International Journal of Computer Science and Information Technology
ISSN: 3005-9682 (Print), ISSN: 3005-7140 (Online) | Volume 5, Number 2, Year 2025
DOI: https://doi.org/10.62051/ijcsit.v5n2.11
Journal homepage: https://wepub.org/index.php/IJCSIT/index

Cloud-assisted Outsourcing of Modular Exponentiation Calculation

Shuai Zhang *

College of Computer Science and Technology, Qingdao University, Qingdao, China
* Corresponding Author: Shuai Zhang

ABSTRACT

With the rapid development of cloud computing and cluster computing, outsourced computation has become an important means for resource-constrained devices to complete complex tasks, and the outsourcing of modular exponentiation operations has attracted much attention. However, outsourced computation faces security challenges such as privacy, verifiability, and efficiency. This paper reviews existing privacy - preserving schemes for modular exponentiation outsourcing and proposes a privacy - preserving modular exponentiation outsourcing scheme based on secure multi - party computation for a dual - server model. This scheme designs a secure multi - party computation protocol using the Montgomery algorithm to accelerate modular exponentiation outsourcing and improve efficiency. It can protect the entire process of the modular exponentiation outsourcing algorithm, the privacy of data owner information, and the final computation results. Meanwhile, the system model, threat model, and design goals are elaborated in detail, and the correctness, security analysis, and experimental comparison of the scheme are carried out. The research shows that this scheme can ensure accurate and secure computations while achieving high efficiency, providing a more reliable solution for modular exponentiation outsourcing.

KEYWORDS

Modular Exponentiation; Outsourced Computation; Secure Multi-party Computation; Montgomery Algorithm.

1. INTRODUCTION

With the rapid development of the internet, an enormous amount of resources has been aggregated, propelling the internet's transition from a traditional communication platform to an intelligent computing platform. In the 1960s, Licklider, who was instrumental in the development of ARPAnet, first introduced the concept of the" Intergalactic Computer Network," a precursor to the notion of " cloud computing." Around 2006, Google and Amazon formally proposed the concept of cloud computing. Cloud computing [1, 2, 3], as an evolution of parallel computing, distributed computing, and grid computing, reduces the demand for hardware resources on the user side while assisting users in completing complex computational tasks. It plays a crucial role in intelligent data processing within Intelligent Decision Support Systems (IDSS). Depending on the target service, cloud computing architecture can be divided into three types: Infrastructure as a Service (IaaS), Platform as a Service (PaaS), and Software as a Service (SaaS). Similarly, cluster computing connects numerous computers to function as a powerful machine that provides services. These service models are implemented through outsourcing, where high-cost aspects such as hardware and software procurement, management, and maintenance are handled by cloud servers, and customers only need to pay for the services. This allows customers to obtain maximum service with minimal cost.

As cloud computing and cluster computing continue to develop rapidly, mobile devices such as smartphones are gradually joining the internet. However, due to the limited computational power, battery life, and other resources of these devices, the widespread application of mobile services remains constrained. Additionally, as the amount of information increases and computational tasks become more complex, resource-constrained users struggle to keep up with the increasingly sophisticated computing models. The emergence of outsourcing services has allowed mobile devices to overcome these resource limitations, enabling users to utilize the vast computing resources available in the cloud on a pay-as-you-go basis. This so-called outsourced computation is gradually becoming the mainstream in the industry. While outsourced computation offers significant convenience, it also introduces several new security challenges [4, 5, 6].

The first challenge is privacy. Outsourced computation tasks may involve sensitive data, such as financial records or personal health information, which must not be exposed to the cloud. However, once users hand over their data to the cloud, they lose control over it, and the powerful computational servers might attempt to glean sensitive information from the user's data, which is something that must be avoided. The second challenge is verifiability. The purpose of using outsourced computation is to reduce the computational burden on the client while ensuring the correct execution of tasks. However, due to various factors, the untrusted cloud may return incorrect results. For instance, the cloud service provider or program might have hardware faults or software bugs, or the cloud may produce a computationally distinguishable but incorrect result to reduce storage and computational costs, rather than executing the task correctly. Therefore, the verifiability of returned results must be considered in outsourced computation scenarios. The third challenge is efficiency. In outsourced computation, users not only need to blind the original data to protect its privacy but also perform verification and recovery steps to ensure that the cloud has honestly executed the computation. The blinding process and verification recovery process are the main tasks the client performs during the entire outsourcing process. Hence, these processes must be efficient, meaning the total computational load on the user in the outsourcing process should be significantly lower than if they were to execute the original computation themselves; otherwise, outsourcing would become meaningless.

1.1. Related Work

Modular exponentiation, also known as modular power operation, is a fundamental operation in cryptographic systems like RSA, DSS, and ElGamal. When the exponent in modular exponentiation is n bits long, approximately 1.5n modular multiplications are required. However, for resource-constrained mobile devices, the computational cost of such operations is prohibitively high. Therefore, studying the secure outsourcing of modular exponentiation holds significant theoretical and practical importance. Existing outsourcing schemes can be classified into two categories based on their security models: Two-Untrusted-Server (TUS) and Single-Untrusted-Server (SUS) based schemes.

