow U^A \pmod{L}$ and $S_2$ computes $R_2 \leftarrow U^A \pmod{L}$.
2. The results $R_1$ and $R_2$ are returned to $E$.

**Result Verification** $V(R_1, R_2, N) \rightarrow \Lambda$:

1. $E$ checks whether $R_1 = R_2 \pmod{N}$. Set $\Lambda \leftarrow \text{True}$ if the equality holds; otherwise set $\Lambda \leftarrow \text{False}$.

**Result Recovery** $R(R_1, N) \rightarrow R$:

1. $E$ recovers the result as $R \leftarrow R_1 \pmod{N}$.

---

**Table I**: Numeric Results

<table>
<thead>
<tr>
<th>$l_N$ (bits)</th>
<th>$t_s$ (ms)</th>
<th>$t_s$ (ms)</th>
<th>$t_s$ (ms)</th>
<th>$t_s$ (ms)</th>
<th>$t_s$ (ms)</th>
<th>$t_s$ (ms)</th>
<th>$t_s$ (ms)</th>
<th>$t_s$ (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>128</td>
<td>256</td>
<td>384</td>
<td>512</td>
<td>640</td>
<td>768</td>
<td>896</td>
<td>1024</td>
</tr>
<tr>
<td>8</td>
<td>1358</td>
<td>2554</td>
<td>4095</td>
<td>7837</td>
<td>10991</td>
<td>11427</td>
<td>17445</td>
<td>20235</td>
</tr>
</tbody>
</table>

---

B. Numeric Results

In this section, we compare the performance of ExpSOS for modular exponentiation with the schemes in [14] and [20] through simulation in mobile phones. The computation of both the end-user and the cloud server is simulated in the same phone Samsung GT-I9100 with Android 4.1.2 operating system. The CPU is Dual-core 1.2 GHz Cortex-A9 with...
1 GB RAM. In the outsourcing process, we focus on the computational gain, denoted as \( r \), of the outsourcing. We measure the local processing time \( t_0 \) to compute the modular exponentiation \( a^b \mod N \) without outsourcing and the local processing time \( t_s \) with outsourcing which includes the problem transformation, result recovery and result verification. To measure the performance of ExpSOS under different levels of complexity, we let the size of the ring \( l_N \) vary from 128 bits to 1024 bits. Also, to show the cost-awareness of ExpSOS, we let the size of the security parameter \( l_b \) vary from 4 bits to 16 bits. The processing time is averaged over 1000 independent rounds. The numeric result is shown in Table I where each number stands for the average processing time for 100 rounds. We can see that when the size of the ring \( l_N \) increases, the performance gain \( r \) also increases for the same security parameter \( b \). This means that when the original problem is more complex, ExpSOS would have a better performance.

The reason is that the complexity of modular exponentiation depends on the number of multiplications that is positively correlated to the logarithm of the size of the ring \( l_N \). However, in ExpSOS the local processing takes almost the same number of multiplications for a fixed security parameter \( b \). We can also see that there exists a trade-off between security and computational overhead. When \( b \) increases, the computational overhead increases accordingly. Since the verifiability is \( 1 - \frac{1}{2^b} \), a bigger \( b \) means better security guarantees.

The numeric results also demonstrate the high efficiency of ExpSOS compared to the schemes in [14] and [20], which coincides with our theoretical analysis. We note that to achieve the same checkability, the schemes in [14] and [20] should be compared with ExpSOS with \( l_b = 4 \). In the simulation, the implementation of the \texttt{Rand} function is rather simplified and it only requires the end-user to conduct an average of 5 multiplications. From the simulation, we can see that the performance of the scheme in [20] is dominated by the computation of \((\cdot)^{2^k}\), where \( \chi \) has 64 bits as recommended in [20]. Our high efficiency comes from the fact that the required number of multiplications is much less and we do not need the end-user to call the \texttt{Rand} function and compute modular inversion.

### VIII. Conclusion

In this paper, we design a secure outsourcing scheme ExpSOS that can be widely used to outsource general exponentiation operations for cryptographic computations, including modular exponentiation and scalar multiplication. The proposed ExpSOS enables end-users to outsource the computation of exponentiation to a single untrusted server at the cost of only a few multiplications. We also provide a verification scheme such that the result is verifiable with probability \( 1 - \frac{1}{2^b} \). With the security parameter \( b \), ExpSOS is cost-aware in that it can provide different security levels at the cost of different computational overhead. The comprehensive evaluation demonstrates that our scheme ExpSOS can significantly improve the existing schemes in efficiency, security and result verifiability.

### References


Kai Zhou (S’16) received the B.S. degree in electrical engineering from Shanghai Jiao Tong University, China, in 2013. He is currently pursuing the Ph.D. degree in electrical and computer engineering from Michigan State University. His research interests include applied cryptography, cloud security and privacy, coding theory, and secure communication.

M. H. Afifi received the B.S. and M.Sc. degrees in electrical engineering from the Department of Electronics and Communications, Arab Academy for Science, Technology, Alexandria, Egypt, in 2009 and 2012, respectively. He is currently pursuing the Ph.D. degree with the Department of Electrical and Computer Engineering, Michigan State University (MSU), East Lansing, Michigan, USA. He is currently a Research Assistant with the Department of Electrical and Computer Engineering, MSU. His research interests include cybersecurity, data privacy, wireless communications, signal processing and wireless sensor networks.

Jian Ren (SM’09) received the B.S. and M.S. degrees in mathematics from Shaanxi Normal University, and the Ph.D. degree in EE from Xidian University, China. He is currently an Associate Professor with the Department of ECE, Michigan State University. His current research interests include network security, cloud computing security, privacy-preserving communications, distributed network storage, and Internet of Things. He was a recipient of the U.S. National Science Foundation Faculty Early Career Development (CAREER) Award in 2009. He is the TPC Chair of IEEE ICNC’17 and the General Chair of ICNC’18.