Efficient Algorithm for Secure Outsourcing of Modular Exponentiation with Single Server

Yanli Ren*, Min Dong, Zhenxing Qian, Xinpeng Zhang, Guorui Feng

Abstract—Outsourcing computation allows an outsourcer with limited resource to delegate the computation load to a powerful server without the exposure of true inputs and outputs. It is well known that modular exponentiation is one of the most expensive operations in public key cryptosystems. Currently, most of outsourcing algorithms for modular exponentiation are based on two untrusted servers or have small checkability with single server. In this paper, we first propose an efficient outsourcing algorithm of modular exponentiation based on two untrusted servers, where the outsourcer can detect the error based on Euler theorem with a probability of 1 if one of the servers misbehaves. We then present an outsourcing algorithm of modular exponentiation with single server, and the outsourcer can also check the failure with a probability of 1. Therefore, the proposed algorithm with single server improves efficiency and checkability simultaneously compare with the previous ones. Finally, we provide the experimental evaluations to demonstrate that the proposed two algorithms are the most efficient ones in all of the outsourcing algorithms for an outsourcer.

Index Terms—Cloud computing, Verifiable outsourcing computation, Modular exponentiation, Euler theorem

1 Introduction

WITH the development of cloud computing, outsourcing computation has attracted widespread attention from academic world and industrial community. In the algorithms of outsourcing computation, the outsourcer with limited resources can delegate expensive computational work to powerful servers. Outsourcing computation realizes full utilization of resource and a lot of computation time and cost can be saved for the outsourcer with limited ability.

Despite of many tremendous advantages, outsourcing computation also meets some security challenges. At the era of cloud computing, it is very difficult to find a cloud server which is absolutely trusted for an outsourcer. Firstly, the computation tasks always contain some sensitive information and the outsourcer does not want to disclose them to the servers [1]. Therefore, the outsourcer should ensure the secrecy of inputs and outputs before it delegates the computation task to the servers, which means sensitive information of the outsourcer will not be exposed during the phase of outsourcing computation. The second problem the outsourcer should consider is how to detect the server's failures with high probability since the servers are potentially malicious, for example, they may be controlled by an adversary and return invalid results to the outsourcer. The outsourcer should verify the outsourcing result returned by the server effectively and refuses it once the server misbehaves. The third challenge is that the test procedure should not include complicated operations because the outsourcer will execute the procedure when it receives the outsourcing result. The algorithm of outsourcing computation is meaningless if the test procedure is too complicated for the outsourcer with limited ability. Therefore, secure outsourcing computation should simultaneously ensure secrecy of sensitive information, verifiability of the outsourcing result, and efficiency of the testing phase.

In the cryptographic community, it is common to outsource some expensive computations to semi-trusted devices. Chaum *et al.* [2] first introduced the concept of "wallets with observers" that allowed a piece of hardware installed on the outsourcer's device to carry out some expensive computations for each transaction. Hohenberger and Lysyanskaya [3] formalized this model and presented an algorithm for the computation of modular exponentiations with two non-colluding servers. Chen *et al.* proposed new outsourcing algorithms for modular exponentiations with improved checkability and efficiency [4], and they also presented the first efficient outsourcing algorithm for bilinear pairing [5] in the one-malicious version of the two untrusted program model. Other work for secure outsourcing computation includes attributed-based cryptosysytem with outsourcing decryption [6] and signature [7], identity-based encryption with outsourcing revocation [8], high-dimensional image feature extraction [9], and so on.

It is well known that modular exponentiation (MExp) is one of the most expensive computations in public key cryptography, and it takes roughly 1.5L modular multiplications (MMs) to compute single modular exponentiation by the square-and-multiply method, where L is the bit length of the exponent. Many outsourcing algorithms of modular exponentiations have been proposed until now. Hohenberger *et al.* [3] first proposed secure outsourcing algorithm of modular exponentiations based on the help of servers [10-12] or precomputations [13-15], and they also gave the formal definition and security model of outsourcing computation. In their model, the outsourcer could detect the error with a probability of $1/2$,

*Corresponding author

- *Y. Ren, M. Dong, Z.Qian, X. Zhang, and G. Feng are with School of Communication and Information Engineering, Shanghai University, China. E-mail: renyanli@shu.edu.cn, dongmin3524@ i.shu.edu.cn, {zxqian, xzhang, grfeng}@ shu.edu.cn.*

i.e., it might be cheated by the untrusted servers with a probability of 1/2. Then Chen *et al.* [4] proposed new outsourcing algorithm of MExp in the one-malicious version of two untrusted servers and compared its efficiency with that of the algorithm in [3]. The comparison result shows that Chen *et al.*'s scheme is better in both efficiency and checkability, and the checkability is improved to $2/3$. However, the outsourcer cannot detect all of the errors if the servers misbehave in the above two algorithms. Further, Ma *et al.* [16] proposed an algorithm for outsourcing batch MExps based on two servers, but the scheme might disclose some information about the inputs. Ren *et al.* [17] proposed an outsourcing algorithm of MExps based on two servers, where the fault could be detected with a probability of 1, but the outsourcer needed to interact with the servers for two times and some communication cost was appended. Based on Chinese remainder theorem (CRT), Kuppusamy *et al.* [18] then presented new outsourcing algorithms of MExp, which was non-interactive and the outsourcer could verify the computational results with a probability of 1. However, the precomputation is so complicated which requires 10 MMs and 3 MInvs.

Besides the above algorithms based on two servers, Dijk *et al.* [19] presented an outsourcing algorithm for MExp with single server, but it could not ensure the privacy of the inputs. In addition, Wang *et al.* [20] also presented an algorithm for outsourcing MExps with single server, but it was not efficient and the checkability was only $1/2$ for outsourcing single MExp. Then some researchers tried to improve efficiency or checkability for outsourcing MExps based on single server while keeping the inputs and outputs undisclosed. Kiraz *et al.* [21] proposed several algorithms for outsourcing MExp with single server, where the checkability was improved, but it was still possible for the outsourcer to be cheated by the server. Cai *et al.* [22] also proposed an outsourcing algorithm of MExp based on single server, but it was not efficient and the outsourcer needed to make a lot of queries for the server. Ye *et al.* described two outsourcing algorithms of MExps based on two servers or single server respectively [23-24], which improved the checkability greatly but much computation cost was appended for the outsourcer.

Our Contributions. In this paper, we present two fully verifiable outsourcing algorithms of MExp based on Euler Theorem. Firstly, we propose an outsourcing algorithm in the one-malicious model of two untrusted servers, and the outsourcer can detect the errors made by the dishonest server with a probability of 1. Compared with the previous algorithms, the proposed one is better in both checkability and efficiency without interaction between the outsourcer and the servers. We also present secure outsourcing algorithm of MExp with full verifiability based on single server, which is the most important contribution of this paper. The second proposed algorithm improves efficiency and checkability simultaneously compared with the previous ones with single server. At the end of this paper, we give the detailed theoretical and experimental analysis, which also show the advantage of the proposed two algorithms in checkability, communication and computation efficiency.

1.1 Organization

The rest of this paper is organized as follows: In Section 2, we give basic definitions and security model of outsourcing computation. The proposed two outsourcing algorithms of MExp and their security analysis, and efficiency comparison are presented in Section 3 and Section 4, respectively. The experimental evaluations of two algorithms are given in Section 5. Finally, we make the conclusions in Section 6.

2 Definitions and Security Model

In this section, we introduce some definitions and security model of outsourcing computation and *Euler theorem* where the proposed algorithms are based on.

2.1 Definitions of Outsource-Security

Hohenberger and Lysyanskaya [3] first give the formal definitions of secure outsourcing computation, and we use their definitions in this paper.