In TUS-based outsourcing schemes, critical information is typically split into two parts using blinding techniques, which are then computed by two non-colluding cloud servers. This ensures that neither server can independently reconstruct any sensitive information. The client then processes the results returned by the servers through local verification and result recovery. Hohenberger et al.[7]were the first to formally define a security model for outsourced computation and proposed a modular exponentiation outsourcing protocol based on the TUS model. This protocol has a time complexity of O(log2n), but the server can deceive the user with a 1/2 probability. Ma et al.[8]proposed two outsourcing protocols that use pseudorandom functions to blind the base and exponent, increasing the verifiability to 3/4, but these protocols do not simultaneously protect the base and exponent. To address this issue, Chen et al.[4] improved existing schemes by introducing a new method for splitting the base and exponent. Their scheme has a time complexity of O(log2n/n) and can detect malicious server behavior with a 2/3 probability. Ye et al.[9] further enhanced verifiability to 19/20 while maintaining the same time complexity by employing a novel splitting method. Liu et al.[10] proposed a scheme for outsourcing modular exponentiation with a composite modulus, in which the client

needs only 5+2k modular multiplications, but the server can still deceive the client with a 1/2 probability. Ma et al.[11] improved an existing scheme for modular exponentiation with a prime modulus by introducing a new splitting method, increasing verifiability to $1 - 2/(3s)$, with a recommended s value of 16. Later, Ren et al.[12] proposed an outsourcing scheme based on Euler's theorem. Their scheme uses the Rand function to generate key pairs for splitting the base and exponent, achieving nearly optimal verifiability with a probability of 1 while maintaining the same complexity. However, these schemes only protect the base and exponent, not the modulus. Uzunkol et al.[13] proposed the first non-interactive fully verifiable outsourcing protocol based on the Chinese Remainder Theorem, which not only protects the base and exponent but also the modulus, allowing the client to detect erroneous results from the server with a probability of 1.

The main advantage of TUS-based outsourcing schemes lies in their higher efficiency and verifiability. Unlike SUS-based schemes, TUS-based schemes are more efficient in executing the local blinding process and in the verification and recovery of results. In SUS-based schemes, blinding typically involves a locally selected small exponent, which not only participates in blinding but also in verification, thereby increasing the computational burden on the client. However, TUS-based schemes rely on the assumption that the two servers do not collude, an idealized assumption that cannot withstand collusion attacks in real-world scenarios. Therefore, studying outsourcing schemes based on a single untrusted server is crucial.

In SUS-based outsourcing schemes, Dijk et al.[14] were the first to propose a secure outsourcing algorithm for modular exponentiation using a single server, where the base and exponent are logically split to blind critical information. In their scheme, the server needs to perform 4.5logs + 3logws + 5 modular multiplications and can detect the server's returned result with a probability of 1. However, this algorithm does not simultaneously protect the base and exponent. Wang et al.[15] introduced another single-server scheme that uses a locally generated random integer of at least 64 bits as a key to blind the input. While their scheme can protect both the base and exponent, it is vulnerable to lattice-based attacks[16]. Liu et al.[17] proposed a new outsourcing scheme for the case where the modulus is a composite number, which offers near-perfect verifiability. Xiang et al.[18] accelerated the entire outsourcing process by utilizing their subroutine FBVP, reducing the complexity to O(1) while maintaining close to optimal verifiability of 1. Subsequently, Xiang et al.[19]further improved their scheme, reducing the number of modular multiplications the client needs to perform to just 3 while preserving the original verifiability. Ye et al. also used locally generated 64-bit random numbers and introduced a new splitting method to blind the input. In their scheme, the client needs to perform 15 + 1.5logr + 1.5logt1 + 1.5logt2modular multiplications and can detect malicious server behavior with a probability of 119/120. Later, Ding et al.[20] extended Ye et al.'s[9] scheme to support dual and multi-modular exponentiation outsourcing, with all their schemes achieving near-perfect verifiability. Cai et al.[21] utilized a new splitting technique to propose another outsourcing scheme capable of detecting errors with a probability close to 1. However, these schemes only protect the base and exponent, failing to safeguard the modulus. Zhou et al.[22], based on Euler's theorem, proposed an outsourcing scheme that simultaneously protects the base, exponent, and modulus, where the client can detect errors with a probability of 1-1/(2b). Li et al.[23] introduced a new outsourcing scheme, using a locally generated random integer of at least 64 bits as a blinding factor and employing a new splitting technique to achieve optimal verifiability of 1. However, a brief security analysis by Rangasamy et al.[24] demonstrated that the schemes by Cai et al.[21], Zhou et al.[22], and Li et al.[23] did not achieve the claimed security, and they improved Zhou et al.'s[24] scheme. Subsequently, Zhu et al.[25] and Fu et al.[26] proposed outsourcing protocols with different splitting methods, and their schemes could also detect malicious behavior by the cloud server with an optimal probability of 1. Fu et al.[27] further proposed a new outsourcing scheme for the case of a composite modulus, which, through the use of new splitting techniques, offers near-perfect verifiability. However, Su et al.[28] pointed out errors in the scheme and proposed a new outsourcing scheme for composite moduli based on Euler's function, where the client can detect errors with a probability of 119/120. Hu et al.[29] proposed a parallel modular exponentiation outsourcing scheme that protects the base, exponent, and

modulus simultaneously. Xu et al.[30] introduced an entirely new outsourcing scheme based on smart contracts, treating smart contracts as cloud servers. Since the result returned by the blockchain after the execution of the smart contract must be correct, no local verification is needed, resulting in a verifiability of 1, while significantly reducing the computational costs for both the client and server. Recently, Li et al. [31] proposed the first distributed outsourcing scheme utilizing multiple non-colluding edge nodes, where the client needs to perform 4n+7+1.5logr modular multiplications and can detect malicious server behavior with a probability of 98.3%.

1.2. Our Contribution

After reviewing existing privacy-preserving schemes for modular exponentiation outsourcing, we recognized the benefits of a dual-server outsourcing approach. Consequently, we developed a privacy-preserving modular exponentiation outsourcing scheme based on secure multi-party computation and enhanced it using optimization techniques from [32]. Under the dual-server model, we propose an outsourcing scheme for computations of the form (a^e mod N). Specifically, our scheme features the following features:

1.Ours scheme safeguards the entire process of the modular exponentiation outsourcing algorithm, including all stages of interaction with cloud servers.