Suppose that we have an algorithm *Alg*, and there are two parties in this algorithm. One is an outsourcer T (trusted and computationally limited) and the other is a server U (untrusted but computationally powerful), where T can make queries to U. When T and U implement *Alg*, there is a server U' instead of U who tries to work maliciously, and it records all the computations when it is invoked. We say (T, U) is an outsource-secure implementation of *Alg*, and U' learns nothing about the inputs and outputs of $T^{U'}$ during the phase of computation.

Definition 1 (Algorithm with Outsource-IO). *If an algorithm Alg takes five inputs and generates three outputs, we say that Alg obeys the outsourcing input/output specification. The inputs and outputs are all classified by how much the adversary* $A = (E, U')$ *knows about them, where E is the adversarial environment.*

Firstly, an honest party creates three inputs. The first one is called an honest, secret input, which is unknown for both E and U'; the second one is called an honest, protected input, which may be known by E, but is protected from U'; and the third one is called an honest, unprotected input, which may be known by both E and U'. In addition, E generates the last two inputs. An adversarial, protected input, which is known for E, but protected from U'; and another one is called an adversarial, unprotected input, which is known for both E and U'.

Similarly, the first output is secret which is unknown for both E and U'; the second output is protected, which may be known for E, but not U'; and the last one is unprotected, which may be known for both E and U'.

Definition 2 (Outsource-security). *We say a pair of algorithms* (T, U) *with outsourcing I/O is an outsource-secure implementation of Alg if:*

Correctness: T^U *is a correct implementation of Alg.*

Security: For all adversaries $A = (E, U')$, *there exist simulators* (S_1, S_2) *such that the following pairs of random variables are computationally blind.*

Pair One: $EVIEW_{real} \sim EVIEW_{ideal}$

The adversarial environment E gets the view by participating in the following real process:

$$EVIEW_{real}^{i} = \{(istate^{i}, x_{hs}^{i}, x_{hp}^{i}, x_{hu}^{i}) \leftarrow I(1^{k}, istate^{i-1});$$
$$(estate^{i}, j^{i}, x_{ap}^{i}, x_{au}^{i}, stop^{i}) \leftarrow E(1^{k}, EVIEW_{real}^{i-1}, x_{hp}^{i}, x_{hu}^{i});$$
$$(tstate^{i}, ustate^{i}, y_{s}^{i}, y_{p}^{i}, y_{u}^{i}) \leftarrow T^{U'(ustate^{i-1})} \times (tstate^{i-1}, x_{hs}^{j^{i}}, x_{hp}^{j^{i}}, x_{hu}^{j^{i}}, x_{ap}^{i}, x_{au}^{i}): (estate^{i}, y_{p}^{i}, y_{u}^{i})\}$$
$$EVIEW_{real} = EVIEW_{real}^{i} \textit{ if } stop^{i} = TRUE.$$

In round i of the real process, the honest, stateful process I picks the honest (secret, protected, and unprotected) inputs $(x_{hs}^{i}, x_{hp}^{i}, x_{hu}^{i})$ those are protected from the environment E. Then E can choose some variables based on its view from the last round: firstly, E sets the value of $estate_{i}$ to remember what it will do next time when it is invoked; the honest inputs $x_{hs}^{j^{i}}, x_{hp}^{j^{i}}, x_{hu}^{j^{i}}$ created previously which $T^{U'}$ has access to, and E can specify the index j^{i} of them but does not know their values; the adversarial, protected input x_{ap}^{i} and unprotected input x_{au}^{i}; finally, it chooses $stop^{i}$ to determine whether round i is the last round in this process.

Next, on the inputs $(tstate^{i-1}, x_{hs}^{j^{i}}, x_{hp}^{j^{i}}, x_{hu}^{j^{i}}, x_{ap}^{i}, x_{au}^{i})$, where $tstate^{i-1}$ is *T's* previous state, the algorithm $T^{U'}$ produces a new state $tstate^{i}$, the secret y_{s}^{i}, protected y_{p}^{i} and unprotected y_{u}^{i} outputs. The current state of U' is saved in the variable $ustate^{i}$ as the input which is given to U'. $(estate^{i}, y_{p}^{i}, y_{u}^{i})$ is the view of the real process in round *i*. The view in the last round is the overall view of the environment in the real process.

The ideal process:

$$EVIEW_{ideal}^{i} = \{(istate^{i}, x_{hs}^{i}, x_{hp}^{i}, x_{hu}^{i}) \leftarrow I(1^{k}, istate^{i-1});$$
$$(estate^{i}, j^{i}, x_{ap}^{i}, x_{au}^{i}, stop^{i}) \leftarrow E(1^{k}, EVIEW_{ideal}^{i-1}, x_{hp}^{i}, x_{hu}^{i});$$
$$(astate^{i}, y_{s}^{i}, y_{p}^{i}, y_{u}^{i}) \leftarrow Alg(astate^{i-1}, x_{hs}^{j^{i}}, x_{hp}^{j^{i}}, x_{hu}^{j^{i}}, x_{ap}^{i}, x_{au}^{i});$$
$$(sstate^{i}, ustate^{i}, Y_{p}^{i}, Y_{u}^{i}, replace^{i}) \leftarrow S_{1}^{U'(ustate^{i-1})} \times (sstate^{i-1}, x_{hp}^{j^{i}}, x_{hu}^{j^{i}}, x_{ap}^{i}, x_{au}^{i}, y_{p}^{i}, y_{u}^{i});$$
$$(z_{p}^{i}, z_{u}^{i}) = replace^{i}(Y_{p}^{i}, Y_{u}^{i}) + (1 - replace^{i})(y_{p}^{i}, y_{u}^{i}): (estate^{i}, z_{p}^{i}, z_{u}^{i})\}$$
$$EVIEW_{ideal} = EVIEW_{ideal}^{i} \textit{ if } stop^{i} = TRUE.$$

The ideal process also proceeds in rounds. In the ideal process, we have a stateful simulator S_{1} which knows nothing sensitive about the secret input x_{hs}^{i}, but given the non-secret outputs that *Alg* generates. There is a variable $replace^{i}$ that decides whether the real values (y_{p}^{i}, y_{u}^{i}) outputs will be replaced by some other values (Y_{p}^{i}, Y_{u}^{i}). In doing so, it is allowed to make queries to oracle U'; moreover, U' saves its state as in the real experiment.

Pair Two: $UVIEW_{real} \sim UVIEW_{ideal}$

The untrusted software U' gets the view by participating in the real process which is described in *Pair One*. $UVIEW_{real} = ustate^{i} \textit{ if } stop^{i} = TRUE.$

The ideal process:

$$UVIEW_{ideal}^{i} = \{(istate^{i}, x_{hs}^{i}, x_{hp}^{i}, x_{hu}^{i}) \leftarrow I(1^{k}, istate^{i-1});$$
$$(estate^{i}, j^{i}, x_{ap}^{i}, x_{au}^{i}, stop^{i}) \leftarrow E(1^{k}, estate^{i-1}, x_{hp}^{i}, x_{hu}^{i}, y_{p}^{i-1}, y_{u}^{i-1});$$
$$(astate^{i}, y_{s}^{i}, y_{p}^{i}, y_{u}^{i}) \leftarrow Alg(astate^{i-1}, x_{hs}^{j^{i}}, x_{hp}^{j^{i}}, x_{hu}^{j^{i}}, x_{ap}^{i}, x_{au}^{i});$$
$$(sstate^{i}, ustate^{i}) \leftarrow S_{2}^{U'(ustate^{i-1})} \times (sstate^{i-1}, x_{hu}^{j^{i}}, x_{au}^{i}): (ustate^{i})\}$$
$$UVIEW_{ideal} = UVIEW_{ideal}^{i} \textit{ if } stop^{i} = TRUE.$$

In the ideal process, we also have a stateful simulator S_{2} which only knows about the unprotected inputs (x_{hu}^{i}, x_{au}^{i}) and it can make queries to U'. As shown in *Pair One*, U' may maintain its state.

Definition 3 (α-Efficient, Secure Outsourcing). *If* T^{U} *is a correct implementation of Alg and the running time of T is no more than an α-multiplicative factor of the running time of Alg, we say* (T, U) *is an α-efficient implementation of Alg.*

Definition 4 (*β*-Checkable, Secure Outsourcing). *We say that algorithm* (T, U) *is a β-checkable implementation of Alg if it is a correct implementation of Alg and T will detect the error with a probability of no less than β if* U' *work maliciously.*

Definition 5 ((*α*, *β*)-Outsource-Security). (T, U) *is an* (α, β)*-outsource-secure implementation of Alg if it is both α-efficient and β-checkable.*

2.2 Security Model

We also use the security model of [3] that Hohenberger *et al.* presented. In their *two untrusted program model*, firstly, E writes the code for two untrusted servers U_{1}', U_{2}' and gives it to *T*, then *T* installs this software in a manner such that all communication between E and U_{1}', U_{2}' must pass through *T*. We can get the new adversary $A = (E, U_{1}', U_{2}')$.

Assume at most one of the servers works maliciously but we do not know which one, and then security means that there is a simulator for both conditions. This is the *one-malicious model* proposed in [3].

Remark 1. For the algorithm with single server, the security model is similar to *two untrusted program model,* the adversary is A $= (E, U')$ and the server is untrusted.

2.3 Euler Theorem

The verifiability of the proposed algorithm is obtained by using *Euler Theorem* [25]. Now we give its brief introduction, please see [25] for the detailed proof.

Theorem 2.1. Assume there are two positive integers x,y, and x is co-prime with y, we can get that

$$x^{\varphi(y)} \equiv 1(mod\ y),$$

where $\varphi(y)$ denotes the number of integers those are smaller than y and are co-prime with y, and this function is also called the *Euler function* of y.

3 Efficient Outsourcing of Modular Exponentiation with Two Servers

In this section, we first propose a verifiable outsourcing algorithm of modular exponentiation based on one-malicious model of two servers, and then give its security analysis and efficiency comparison with the previous ones.

Let p, q be two large primes and $q|p-1$. As in [3], we also need a subroutine called *Rand* to generate a random tuple with the form of $(c, c^{-1}, g^c\ mod\ p)$, where $c \in \mathbb{Z}_q$, so that the computations of the outsourcer can be speeded. The inputs for *Rand* are a large prime p and a base $g \in \mathbb{Z}_p^*$. In order to realize the subroutine, we can use the techniques named an EBPV generator [15] or a table-lookup method [26].

3.1 Outsourcing Algorithm

In the proposed algorithm, the inputs are $a \in \mathbb{Z}_q$ and $u \in \mathbb{Z}_p^*$ such that $u^q \equiv 1(mod\ p)$, and the output is $u^a\ mod\ p$. Note that both u and a are computationally blinded to the servers, and the servers output $y^x\ mod\ p$ when they receive the inputs (x, y). The details are as follows:

1. T firstly runs *Rand* twice to create two tuples $(\alpha, \alpha^{-1}, g^\alpha)$ and $(\beta, \beta^{-1}, g^\beta)$. We get the first logical division:
$$u^a = (vw)^a = g^{\alpha a} w^a = g^\beta g^\gamma w^a,$$
where $v = g^\alpha, w = u/v, \gamma = \alpha a - \beta$.
2. In order to hide the exponent a, T picks random numbers $t, h_1 \in \mathbb{Z}_q$, and a prime n which is co-prime with w, and then T gets s by computing $a \equiv \varphi(n)t + s(mod\ q)$, where $\varphi(n)$ is the *Euler function* of n, and $\varphi(n) = n-1$.
The second logical division is as follows:
$$u^a = g^\beta g^\gamma w^a = g^\beta g^\gamma w^{\varphi(n)t+s} = g^\beta g^\gamma w^s w^{h_1} w^{h_2},$$
where $h_2 = \varphi(n)t - h_1$.
3. Next T runs *Rand* to get a tuple $(t_1, {t_1}^{-1}, g^{t_1})$.
4. Then T makes queries to U_1 in random order:
$U_1(\gamma/t_1, g^{t_1}) \to g^\gamma$;
$U_1(h_1, w) \to w^{h_1}$;
$U_1(s, w) \to w^s$.
Similarly, T queries U_2 in random order:
$U_2(\gamma/t_1, g^{t_1}) \to g^\gamma$;
$U_2(h_2, w) \to w^{h_2}$;
$U_2(s, w) \to w^s$.

Finally, T checks whether both U_1 and U_2 produce the correct outputs, *i.e.*,

$$g^\gamma = U_1(\gamma/t_1, g^{t_1}) = U_2(\gamma/t_1, g^{t_1}),$$
$$w^s = U_1(s, w) = U_2(s, w),$$

and we know that from *Euler Theorem*

$$w^{h_1} w^{h_2} = w^{\varphi(n)t} \equiv 1(mod\ n).$$

If not, T outputs "error"; otherwise, T multiplies the outputs of U_1 and U_2 and obtains that

$$\begin{aligned} g^\beta g^\gamma w^s w^{h_1} w^{h_2} &\equiv g^\beta g^\gamma w^{\varphi(n)t+s} \\ &\equiv g^{\alpha a} w^a \\ &\equiv (vw)^a \equiv u^a (mod\ p). \end{aligned}$$

Remark 2. In our algorithm, T randomly chooses a prime n, and n is only used to check the results. Note that we set n as a prime in order to detect the failures made by two servers. The primes can be stored in a table and T can choose the prime from the table, or we can choose several primes those are all co-prime with w and we multiply them to get a new number. Obviously, the new number is also co-prime with w. On the other hand, the new number is the result multiplied by several primes, so it is easy to get the value of *Euler Function* of the new number.

3.2 Security Analysis

Theorem 3.1. *In the one-malicious model, if the input (a, u) is honest, secret; or honest, protected; or adversarial, protected, $(T, (U_1, U_2))$ is an outsource-secure implementation of the proposed algorithm.*

Proof. The proof is similar to [3-4]. The correctness of the proposed algorithm is shown in Section 3.1, and now we pay more attention to its security. Let $A = (E, U_1', U_2')$ be an adversary that interacts with algorithm in the one-malicious model of two untrusted servers.

Pair One: $EVIEW_{real} \sim EVIEW_{ideal}$, which means that the environment E learns nothing during the execution of $(T, (U_1, U_2))$. If the input (a, u) is honest, protected, or adversarial, protected, it is obvious that the simulator S_1 behaves same as in the real execution. Therefore, it only needs to prove the case where (a, u) is an honest, secret input.