2.Our algorithm protects both the privacy of the data owner's information and the final computation results. It can be categorized into three protection schemes: (1) protection of the modulus, (2) protection of the base and exponent, (3) protection of the modulus, base, and exponent.

3.Based on the characteristics of the Montgomery algorithm, we have designed a secure multi-party computation protocol leveraging this algorithm to accelerate modular exponentiation outsourcing and enhance efficiency.

1.3. Road Map

The remainder of this paper is organized as follows: Section 2 introduces the system model, threat model, and design objectives. Section 3 covers the essential symbols and mathematical concepts used in our design. Additionally, Section 3 reviews the Montgomery algorithm and various secure multi-party computation protocols. In Section 4 and 5, we provide a detailed description of our proposed solution. Section 6 showcases our experimental analysis and comparisons. Finally, Section 7 concludes the paper.

2. SYSTEM MODEL, THREAT MODEL AND DESIGN GOAL

2.1. System Model

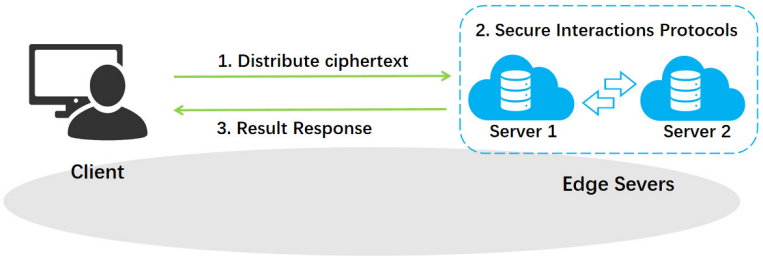

Figure 1. The system model

Client: The user possesses the values a, e, and N, and requires the computation of (a^e mod N). Due to limited local computational resources, the user opts to encrypt a, e, and N, outsourcing the computation to a cloud server. By leveraging the cloud server's robust processing power, the user seeks to obtain the necessary computational services.

Edge Severs: In our model, the edge servers consist of two distinct servers, S 1 and S 2, each equipped with substantial computational power and ample storage capacity. Upon receiving a computation request from the user, S 1 and S 2 collaborate to compute (a^e mod n) and subsequently transmit the result back to the user.

2.2. Threat Model

In a dual-server outsourcing system model, it is typically assumed that there is at most one untrusted server, with the assumption that the two servers do not collude. The untrusted server may tamper with user-uploaded data or selectively delete infrequently accessed or used data for its own benefit, while the trusted server will honestly adhere to the outsourcing scheme's protocol. In practice, users can upload data to multiple cloud servers or outsource complex modular exponentiation calculations to several cloud servers, benefiting from the services provided by multiple cloud providers. Moreover, theoretically, the probability of different cloud service providers colluding to access user privacy for commercial gain is nearly zero. Therefore, this paper employs a dual-server model with at most one untrusted server that does not collude with the other.

2.3. Design Goal

In summary, our goal is to develop a computation outsourcing scheme that is accurate, secure, and highly efficient. Based on the explanations of our system architecture and threat models, we will provide their definitions in the subsequent sections.

2.3.1. Correctness

Generally, if the user and both edge servers honestly and correctly execute the protocol, the user can accurately compute the modular exponentiation result based on the computations output1 and output2 returned by the edge servers.

2.3.2. Security

During the interaction between the user and the server, apart from the information that has already been disclosed, the scheme does not expose any additional sensitive data that needs to be protected. This includes details such as the base a, exponent e, modulus n, and the results of modular exponentiation.

2.3.3. Verifiability

Users can verify the results returned by both servers, output1 and output2. If a malicious server provides incorrect calculations, the system must ensure that users can identify the issue with certainty.

3. PREPARATORY KNOWLEDGE

In this section, we present the notations and some preliminary knowledge that needs to be clarified in the rest of our paper.

3.1. Notations

Table 1 lists the symbols that frequently appear in this paper and their explanations.

Table 1. Notations

Notations	Descriptions
$[x]_1$	An additive share of x
$\langle x \rangle_1$	A multiplication shares of x
$\mathbb{R}$	The set of real numbers
$\mathbb{Z}$	The set of integer numbers

3.2. Montgomery Form and Reduction Algorithm

For any integer N > 0, we choose an integer $R = 2^k > N$ satisfying gcd(R, N) = 1, the Montgomery form of any integer $x \in \mathbb{Z}_N$ is defined as $\bar{x} = xR \bmod N$. For any two integers $x, y \in \mathbb{Z}_N$, the celebrated Montgomery reduction algorithm introduced in 1985 by Montgomery [32] and also known as Montgomery modular multiplication is a method for performing fast modular multiplication xy mod N. Precisely, as shown in Algorithm 1, given two Montgomery forms $\bar{x} = xR \bmod N$ and $\bar{y} = yR \bmod N$, the Montgomery reduction algorithm can efficiently calculates $\bar{z} = xyR \bmod N$ which is the Montgomery form of the product xy.

Algorithm 1 MontRedc(R, N, N', x, y)

Input: x̄, ȳ and N, N', R satisfying $N' = -N^{-1} \bmod R$

Output: $\bar{z} = xyR \bmod N$.

1: $z \leftarrow \bar{x} \cdot \bar{y}$.

2: $\bar{z} \leftarrow [z \cdot N' \bmod R] \cdot N/R$.

3: if $\bar{z} > N$ then

4: $\bar{z} = \bar{z} - N$.

5: return z̄.