If the input (a, u) is honest and secret, the simulator S_1 will behave as follows. On receiving the inputs in round i, S_1 ignores them and instead makes three random queries with the form of $(\alpha_j \in \mathbb{Z}_q, \beta_j \in \mathbb{Z}_p^*)$ to U_1' and U_2'. Then simulator S_1 tests all outputs $(i.e., {\beta_j}^{\alpha_j})$ from each server and sets the values on the basis of the results returned from the servers. If S_1 detects an error, simulator S_1 saves its states and outputs $Y_p^i = "error"$, $Y_u^i = \emptyset$, $replace^i = 1$. If there is no error detected, S_1 outputs $Y_p^i = \emptyset$, $Y_u^i = \emptyset$, $replace^i = 0$; otherwise, S_1 selects a random element $r \in \mathbb{Z}_p^*$, outputs $Y_p^i = r$, $Y_u^i = \emptyset$ and makes $replace^i = 1$. In either case, S_1 also saves the appropriate states. In the ideal experiment, the inputs are chosen randomly; and in the real one, all queries those T makes to servers are re-randomized, and thus computationally random. So we say that the input distributions to (U_1', U_2') in the ideal and real experiments are computationally blind. If (U_1', U_2') work honestly in round i in the real experiment, $T^{(U_1' U_2')}$ perfectly executes the algorithm and S_1 chooses not to replace the outputs, so we can easily obtain that $EVIEW_{real} \sim EVIEW_{ideal}$. If one of (U_1', U_2') is dishonest in round i, then all failures will be detected by T and S_1, and resulting in an output of "error". In the real experiment, the four outputs generated by (U_1', U_2') are multiplied together along with a random value r. Thus, even one of (U_1', U_2') behaves

Table 1. Comparison of the algorithms with two servers

	[3]	[4]	[23]	[17]	[18]	ours
MM	9	7	17	13	8	8
MInv	5	3	2	3	1	1
Invoke(Rand)	6	5	2	6	4	3
Precomputa-tion	0	0	0	0	10MM+3MInv	0
Invoke(U_1)	4	3	3	3	3	3
Invoke(U_2)	4	3	3	3	2	3
Checkability	$1/2$	$2/3$	0.95	1	1	1
Communica-tion round	1	1	1	2	1	1

dishonestly, we can also get that $EVIEW_{real} \sim EVIEW_{ideal}$. So by the hybrid argument, we conclude that $EVIEW_{real} \sim EVIEW_{ideal}$.

Pair Two: $UVIEW_{real} \sim UVIEW_{ideal}$, which means the untrusted software (U_1', U_2') learns nothing during the execution of (T, U_1', U_2').

The simulator S_2 behaves as follows in the ideal experiment. On receiving the inputs in round i, S_2 ignores them and makes three random queries to both U_1' and U_2'. Then S_2 saves its own states and the states of (U_1', U_2'). These differences between the ideal and real experiments can be easily distinguished by E. However, E cannot communicate with (U_1', U_2') as shown in our security model. Because in the i-th round of real experiment, T always re-randomizes its inputs to (U_1', U_2'); and in the ideal experiment, S_2 always makes random and independent queries to (U_1', U_2'). Thus, we always have $UVIEW_{real} \sim UVIEW_{ideal}$ for each round. So, by the hybrid argument, we also conclude that $UVIEW_{real} \sim UVIEW_{ideal}$.

Theorem 3.2. *In the one-malicious model, the algorithm* $(T, (U_1, U_2))$ *is an* $(O(\frac{log^2 m}{m}), 1)$ *-outsource-secure implementation of our algorithm, where m is the bit length of q.*

Proof. It is well known that it takes about $1.5m$ modular multiplications for computing the modular exponentiations if we use the square-and-multiply method for an m-bit exponent. The proposed algorithm makes 3 calls to *Rand*, 8 modular multiplications (MMs) and 1 modular inverse (Minv) to compute $u^a \ mod \ p$. In addition, it also takes $O(log^2 m)$ MMs by using the EBPV generator or $O(1)$ MMs by using the method of table-lookup for *Rand*. Thus, $(T, (U_1, U_2))$ is an $O(\frac{log^2 m}{m})$-implementation of the proposed algorithm.

In the security model, all the queries those T makes can be verified successfully. So any error will be detected with a probability of 1 if one of (U_1, U_2) is dishonest.

3.3 Efficiency Comparison

In Table 1, we compare the checkability and efficiency of the proposed algorithm with those of the algorithms in [3], [4], [17], [18] and [23], where “MM, MInv, Invoke” denote mod ular multiplication, modular inverse, and an invocation of *Rand* or U, respectively.From Table 1, we conclude that the proposed algorithm only requires 8 MMs, 1 Minv, 3 invocations of *Rand*, 6 invocations of U_1 and U_2. Compared with the algorithms in [3, 4, 23], the proposed one improves the efficiency and checkability simultaneously. Specially, the checkability of [17, 18] and the proposed algorithm are both 1, which means they are all fully verifiable, but the outsourcer needs to interact with the servers for two times in [17] which appends much communication cost, while the other five algorithms only need one communication round. In addition, the algorithm of [18] requires 8 MMs, 1 Minv, 4 invocations of *Rand*, 5 invocations of two servers, but the precomputation is so complicated which needs 10 MMs and 3 MInvs while the other five algorithms do not need any precomputation before outsourcing MExp to the server. Therefore, the proposed algorithm is more efficient than the algorithms in [17] and [18]. In a word, Table 1 shows the proposed algorithm is best from three aspects of checkability, computation and communication efficiency.

4 EFFICIENT OUTSOURCING OF MODULAR EXPONENTIATION WITH SINGLE SERVER

In this section, we first propose an outsourcing algorithm of modular exponentiation with single untrusted server, and then give its security analysis and efficiency comparison.

4.1 Outsourcing Algorithm

In the proposed outsourcing algorithm with single server, we use another subroutine called *Rand'* to generate a random tuple with the form of

$$\{(\alpha, g^{\alpha}, g^{-\alpha}), (\beta, g^{\beta}), (\theta_x, g^{\theta_x}), (t_x^{-1}, g^{t_x}), (\xi_j, g^{-\xi_j}), \mu_j, k, g^{\sum_{i=1}^{k} \xi_i \mu_i}, g^{\sum_{i=k+1}^{b} \xi_i \mu_i}\},$$

where $x = \{1,2,3,4\}$, $j = \{1,2,\ldots,b+3\}$, $k \in \{2,\ldots,b-2\}$, and b is a positive integer. Note that *Rand'* can also be realized by an EBPV generator or a table-lookup method. The outsourcer T executes the following operations:

1. *T* firstly runs *Rand*' once to create a blinding vector:

$$\{(\alpha, g^{\alpha}, g^{-\alpha}), (\beta, g^{\beta}), (\theta_x, g^{\theta_x}), (t_x^{-1}, g^{t_x}), (\xi_j, g^{-\xi_j}), \mu_j, k, g^{\sum_{i=1}^{k}\xi_i\mu_i}, g^{\sum_{i=k+1}^{b}\xi_i\mu_i}\}.$$

We denote $v = g^{\alpha}$, and then get the first logical division:

$$u^a = (vw)^a = g^{\alpha a}w^a = g^{\beta}g^{\gamma}w^a,$$

where $w = u/v$ and $\gamma = \alpha a - \beta$.

2. *T* randomly chooses t_5 and a prime n_1, which is coprime with w. Then *T* gets *s* by computing

$$a \equiv \varphi(n_1)t_5 + s(mod\ q),$$

where $\varphi(n_1)$ is the value of *Euler function* of n_1, and $\varphi(n_1) = n_1 - 1$.
The second logical division is:

$$u^a = g^{\beta}g^{\gamma}w^a = g^{\beta}g^{\gamma}w^{\varphi(n_1)t_5+s}.$$

3. Next, *T* compues

$$r_1 = \varphi(n_1)t_5 - \sum_{i=1}^{k}\mu_i\ ,$$
$$r_2 = \varphi(n_1)t_5 + s + \sum_{i=k+1}^{b}\mu_i,$$
$$w_j = wg^{-\xi_j}, j = \{1,2,\ldots,b+3\}.$$

4. Then *T* randomly chooses a prime n_2, which is coprime with g. Then *T* obtains t_6, t_7, t_8, t_9 by computing

$$\gamma \equiv \varphi(n_2)t_6 + \theta_1(mod\ q),$$
$$s\xi_{b+1} \equiv \varphi(n_2)t_7 + \theta_2(mod\ q),$$
$$r_1\xi_{b+2} \equiv \varphi(n_2)t_8 + \theta_3(mod\ q),$$
$$r_2\xi_{b+3} \equiv \varphi(n_2)t_9 + \theta_4(mod\ q),$$

where $\varphi(n_2)$ is the value of *Euler function* of n_2, and $\varphi(n_2) = n_2 - 1$. Note that $[\varphi(n_2)]^{-1}\ mod\ q$ is needed to compute t_6, t_7, t_8, t_9.