3.3. Modular Exponentiation

The Montgomery multiplication algorithm is particularly suitable for scenarios involving multiple modular mul tiplications with the same modulus. This is especially applicable in cases requiring computation of modular expo nentiation, such as calculating $a^e \bmod n$. Instead of direct exponentiation, a series of squaring and multiplication operations modulo n can be used. In $Algorithm$ 2, we summarize the modular exponentiation performed using the Montgomery product function, Algorithm 1. The exponentiation algorithm employs the binary method.

3.4. Secure Multi-Party Computation (MPC) Protocols

3.4.1. Secure Multiplication Protocol

In order to support multiplication calculations based on additive shares, Beaver Creatively proposed a technique called Beavertriplet [33]. The main idea is to introduce an offline stage to generate in advance A triplet $\{a, b, c \mid c = ab\}$. Then, according to the following equation

$$(x - a)(y - b) = xy - a(y - b) - b(x - a) - ab$$

Algorithm 2 MonExp(a, e, N)

Input: a,e,N

Output: $x = a^e mod\ N$.

1: Find an integer R such that $2^{k-1} \leq N < 2^k = R$.

2: Calculate $N' = -N^{-1} \mathrm{mod} R$.

3: $e = (e_{\ell-1} e_{\ell-2} \cdots e_0)_2$.

4: $\bar{a} \leftarrow a \cdot R \bmod N$.

5: $\bar{x} \leftarrow 1 \cdot R \bmod N$.

6: **for** $i = \ell - 1$ down to 0 **do**

7: $\bar{x} \leftarrow \mathrm{MonRec}(N', N, \bar{x}, \bar{x}, R)$.

8: if $e_i = 1$ then

9: $\bar{x} \leftarrow \mathrm{MonRec}(N', N, \bar{a}, \bar{x}, R)$.

10: $x \leftarrow \mathrm{MontRec}(N', R, N, \bar{x}, 1)$.

11: return x.

where $x, y, a, b \in R$, the information of x and y will be covered by the triple, but the additive shares of xy can be calculated. As shown in Algorithm 3, a trustworthy party T is utilized to undertake the task of generating random numbers during the offline phase. Each server has one additive share of x and y. Then, the servers will use a_i and b_i to cover its shares and expose e and f. Each server can use $[a]_i$, $[b]_i$, and $[c]_i$ to calculate the additive share of xy with ef.

Algorithm 3 SecMul($[x]_i$,$[y]_i$)

Input: S_i has $[x]_i$,$[y]_i$

Output: S_i has $[xy]_i$

Offline Phase:

1: T generates random numbers $a, b \in \mathbb{R}$,then computes $c = ab$.

2: T randomly splits a,b,c into n additive shares ($[a]_i$,$[b]_i$,$[c]_i$) and send them to S_i.

Online Phase:

1: S_i computes $e_i = [x]_i - [a]_i$ and $f_i = [y]_i - [b]_i$.

2: S_i collaboratively recovers e and f.

3: S_i computes $[xy]_i = [c]_i + [b]_i e + [a]_i f$.

4: s_1 computes $[xy]1 = [xy]_1 + ef$.

Algorithm 4 SecCmp($[x]_i$,$[y]_i$)

Input: S_i has $[x]_i$,$[y]_i$

Output: S_i gets $sgn(x - y)$.

Offline Phase:

1: T generates random positive number t,then computes the additive shares $[t]_i$ and send them to corresponding S_i.

2: T generates enough random numbers that the sub-protocol uses and sends them to S_i.

Online Phase:

1: S_i computes $[\alpha]_i = [x]_i - [y]_i$.

2: S_i collaboratively compute the $[t\alpha]i = \mathrm{SecMul}([t]_i,[\alpha]_i)$.

3: S_1 collaboratively recover tα, the sign of tα is equal to sgn(x − y).

3.4.2. Secure Comparison Protocol

For the comparison, we just need the sign information of the difference between two inputs. As we all know, the multiplication with a positive number will not change the sign of the original number. Generally, the difference between two numbers will reveal no information about the numbers. Here, we let T provide a random positive number to obscure (transform) the difference by SecMul. Then, the servers can reveal the obscured difference for further computation [34]. The comparison protocol is presented in Algorithm 4.

4. ALGORITHM FOR OUTSOURCING MODULAR EXPONENTIATION OPERATION

4.1. Computational Task Description and Basic Ideas

Given a, e, and N, the user aims to compute $a^e \ mod \ N$. Traditional approaches to solving this problem include methods like fast modular exponentiation, Montgomery modular exponentiation, and double modular exponentiation. However, due to the involvement of large integers, often hundreds or thousands of digits long, in cryptographic applications, users with limited resources find such operations prohibitively expensive. In such scenarios, secure outsourcing of computations can be employed to handle this task. In modular exponentiation, numerous division operations occur alongside multiple modular multiplications using the same modulus. We have found that Montgomery modular exponentiation is particularly suitable for this task. This method enhances the efficiency of large number modular computations through the use of Montgomery multiplication. By employing preprocessing and transformation techniques, Montgomery's algorithm makes modular multiplication more efficient, which is especially beneficial for handling large number computations in embedded systems. It significantly improves the performance of modular exponentiation, particularly when extensive repetitive calculations are involved. However, in the context of secure outsourcing protocols, protecting the privacy of the base, exponent, and modulus is crucial. If these elements are not adequately safeguarded, edge servers might gain access to substantial amounts of sensitive information. Therefore, we propose utilizing secure multi-party computation techniques and have designed a series of protocols to facilitate secure outsourcing of modular exponentiation tasks.

4.2. Secure Two-party Modular Multiplication Protocols

Given a large integer N and two integers $x, y \in \mathbb{Z}_N$, computing xy mod N is a fundamental task in primality testing. The operation xy mod $N = xy - \lfloor \frac{xy}{N} \rfloor N$ is frequently utilized for this purpose. To perform xy mod N se curely while preserving the confidentiality of the input data, a straightforward approach employing secure multiparty computation techniques involves protocols for secure multiplication and secure division. Nonetheless, secure division entails considerable communication overhead and computational delay. Fortunately, the Montgomery modular multi plication algorithm (Algorithm 1) provides an efficient method for modular multiplication that obviates the need for division. In this section, we build on the Montgomery modular multiplication algorithm to introduce a derivative of our primary framework: designs for secure multi-party modular multiplication. For given integers x,y, N, we begin by selecting a large number $R = 2^k > N$ and calculating $N' = -N^{-1} \ mod \ R, R' = R^{-1} \ mod \ N$. We then present four secure two-party modular multiplication protocols tailored to various application contexts:

Algorithm 5 SecMontRec($[N]_i$,$[N']_i$, x,y,R)

Input: S_i : $[N]_i$,$[N']_i$, x,y,R.

Output: S_i : $[xyR^{-1} \bmod N]_i$.

Offline Phase:

1: T generates enough random numbers that the sub-protocol uses and sends them to S_i

Online Phase:

1: S_1 computes the z = xy.

2: S_1 randomly generates addition shares $[z]_1$,$[z]_2$ and sends $[z]_2$ to S_2.

3: S_1 and S_2 collaboratively compute $[zN']_i$ = SecMul($[z]_i$,$[N']_i$).

4: S_i computes $[c]_i = [zN']_i \bmod R$.

5: S1 and S2 collaboratively compute the $[cN]_i$ = SecMul($[c]_i$,$[N]_i$).

6: S_i computes $[d]_i = \frac{[z]_i + [cN]_i}{R}$.

7: S_i gets sgn(d − N) from SecCmp($[d]_i$,$[N]_i$).

8: if sgn > 0 then

9: S_i computes $[h]_i = [d]_i - [N]_i$.

10: else

11: S_i sets $[h]_i = [d]_i$.

12: S_i gets sgn(h − N) from SecCmp($[h]_i$,$[N]_i$).

13: if sgn > 0 then

14: S_i computes $[h]_i = [h]_i - [N]_i$.

15: **return** $[h]_i$.

(1) Protocol I: Protecting the confidentiality of N while permitting flexibility in other parameters. In this context, the inputs are R, x, y, and addition shares $[N]_i$, $[N']_i$ (i = 1,2) corresponding to N and N′. The output consists of addition shares $[z]_i$ for $z = xyR' \bmod N$. Alternatively, if we use the Montgomery representations $\bar{x} = xR \bmod N$ and $\bar{y} = yR \bmod N$ instead of x and y, the output will be addition shares $[\bar{z}]_i$ where $\bar{z} = xyR \bmod N$. Details of this secure method are presented in Algorithm 5.

(2) Protocol II: Protecting the privacy of either (x, N) or (y, N), but not both simultaneously. Without loss of generality, assume we focus on protecting (y, N). The inputs are R, x, and addition shares $[y]_i$, $[N]_i$, $[N']_i$ (i = 1,2) corresponding to y, N, and N′. The output will be addition shares $[z]_i$ for $z = xyR' \bmod N$. Similarly to Protocol I, if we substitute x and $[y]_i$ with Montgomery forms $\bar{x} = xR \bmod N$ and addition shares $[\bar{y}]_i$ for $\bar{y}$ = yR mod N, the output will be addition shares $[\bar{z}]_i$ where $\bar{z}$ = xyR mod N.

(3) Protocol III: Focuses on preserving the privacy of (x,y). In this scenario, the algorithm takes as input R, N, N′, and the additive shares $[x]_i$, $[y]_i$ (where i = 1,2), and outputs the additive shares $[z]_i$, where $z = xyR' \bmod N$. Similar to the previous protocols, if $[x]_i$ and $[y]_i$ are in Montgomery form, i.e., $x = xR \bmod N$ and $y = yR \bmod N$, the output will be the additive shares $[z]_i$ such that $z = xyR \bmod N$.

(4) Protocol IIII: Ensuring the confidentiality of (x,y, N). For this scenario, the inputs are R and addition shares $[x]_i$, $[y]_i$, $[N]_i$, $[N']_i$ corresponding to x, y, N, and N′. The output includes addition

shares $[z]_i$ for $z = xyR' \, mod \, N$. Alternatively, if Montgomery forms $[\bar{x}]_i$ and $[\bar{y}]_i$ of $\bar{x}$ = xR mod N and $\bar{y}$ = yR mod N are used, the output will be addition shares $[\bar{z}]_i$ where $\bar{z}$ = xyR mod N.

It is important to observe that, in the subsequent discussions, we refer to the same algorithm name for the three protocols mentioned above, distinguishing them by the nature of their input data.

4.3. Secure Two-party Exponentiation Protocol

Algorithm 6 SecModExp($[N]_i$,$[N']_i$,$[\bar{1}]_i$,$[\bar{a}]_i$,$[\bar{b}]_i$,e,R)

Input: S_i : $[N]_i$,$[N']_i$,$[\bar{1}]_i$,$[\bar{a}]_i$,$[\bar{b}]_i$,e,R.

Output: S_i : $[\bar{x}]_i = [a^e R \mathrm{mod} N]_i$ and $[\bar{z}]_i = [b^e R \mathrm{mod} N]_i$.

1: S_i computes the binary representation of $e = (e_{k-1} e_{k-2} \ldots e_0)_2$.

2: S_i computes $[\bar{x}]_i \leftarrow [\bar{1}]_i$ and $[\bar{z}]_i \leftarrow [\bar{1}]_i$.