Note that in the above phase, the blinding vector is randomly generated and many parameters are randomly chosen by *T*, and so they are only known for the outsourcer. Therefore, the following parameters

$$n_1, n_2, k, g^{\sum_{i=1}^{k}\xi_i\mu_i}, g^{\sum_{i=k+1}^{b}\xi_i\mu_i}, t_i, \theta_i, g^{\theta_i}, i = \{1,2,3,4\}$$

are all secret for the server.

5. *T* queries U in random order as below:

$$U\,((\varphi(n_2)t_6 + \theta_1)/t_1\,, g^{t_1}) \rightarrow \eta_1 = g^{\varphi(n_2)t_6+\theta_1};$$
$$U\,((\varphi(n_2)t_7 + \theta_2)/t_2\,, g^{t_2}) \rightarrow \eta_2 = g^{\varphi(n_2)t_7+\theta_2};$$
$$U\,((\varphi(n_2)t_8 + \theta_3)/t_3\,, g^{t_3}) \rightarrow \eta_3 = g^{\varphi(n_2)t_8+\theta_3};$$
$$U\,((\varphi(n_2)t_9 + \theta_4)/t_4\,, g^{t_4}) \rightarrow \eta_4 = g^{\varphi(n_2)t_9+\theta_4};$$
$$U(\mu_j, w_j) \rightarrow w_j^{\mu_j}, j = \{1,2,\ldots,b\};$$
$$U(s, w_{b+1}) \rightarrow w_{b+1}^{s};$$
$$U(r_1, w_{b+2}) \rightarrow w_{b+2}^{r_1};$$
$$U(r_2, w_{b+3}) \rightarrow w_{b+3}^{r_2}.$$

Finally, *T* checks whether U produces the correct outputs, *i.e.*, according to *Euler Theorem* we know that

$$\eta_x \equiv g^{\theta_x}(mod\ n_2), x = \{1,2,3,4\}; \quad (1)$$

$$w_{b+2}^{r_1}(\prod_{i=1}^{k} w_i^{\mu_i}) \cdot g^{\sum_{i=1}^{k}\xi_i\mu_i}\eta_3 = w^{\varphi(n_1)t_5} \equiv 1(mod\ n_1); \quad (2)$$

$$w_{b+3}^{r_2}\eta_4 \equiv (\prod_{i=k+1}^{b} w_i^{\mu_i}) \cdot g^{\sum_{i=k+1}^{b}\xi_i\mu_i}w_{b+1}^{s}\eta_2(mod\ n_1). \quad (3)$$

Note that the effectiveness of equations (1-3) are based on the privateness of

$$n_1, n_2, g^{\sum_{i=1}^{k}\xi_i\mu_i}, g^{\sum_{i=k+1}^{b}\xi_i\mu_i}, g^{\theta_i}, i = \{1,2,3,4\}.$$

It is impossible for the server to cheat the outsourcer since the above values are all secret as described above. In fact, $\prod_{w=1}^{k} w_i{}^{\mu_i}$ and $\prod_{w=k+1}^{b} w_i{}^{\mu_i}$ are also private for the server since the above queries are executed by the outsourcer in random order, and so each μ_i is indistinguishable from the server. Moreover, k is randomly chosen by the *Rand* subroutine and it is secret for the server, and thus the server cannot obtain the values of $\prod_{w=1}^{k} w_i{}^{\mu_i}$ and $\prod_{w=k+1}^{b} w_i{}^{\mu_i}$. So, the outsourcing result can be fully verified based on the above analysis.

If not, *T* outputs "error"; otherwise, *T* computes:

$$g^{\beta}\eta_1 w_{b+1}^{s}\eta_2 w^{\varphi(n_1)t_5} \equiv g^{\beta}g^{\gamma}w^{\varphi(n_1)t_5+s} \equiv g^{\beta}g^{\gamma}w^a \equiv u^a(mod\ p), \quad (4)$$

where $w^{\varphi(n_1)t_5}$ can be obtained by the outsourcer *T* from the equation (2).

Remark 3. Now we show the correctness of the above scheme. Firstly,

$$\eta_1 = g^{\varphi(n_2)t_6+\theta_1} \equiv g^{\theta_1}(mod\ n_2),$$
$$\eta_2 = g^{\varphi(n_2)t_7+\theta_2} \equiv g^{\theta_2}(mod\ n_2),$$
$$\eta_3 = g^{\varphi(n_2)t_8+\theta_3} \equiv g^{\theta_3}(mod\ n_2),$$
$$\eta_4 = g^{\varphi(n_2)t_9+\theta_4} \equiv g^{\theta_4}(mod\ n_2),$$

and so the equation (1) holds.

Secondly, because that

$$r_1\xi_{b+2} \equiv \varphi(n_2)t_8 + \theta_3(mod\ q),$$
$$r_1 = \varphi(n_1)t_5 - \sum_{i=1}^{k}\mu_i\ ,$$

and so

$$w_{b+2}^{r_1}(\prod_{i=1}^{k} w_i^{\mu_i}) \cdot g^{\sum_{i=1}^{k}\xi_i\mu_i}\eta_3$$
$$= (wg^{-\xi_{b+2}})^{r_1}[\prod_{i=1}^{k}(wg^{-\xi_i})^{\mu_i}]\, g^{\sum_{i=1}^{k}\xi_i\mu_i}g^{\varphi(n_2)t_8+\theta_3}$$
$$= w^{r_1}g^{-r_1\xi_{b+2}}(\prod_{i=1}^{k} w^{\mu_i}g^{-\mu_i\xi_i}) \cdot g^{\sum_{i=1}^{k}\xi_i\mu_i}g^{\varphi(n_2)t_8+\theta_3}$$
$$= w^{r_1}g^{-r_1\xi_{b+2}}(\prod_{i=1}^{k} w^{\mu_i}) \cdot g^{r_1\xi_{b+2}}$$
$$= w^{r_1}(\prod_{i=1}^{k} w^{\mu_i})$$
$$= w^{\varphi(n_1)t_5} \equiv 1(mod\ n_1),$$

and therefore the equation (2) is proved.