3: **for** $\ell = k - 1$ down to 0 **do**

4: S_i collaboratively computes $[\bar{x}]_i \leftarrow$ SecMontRec($[N']_i$,$[N]_i$,$[\bar{x}]_i$, $[\bar{x}]_i$,R).

5: S_i collaboratively computes $[\bar{z}]_i \leftarrow$ SecMontRec($[N']_i$,$[N]_i$,$[\bar{z}]_i$,$[\bar{z}]_i$,R).

8: if $e_l = 1$ then

9: S_i collaboratively computes $[\bar{x}]_i \leftarrow$ SecMontRec($[N']_i$,$[N]_i$,$[\bar{x}]_i$, $[\bar{a}]_i$,R).

5: S_i collaboratively computes $[\bar{z}]_i \leftarrow$ SecMontRec($[N']_i$,$[N]_i$,$[\bar{z}]_i$,$[\bar{b}]_i$,R).

11: **return** $[\bar{x}]_i and \, [\bar{z}]_i$.

Building upon the modular exponentiation algorithm Algorithm2 and the secure two-party Montgomery modular multiplication algorithm (Algorithm 5), this section will illustrate how to securely compute ae mod N while preserving the privacy of the modulus, within the context of two non-colluding HTC servers. The details of our secure multiparty computation design are shown in Algorithm 6. Out of which, $R = 2^k > N > 2^{k-1}$, $\bar{1}$ = R mod N, $\bar{a}$ = aR mod N and $1 \leq e \leq N^{-1}$, $[N]_i$,$[N']_i$,$[\bar{1}]$i are additional shares of N,N′ and $\bar{1}$, respectively.

5. OUR MAIN CONSTRUCTION

In this section, we present our proposed design for securely computing au mod N while preserving the privacy of a, u, and N. Our approach can be viewed as a secure multiparty implementation of the modular exponentiation algorithm utilizing Montgomery multiplication. Specifically, the protocol is composed of the following four phases:

5.1. DataBlinding Stage

In this phase, the user is required to generate the necessary data for the interaction stage and apply blinding encryption to any sensitive information that requires confidentiality. Subsequently, the blinded data is distributed to various servers. Given a,u, N ∈ Z, the algorithm outputs the following values: [a]i,[N]i,[N′]i,[1]i,e,R. For further details, refer to Algorithm 7.

5.2. ServerInteraction Stage

After the data blinding phase, the two edge cloud servers receive the user-sent data, which includes $[a]_i$, $[N]_i$, $[N']_i$, $[1]_i$, e, and R. The servers then collaborate to compute the Montgomery form of the modular exponentiation result $[x]_i = [a^e R \bmod N]$ as outlined in Algorithm 6. To restore the original

format, Algorithm 5 must be executed, yielding additive shares of ae mod N. This result is subsequently sent back to the user. See Algorithm 8 for further details.

Algorithm 7 DataBlinding(a,u,N)

Input: a,u,N.

Output: S_i gets $[\bar{1}]_i$,$[\bar{a}]_i$,$[\bar{b}]_i$, $[u]_i$, $[N]_i$, $[N']_i$, e, R.

1: Calculate an integer R that satisfies $R = 2^k > N$.

2: Calculate $N' \leftarrow -N^{-1} \, mod \, R$.

3: Calculate $\bar{1} \leftarrow$ R mod N.

4: Calculate $\bar{a} \leftarrow a \cdot R \bmod N$.

5: Calculate $r = u \bmod 2^{32}$,e = u−r.

6: Randomly generate addition shares$[N]_1$,$[N]_2$,satisfy $N = [N]_1 + [N]_2$.

7: Randomly generate addition shares$[N']_1$,$[N']_2$,satisfy $N' = [N']_1 + [N']_2$.

8: Randomly generate addition shares$[\bar{1}]_1$,$[\bar{1}]_2$,satisfy $\bar{1} = [\bar{1}]_1 + [\bar{1}]_2$.

9: Randomly generate addition shares$[\bar{a}]_1$,$[\bar{a}]_2$,satisfy $\bar{a} = [\bar{a}]_1 + [\bar{a}]_2$.

10: Randomly generate addition shares$[\bar{b}]_1$,$[\bar{b}]_2$,satisfy $\bar{a} = [\bar{b}_1 + [\bar{b}]_2$.

11: Send $[\bar{1}]_i$,$[\bar{a}]_i$,$[\bar{b}]_i$, $[u]_i$, $[N]_i$, $[N']_i$,e, R to S_i.

Algorithm 8 ServerInter($[N]_i$,$[N']_i$,$[\bar{1}]_i$,$[\bar{a}]_i$,$[\bar{b}]_i$,e, R)

Input: S_i : $[N]_i$,$[N']_i$,$[\bar{1}]_i$,$[\bar{a}]_i$,$[\bar{b}]_i$,e,R.

Output: S_i : $[C]_i = [a^e R \bmod N]_i$ and $[D]_i = [b^e R \bmod N]_i$.

1: S_1 and S_2 collaboratively computes $[\bar{x}]_i$,$[\bar{z}]_i \leftarrow$ SecModExp($[N]_i$,$[N']_i$,$[\bar{1}]_i$,$[\bar{a}]_i$,$[\bar{b}]_i$,e, R).

2: S_1 and S_2 collaboratively computes $[C]_i \leftarrow$ SecMontRec([N′]i,[N]i,$[\bar{x}]_i$,1,R)

3: S_1 and S_2 collaboratively computes $[D]_i \leftarrow$ SecMontRec([N′]i,[N]i,$[\bar{z}]_i$,1,R)

4: Si send $[C]_i$,$[D]_i$ to Client.

5.3. Decryption Stage

Algorithm 9 Decryption($[C]_i$,a,r, N)

Input: $[C]_i$,a,r, N.

Output: $Y = a^u \bmod N$.

1: Calculate $C = [C]_1 + [C]_2$.

2: Calculate y ← MonExp(a,r,N).

3: Calculate $Y = C \cdot y$.

Algorithm 10 Validation([C]i,[D]i)

Input: $[C]_i$,$[D]_i$.

Output: True or False.

1: Calculate C = [C]1 + [C]2.

2: Calculate D = [D]1 + [D]2.

3: **if** C = D **then**

4: Outputs “True”.

5: **else**

6: Outputs “False”.

After the interaction between the edge servers is completed, the client receives $[C]_i$. At this point, C (where $C = [C]_1 + [C]_2$) is not the final result but represents only a part of the final output, specifically a multiplicative share. The remaining part is held by the client, who needs to compute $y = a^r \bmod N$ independently. For a detailed description of the algorithm, refer to Algorithm 9.

5.4. Validation Stage

In the verification phase, after receiving $[C]_i$, the client records C_1 (where $C_1 = [C]_1 + [C]_2$). Next, we rerun the data blinding and interaction phases, keeping the input for the blinding phase unchanged. Once these phases are completed, $[C]_i$ is updated, and we obtain C_2 (where $C_2 = [C]_1 + [C]_2$). The correctness of the result is verified by comparing C_1 and C_2. If they are equal, the result is considered correct; otherwise, the client outputs “False”. See Algorithm 10 for details.

6. PRACTICAL PERFORMANCE EVALUATION

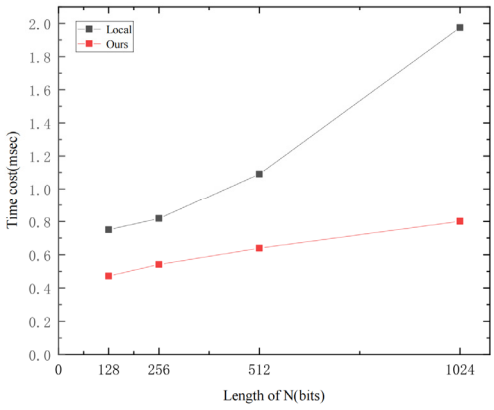

Figure 2. Comparison of time cost.

Our experiments were conducted on a Windows 11 machine equipped with an AMD Ryzen 7 6800H processor featuring NVIDIA GeForce RTX 3060 and 16 GB of memory.

In our experiment, we conducted experiments by implementing the scheme on different modules ranging from 128 to 1024 bits. The radix was set as the "plaintext", the base (a) was set as a 64-bit random number, and the exponent (e) was set as 65537 (a commonly used public key in RSA). To ensure a more accurate measurement of the computation time, we performed 20 trials for a specific bit size of the modulus, and then calculated their average values. As shown in Figure 2, when calculating ($a^e \bmod N$), our outsourcing scheme significantly outperformed the case without

outsourcing. As the number of bits of the modulus increases, the growth rate of the time consumed by the non-outsourcing scheme is notably faster than that of the outsourcing scheme. The larger the number of bits of the modulus that needs to be calculated, the more resources and energy the scheme can save.

7. CONCLUSION

This paper focuses on the research of modular exponentiation outsourcing. Firstly, it introduces the rise of outsourced computation in the context of the development of cloud computing and cluster computing, as well as the security challenges such as privacy, verifiability, and efficiency faced by modular exponentiation outsourcing. By reviewing related work, the advantages and disadvantages of two types of outsourcing schemes based on Two - Untrusted - Server (TUS) and Single - Untrusted - Server (SUS) are analyzed. On this basis, a privacy - preserving modular exponentiation outsourcing scheme based on secure multi - party computation for a dual - server model is proposed. This scheme has features such as protecting the entire process of the algorithm, data privacy, and enhancing computational efficiency. Then, the system model, threat model, and design goals are elaborated in detail, and preparatory knowledge including notations and the Montgomery algorithm is introduced. Subsequently, the correctness and security of the proposed scheme are analyzed, and experimental comparisons are carried out. The research results show that this scheme can effectively address the security challenges faced by outsourced computation. While ensuring accurate and secure computations, it improves computational efficiency, providing a reliable new option for modular exponentiation outsourcing and showing potential for practical applications.

REFERENCES

[1] S. Subashini, V. Kavitha, A survey on security issues in service delivery models of cloud computing, Journal of Network and Computer Applications 34 (2011) 1–11.

[2] D. Zissis, D. Lekkas, Addressing cloud computing security issues, Future Generation Computer Systems 28 (2012) 583–592.

[3] J. Wang, H. Ma, Q. Tang, J. Wang, Efficient verifiable fuzzy keyword search over encrypted data in cloud computing, Computer Science and Information Systems 10 (2013) 667–684.

[4] X. Chen, J. Li, J. Ma, et al., New algorithms for secure outsourcing of modular exponentiations, IEEE Transactions on Parallel and Distributed Systems 25 (2013) 2386–2396.

[5] Y. Ren, N. Ding, X. Zhang, et al., Verifiable outsourcing algorithms for modular exponentiations with improved checkability, in: Proceedings of the 11th ACM on Asia Conference on Computer and Communications Security, Association for Computing Machinery, ACM, New York, NY, USA, 2016, pp. 293–303.

[6] L. Kuppusamy, J. Rangasamy, Crt-based outsourcing algorithms for modular exponentiations, in: International Conference on Cryptology in India, Springer, Springer, Cham, 2016, pp. 81–98.

[7] S. Hohenberger, A. Lysyanskaya, How to securely outsource cryptographic computations, in: Theory of Cryptography Conference, Springer, Berlin, Heidelberg, 2005, pp. 264–282.

[8] X. Ma, J. Li, F. Zhang, Outsourcing computation of modular exponentiations in cloud computing, Cluster Computing 16 (2013) 787–796.