Thirdly, since that

$$r_2\xi_{b+3} \equiv \varphi(n_2)t_9 + \theta_4 (mod\ q),$$

and so

$$w_{b+3}^{r_2}\eta_4 = \left(wg^{-\xi_{b+3}}\right)^{r_2} g^{\varphi(n_2)t_9+\theta_4} = \left(wg^{-\xi_{b+3}}\right)^{r_2} g^{r_2\xi_{b+3}} = w^{r_2}. \quad (5)$$

In addition, because that

$$s\xi_{b+1} \equiv \varphi(n_2)t_7 + \theta_2 (mod\ q),$$

and so

$$\begin{aligned}&(\prod_{i=k+1}^{b} w_i^{\mu_i}) \cdot g^{\sum_{i=k+1}^{b}\xi_i\mu_i} w_{b+1}^{s}\eta_2 \\ &= \left[\prod_{i=k+1}^{b} (wg^{-\xi_i})^{\mu_i}\right] \cdot g^{\sum_{i=k+1}^{b}\xi_i\mu_i} w_{b+1}^{s} g^{\varphi(n_2)t_7+\theta_2} \\ &= [\prod_{i=k+1}^{b} (wg^{-\xi_i})^{\mu_i}] g^{\sum_{i=k+1}^{b}\xi_i\mu_i} (wg^{-\xi_{b+1}})^{s} g^{\varphi(n_2)t_7+\theta_2} \\ &= (\prod_{i=k+1}^{b} w^{\mu_i}) \left(wg^{-\xi_{b+1}}\right)^{s} g^{s\xi_{b+1}} \\ &= (\prod_{i=k+1}^{b} w^{\mu_i}) \cdot w^{s} \\ &= w^{s+\sum_{i=k+1}^{b}\mu_i} \quad (6)\end{aligned}$$

Moreover, $r_2 = \varphi(n_1)t_5 + s + \sum_{i=k+1}^{b}\mu_i$. From (5), we conclude that

$$w_{b+3}^{r_2}\eta_4 = w^{r_2} = w^{\varphi(n_1)t_5+s+\sum_{i=k+1}^{b}\mu_i} \equiv w^{s+\sum_{i=k+1}^{b}\mu_i} (mod\ n_1)$$

Thus, the equation (3) is proved through the above analysis.

Assume the above equations are verified successfully, T computes:

$$\begin{aligned}&g^{\beta}\eta_1 w_{b+1}^{s}\eta_2 w^{\varphi(n_1)t_5} \\ &\equiv g^{\beta} g^{\varphi(n_2)t_6+\theta_1} (wg^{-\xi_{b+1}})^{s} g^{\varphi(n_2)t_7+\theta_2} w^{\varphi(n_1)t_5} \\ &\equiv g^{\beta} g^{\gamma} (wg^{-\xi_{b+1}})^{s} g^{s\xi_{b+1}} w^{\varphi(n_1)t_5} \\ &\equiv g^{\beta} g^{\gamma} w^{\varphi(n_1)t_5+s} \\ &\equiv g^{\beta} g^{\gamma} w^{a} \\ &\equiv u^{a} (mod\ p)\end{aligned}$$

Therefore, T can obtain the final computational result by equation (4).

Remark 4. It is obvious that security and checkability of the algorithm are also relative to the value of *b*. From [20], we know that the bit length of a random number should be larger than 64, which means that it has at least 2^{64} possible values. Equivalently, the parameter *b* should be larger than 33 in the proposed algorithm with single server. The reason is shown as follows. As we know, security of the above proposed algorithm is relative to r_2 and $\sum_{i=k+1}^{b}\mu_i$ since

$$a \equiv \varphi(n_1)t_5 + s \equiv r_2 - \sum_{i=k+1}^{b}\mu_i\ (mod\ q).$$

For the server, the number of possible values for r_2 is $b + 7$ since the outsourcer queries the server for $b + 7$ times in random order. After determining r_2, the server needs to find all of μ_i and compute $\sum_{i=k+1}^{b}\mu_i$. Therefore, the number of possible values for $\sum_{i=k+1}^{b}\mu_i$ is $C_{b+6}^{b} \cdot \sum_{k=2}^{b-2} C_b^k$, where C_{b+6}^{b} means the combination of picking b queries from $b + 6$ queries and C_b^k means the combination of picking k integers from b integers. Further, the number of all the possible values for a is about $(b + 7) \cdot C_{b+6}^{b} \cdot \sum_{k=2}^{b-2} C_b^k$, which approximately equals 2^{64} if $b = 33$. So, the number of possible values for a random number is equivalent to that of [20] when $b = 33$.

4.2 Security Analysis

Security analysis of the algorithm with single server is similar to that of the algorithm with two servers, so we give a brief proof here, and the correctness of the algorithm is shown as Section 4.1.

Theorem 4.1. *In the outsourcing algorithm with single server, if the input* (a, u) *is honest, secret; or honest, protected; or adversarial, protected,* (T, U) *is an outsource-secure implementation of the proposed algorithm.*

Proof. The proof is similar to [20]. Let $A = (E, U')$ be an adversary that interacts with algorithm in the security model with single server.

Pair One: $EVIEW_{real} \sim EVIEW_{ideal}$

As shown in Theorem 3.1, the simulator S_1 behaves same as in the real execution if the input (a, u) is honest, protected, or adversarial, protected. Thus, it only needs to prove the case where (a, u) is an honest, secret input. The simulator S_1 will behave as follows. On receiving the inputs in round *i*, S_1 ignores them and instead makes $b + 7$ random queries to U'. Then simulator S_1 tests all outputs from the server. If S_1 detects an error, it saves its states and outputs $Y_p^i = "error"$, $Y_u^i = \emptyset$, $replace^i = 1$. If there is no error detected, S_1 outputs $Y_p^i = \emptyset$, $Y_u^i = \emptyset$, $replace^i = 0$; otherwise, S_1 selects a random element r and outputs $Y_p^i = r$, $Y_u^i = \emptyset$ and makes $replace^i = 1$. In either case, S_1 also saves its states. If U' works honestly in round i of the real experiment, then $T^{U'}$ perfectly executes the algorithm and S_1 chooses not to replace the outputs with a random element, so we obtain that $EVIEW_{real} \sim EVIEW_{ideal}$. If U' is dishonest in round *i*, then the failures will be detected by *T* and S_1 with a probability of 1, and resulting in an output of "error". In the real experiment, the $k + 5$ outputs generated by U' are multiplied together along with a random value *r*. Thus, we can also get that $EVIEW_{real} \sim EVIEW_{ideal}$ even U' behaves dishonestly.

Pair Two: $UVIEW_{real} \sim UVIEW_{ideal}$

The simulator S_2 behaves as follows. On receiving the input in round *i*, S_2 ignores them and instead makes $b + 7$ random queries to U'. Then S_2 saves its own states and the states of U'. These differences between ideal and real experiments can be easily distinguished by *E*, but *E* cannot transfer this information to *U*. Thus, in each round, we always have $UVIEW_{real} \sim UVIEW_{ideal}$.

Theorem 4.2. *In the outsourcing algorithm with single server,* (T, U) *is an* $(O(\frac{log^2 m}{m}),1)$*-outsource-secure implementation of our algorithm.*

Proof. The proposed algorithm with single server makes 1 call to *Rand'*, $2b + 16$ modular multiplications (MMs) and 1 modular inverse (Minv) to compute $u^a\ mod\ p$. Our algorithm also takes $O(log^2 m)$ MMs by using the EBPV generator or $O(1)$ MMs by using the method of table-lookup for *Rand'*. Thus, (T, U) is an $O(\frac{log^2 m}{m})$-efficient implementation of the proposed algorithm with single server. On the other hand, all the queries those T makes can be verified to detect the failures. Therefore, the fault returned by the server will be detected with a probability of 1 if U is dishonest.

4.3 Comparison

We also compare the checkability and efficiency of the proposed algorithm based on single untrusted server with those of [20], [21], [22] and [24].

In Table 2, we present the comparison of checkability and efficiency for the outsourcer in the algorithms with single server. As shown in [20], [21] and presented in Remark 4, at the same level of security, we set $c = r = 4, k = l = 29, b = 33$, and χ, t_1, t_2 are all random numbers which are larger than 2^{64}.

Table 2. Comparison of the algorithms with single server

	[20]	[24]	[21]	[22]	ours
MM	$12 + 1.5 log \chi \geq 108$	$1.5\, log\, r + 1.5\, log\, t_1 + 1.5\, log\, t_2 + 15 \geq 210$	$l + k + 8\, log\, c + 38 = 112$	22	$2b + 16 = 82$
MInv	4	6	1	9	1
Rand'	6	6	5	12	1
Check ability	0.5	0.991	≈ 0.917	≈ 1	1

From Table 2, it is obvious that the proposed algorithm is superior to [20], [21] and [24] in both efficiency and checkability. Compared with the algorithms in [22], the proposed algorithm appends some modular multiplications (MMs) but decreases the times of modular inversions (MInvs) greatly. As we know, the computation complexity of MInv is much higher than that of MM, and so the proposed algorithm is more efficient than the algorithm in [22]. Thus, the proposed algorithm with single server improves the checkability and efficiency simultaneously compare with the previous ones.

5 Experiment Evaluation

In this section, we provide the experiment evaluations to show the efficiency of the proposed two algorithms for outsourcing modular exponentiations.

The parameters p and q are similar to [4], that is, p is a 512-bit prime and q is a 160-bit prime. The programming language is C++.

In the first proposed algorithm based on two untrusted servers, the cloud servers and the outsourcer are simulated by Intel Core i7 processors at 3.4GHz and 4G RAM, and Intel Pentium CPU at 1.0GHz and 2G memory, respectively.

In Fig.1, we give the comparison of computation cost for an outsourcer and two servers in the first proposed outsourcing algorithm. It is obvious that the computation time for T is much smaller than that of computing modular exponentiation directly since a lot of computation cost has been delegated to two servers. Moreover, the computation time of the servers is also smaller than that of directly computing modular exponentiation.

We also compare the computational time for the outsourcer T in the first proposed algorithm with that of the algorithms in [3], [4], [17-18] and [23] in Fig.2. As shown in Fig.2, the first proposed algorithm is the most efficient one compare with the algorithms in [3], [4], [17-18] and [23] for the outsourcer. Moreover, it improves the checkability to 1 which means the outsourcer can verify the computational result returned by the server with a probability of 1. Therefore, the proposed algorithm improves the efficiency and checkability of the outsourcer simultaneously compare with the previous ones.

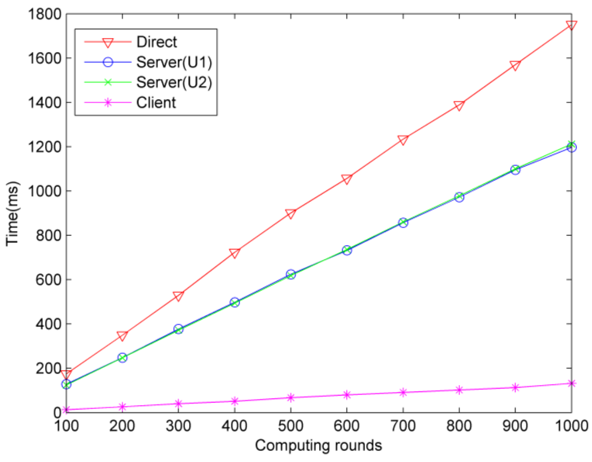

Fig.1. Simulation for the first proposed algorithm based on two servers

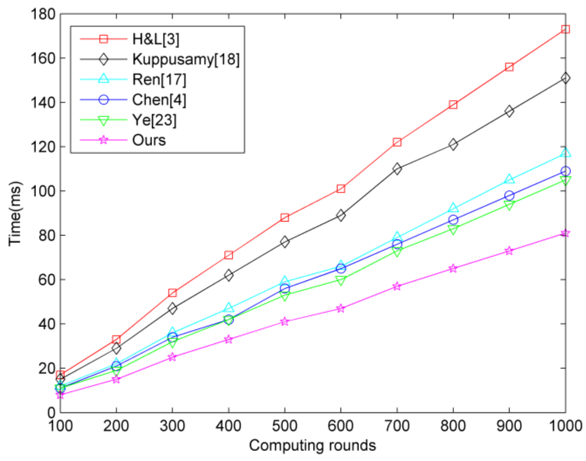

Fig.2. Time comparison among the outsourcing algorithms based on two servers

In the second proposed algorithm with single server, the cloud server and the outsourcer are separately simulated by a machine with 32 Intel Xeon CPUs running at 2.6GHz and 32G RAM, and Intel Pentium CPU running at 2.0GHz and 2G memory.

In Fig.3, we give the comparison of computation cost for an outsourcer and a server in the second proposed algorithm with single server. It is shown that the computation cost for the outsourcer T is much smaller than that of directly computing modular exponentiation, and the computation cost of the server is also smaller than that of direct computation.

In Fig.4, we compare the computation time for the outsourcer in the second proposed algorithm with that of the algorithms in [20], [21], [22] and [24]. As shown in Section 4.3, at the same level of security, we set $c = r = 4, k = l = 29, b = 33$, and χ, t_1, t_2 are all random numbers which are larger than 2^{64}. From Fig.4, we conclude that for the outsourcer, the second proposed algorithm is the most efficient one in all of the outsourcing algorithms with single server when computing modular exponentiation.

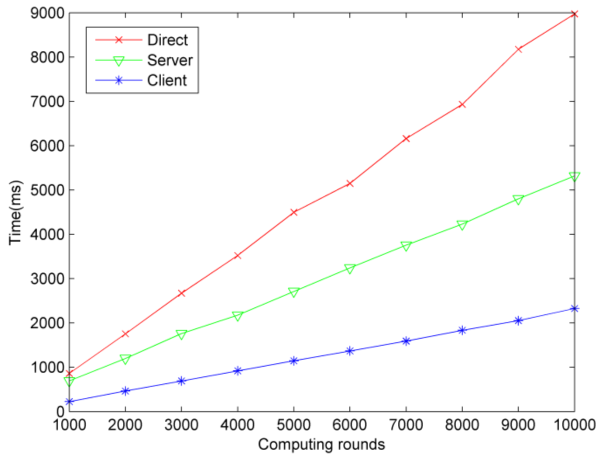

Fig.3. Simulation for the second proposed algorithm with single server

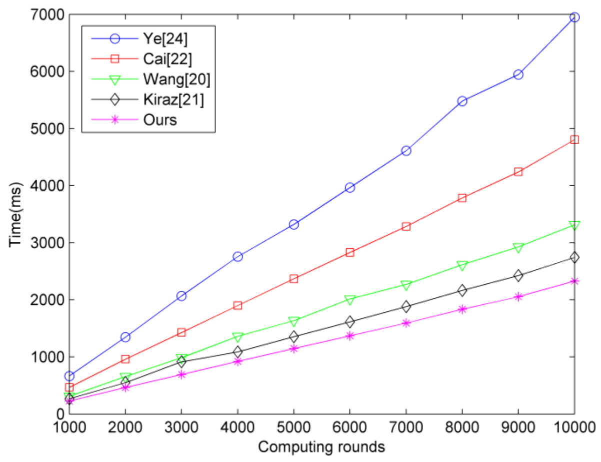

Fig.4. Time comparison among the outsourcing algorithms with single server

Remark 5. In the experiments, we set different computing counts in the proposed two algorithms. The reason is that we need more counts to make all CPUs have enough time to start their computations since a machine with 32 CPUs are used to simulate the cloud server in the algorithm with single server.

6 Conclusion

In this paper, we propose two non-interactive verifiable outsourcing algorithms for modular exponentiation. Firstly, we propose a new outsourcing algorithm of modular exponentiation based on two untrusted servers, where the outsourcer can detect the error based on *Euler theorem* with a probability of 1 if one of the servers misbehaves. We then propose another outsourcing algorithm with single server, which improves the efficiency and the checkability simultaneously compare with the previous ones. Finally, we conclude that the proposed algorithms are most efficient in all of the outsourcing algorithms for the outsourcer by theoretical analysis and experimental evaluation.

.Acknowledgment

The work described in this paper was supported by the National Natural Science Foundation of China (Grant No. 61572309, U1736120, U1636206, 61525203, U1536108, U1536109), and the Shanghai Dawn Scholar Plan (14SG36) and the Shanghai Excellent Academic Leader Plan (16XD1401200).