[9] J. Ye, J. Wang, Secure outsourcing of modular exponentiation with single untrusted server, in: Proceedings of the 2015 18th International Conference on Network-Based Information Systems, IEEE, Taiwan, 2015, pp. 643–645.

[10] J. Liu, B. Yang, A new algorithm for secure outsourcing composite modular exponentiation, in: Proceedings of the 2015 2nd International Conference on Information Science and Control Engineering, IEEE, Shanghai, China, 2015, pp. 461–465.

[11] Y. Ma, H. Tian, B. Wei, Secure modular exponentiation outsource with two untrusted programs and improved checkability, Journal of Information Science & Engineering 32 (2016).

[12] R. Yanli, D. Min, Q. Zhenxing, Z. Xinpeng, F. Guorui, Efficient algorithm for secure outsourcing of modular exponentiation with single server, IEEE Transactions on Cloud Computing PP (2018) 1–1.

[13] O. Uzunkol, J. Rangasamy, L. Kuppusamy, Hide the modulus: A secure non-interactive fully verifiable delegation scheme for modular exponentiations via crt, in: Proceedings of the International Conference on Information Security,

Springer, Cham, 2018, pp. 250–267.

[14] M. V. Dijk, D. Clarke, B. Gassend, et al., Speeding up exponentiation using an untrusted computational resource, Designs, Codes and Cryptography 39 (2006) 253–273.

[15] Y. Wang, Q. Wu, D. S. Wong, et al., Securely outsourcing exponentiations with single untrusted program for cloud storage, in: European Symposium on Research in Computer Security, Springer, Cham, 2014, pp. 326–343.

[16] C. Chevalier, F. Laguillaumie, D. Vergnaud, Privately outsourcing exponentiation to a single server: Cryptanalysis and optimal constructions, Algorithmica (2020) 1–44.

[17] J. Liu, B. Yang, Z. Du, Outsourcing of verifiable composite modular exponentiations, in: Proceedings of the 5th International Conference on Intelligent Networking and Collaborative Systems, IEEE, Xi'an, China, 2013, pp. 546–551. 15.

[18] C. Xiang, C. Tang, Verifiable and Secure Outsourcing Schemes of Modular Exponentiations Using One Untrusted Cloud Server and Their Application, Technical Report 2014/500, IACR Cryptology ePrint Archive, 2014. URL: https://eprint.iacr.org/2014/500.pdf.

[19] C.Xiang, C. Tang, Efficientoutsourcing schemes of modular exponentiations with checkability for untrusted cloud server, Journal of Ambient Intelligence and Humanized Computing 6 (2015) 131–139.

[20] Y. Ding, Z. Xu, J. Ye, et al., Secure outsourcing of modular exponentiations under single untrusted programme model, Journal of Computer and System Sciences 90 (2017) 1–13.

[21] J. Cai, Y. Ren, T. Jiang, Verifiable outsourcing computation of modular exponentiations with single server, IJ Network Security 19 (2017) 449–457.

[22] K. Zhou, M. H. Afifi, J. Ren, Expsos: Secure and verifiable outsourcing of exponentiation operations for mobile cloud computing, IEEE Transactions on Information Forensics and Security 12 (2017) 2518–2531.

[23] S. Li, L. Huang, A. Fu, et al., Cexp: Secure and verifiable outsourcing of composite modular exponentiation with single untrusted server, Digital Communications and Networks 3 (2017) 236–241.

[24] J. Rangasamy, L. Kuppusamy, Revisiting single-server algorithms for outsourcing modular exponentiation, in: International Conference on Cryptology in India, Springer, 2018, pp. 3–20.

[25] Y. Zhu, A. Fu, S. Yu, et al., New algorithm for secure outsourcing of modular exponentiation with optimal checkability based on single untrusted server, in: 2018 IEEE International Conference on Communications (ICC), IEEE, Kansas City, MO, 2018, pp. 1–6.

[26] A. Fu, Y. Zhu, G. Yang, et al., Secure outsourcing algorithms of modular exponentiations with optimal checkability based on a single untrusted cloud server, Cluster Computing 21 (2018) 1933–1947.

[27] A. Fu, S. Li, S. Yu, et al., Privacy-preserving composite modular exponentiation outsourcing with optimal checkability in single untrusted cloud server, Journal of Network and Computer Applications 118 (2018) 102–112.

[28] Q. Su, R. Zhang, R. Xue, Secure outsourcing algorithms for composite modular exponentiation based on single untrusted cloud, The Computer Journal 63 (2020) 1271–1271.

[29] Q. Hu, M. Duan, Z. Yang, et al., Efficient parallel secure outsourcing of modular exponentiation to cloud for iot applications, IEEE Internet of Things Journal (2020).

[30] D. Xu, Y. Ren, X. Li, et al., Efficient and secure outsourcing of modular exponentiation based on smart contract, International Journal of Network Security 22 (2020) 934–944.

[31] H. Li, J. Yu, H. Zhang, et al., Privacy-preserving and distributed algorithms for modular exponentiation in iot with edge computing assistance, IEEE Internet of Things Journal 7 (2020) 8769–8779.

[32] P. L. Montgomery, Modular multiplication without trial division, Mathematics of Computation 44 (1985) 519–521. URL: https://api. semanticscholar.org/CorpusID:119574413.

[33] D. Beaver, Efficient multiparty protocols using circuit randomization, in: Advances in Cryptology—CRYPTO'91: Proceedings 11, Springer, 1992, pp. 420–432.

[34] Z. Xia, Q. Gu, W. Zhou, L. Xiong, J. Weng, N. Xiong, Str: Secure computation on additive shares using the share-transform-reveal strategy, IEEE Transactions on Computers (2021)