References

[1] R. Gennaro, C. Gentry, and B. Parno, *Non-interactive verifiable computing: outsourcing computation to untrusted workers*, CRYPTO 2010, LNCS 6223, Springer, pp. 465-482, 2010.
[2] D. Chaum, T. Pedersen, *Wallet databases with observers*, CRYPTO 1992, LNCS 740, Springer, pp. 89-105, 1992.
[3] S. Hohenberger, A. Lysyanskaya, *How to securely outsource cryptographic computations*, TCC 2005, LNCS 3378, Springer, pp. 264-282, 2005.
[4] X. Chen, J. Li, and J. Ma, *New algorithms for secure outsourcing of modular exponentiations*, IEEE Transactions on Parallel and Distributed Systems, vol. 25, no. 9, pp. 2386-2396, 2014.
[5] X. Chen, W. Susilo, J. Li, D. Wong, et al., *Efficient algorithms for secure outsourcing of bilinear pairings*, Theoretical Computer Science, vol. 562, pp. 112-121, 2015.
[6] J. Lai, R. Deng, C. Guan, and J. Weng, *Attribute-based encryption with verifiable outsourced decryption*, IEEE Transactions on Information Forensics and Security, vol. 8, no. 8, pp. 1343-1354, 2013.
[7] X. Chen, J. Li, X. Huang, J. Li, Y. Xiang, and D. Wong, *Secure outsourced attribute-based signatures*. IEEE Transactions on Parallel and Distributed Systems, vol. 25, no. 12, pp. 3285-3294, 2014.
[8] Y. Ren, N. Ding, X. Zhang, H. Lu, and D. Gu, *Identity-based encryption with verifiable outsourcing revocation*, The Computer Journal, vol. 59, no. 11, pp. 1659-1668, 2016.
[9] Y. Ren, X. Zhang, G. Feng, Z. Qian, F. Li, *How to extract*

image features based on co-occurrence matrix securely and efficiently in cloud computing, IEEE Transactions on Cloud Computing, doi: 10.1109/TCC.2017. 2737980

[10] W. Wu, Y. Mu, W. Susilo, and X. Huang, *Server-aided verification signatures: definitions and new constructions*, ProvSec 2008, LNCS 5324, Springer, pp. 141-155, 2008.

[11] T. Matsumoto, K. Kato, and H. Imai, *Speeding up secret computations with insecure auxiliary devices*, CRYPTO 1988, LNCS 403, Springer, pp. 497-506, 1988.

[12] M. Girault, D. Lefranc, *Server-aided verification: theory and practice*, ASIACRYPT 2005, LNCS 3788, Springer, pp. 605-623, 2005.

[13] V. Boyko, M. Peinado, and R. Venkatesan, *Speeding up discrete log and factoring based schemes via precomputations*, EUROCRYPT 1998, LNCS 1403, Springer, pp. 221-232, 1998.

[14] S. Even, O. Goldreich, and S. Micali, *On-line/off-line digital signatures*, Journal of Cryptology, vol. 9, no. 1, pp. 35-67, 1996.

[15] P. Nguyen, I. Shparlinski, and J. Stern, *Distribution of modular sums and the security of server aided exponentiation*, CCNT 1999, pp. 1-16, 1999.

[16] X. Ma, J. Li, F. Zhang, *Efficient and secure batch exponentiations outsourcing in cloud computing*, Proceedings of 4th International Conference on Intelligent Networking and Collaborative Systems, IEEE, pp. 600-605, 2012.

[17] Y. Ren, N. Ding, X. Zhang, H. Lu and D. Gu, *Verifiable outsourcing algorithms for modular exponentiations with improved checkability*, ASIACCS 2016, ACM, pp. 293-303, 2016.

[18] L. Kuppusamy, J. Rangasamy, *CRT-based outsourcing algorithms for modular exponentiations*, INDOCRYPT 2016 , LNCS 10095, Springer, pp. 81-98, 2016.

[19] M. Dijk, D. Clarke, B. Gassend, G. Suh, and S. Devadas, *Speeding up exponentiation using an untrusted computational resource*, Designs, Codes and Cryptography, vol. 39, no. 2, pp.253-273, 2006.

[20] Y. Wang, Q. Wu, D. S. Wong, *Securely outsourcing exponentiations with single untrusted program for cloud storage*, ESORICS 2014, LNCS 8712, Springer, pp. 326–343, 2014.

[21] M. Kiraz, O. Uzunkol, E*fficient and verifiable algorithms for secure outsourcing of cryptographic computations*, International Journal of Information Security, vol. 15, no. 5, pp. 519-537, 2016.

[22] J. Cai, Y. Ren, C. Huang, *Verifiable outsourcing computation of modular exponentiations with single server*, International Journal of Network Security, vol. 19, no. 3, pp. 449-457, 2017.

[23] J. Ye, X. Chen, and J. Ma, *An improved algorithm for secure outsourcing of modular exponentiations*, Proceedings of 29th International Conference on Advanced Information Networking and Applications, IEEE, pp. 73-76, 2015.

[24] J. Ye, J. Wang, *Secure outsourcing of modular exponentiation with single untrusted server*, Proceedings of 18th International Conference on Network-Based Information Systems, pp. 643-645, 2015.

[25] D. Stinson, Cryptography theory and practice (Third Edition), 2016.

[26] M. Atallah, K. Frikken, *Securely outsourcing linear algebra computations*, ASIACCS 2010, ACM, pp. 48-59, 2010.

Yanli Ren is an associate professor in School of Communication and Information Engineering at Shanghai University, China. She was awarded a M.S. degree in applied mathematics in 2005 from Shaanxi Normal University, China, and a PhD degree in computer science and technology in 2009 from Shanghai Jiao Tong University, China. She has published more than 60 papers on international journals and conferences.Her research interests include applied cryptography, secure outsourcing computing, and network security.

Min Dong received the BS degree on communication engineering from Shanghai University, China, in 2015. Currently, he is a graduate student at the School of Communication and Information Engineering in Shanghai University, Shanghai, China. His research interests include applied cryptography and secure outsourcing computing.

Zhenxing Qian received both the B.S. and the Ph.D. degrees from University of Science and Technology of China (USTC) in 2003 and 2007, respectively. Since 2009, he has been with the faculty of the School of Communication and Information Engineering, Shanghai University, where he is currently a Professor. He has published over 80 peer-reviewed papers on international journals and conferences. His research interests include information hiding, image processing and multimedia security.

Xinpeng Zhang received the B.S. degree in computational mathematics from Jilin University, China, in 1995, and the M.S. and Ph.D. degrees in communication and information system from Shanghai University, China, in 2001 and 2004, respectively. Since 2004, he has been with the faculty of the School of Communication and Information Engineering, Shanghai University, where he is currently a Professor. He was with the State University of New York at Binghamton as a visiting scholar from January 2010 to January 2011, and Konstanz

University as an experienced researcher sponsored by the Alexander von Humboldt Foundation from March 2011 to May 2012. He is an Associate Editor for IEEE Transactions on Information Forensics and Security. His research interests include multimedia security, image processing, and digital forensics. He has published more than 200 papers in these areas.

Guorui Feng received the B.S. and M.S. degree in computational mathematic from Jilin University, China, in 1998 and 2001 respectively. He received Ph.D. degree in electronic engineering from Shanghai Jiaotong University, China, 2005. From January 2006 to December 2006, he was an assistant professor in East China Normal University, China. During 2007, he was a research fellow in Nanyang Technological University, Singapore. Now he is with the school of communication and information engineering, Shanghai University, China. His current research interests include image processing, image analysis and computational intelligence.