Verifiable Outsourcing Algorithms for Modular Exponentiations with Improved Checkability

Yanli Ren*
School of Communication and Information Engineering
Shanghai University
Shanghai 200444, China
renyanli@shu.edu.cn

Ning Ding†
School of Electronic Information and Electrical Engineering
Shanghai Jiao Tong University
Shanghai 200240, China
dingning@sjtu.edu.cn

Xinpeng Zhang
School of Communication and Information Engineering
Shanghai University
Shanghai 200444, China
xzhang@shu.edu.cn

Haining Lu
School of Information Security and Engineering
Shanghai Jiao Tong University
Shanghai 200240, China
hnlu@sjtu.edu.cn

Dawu Gu
School of Electronic Information and Electrical Engineering
Shanghai Jiao Tong University
Shanghai 200240, China
dwgu@sjtu.edu.cn

ABSTRACT

The problem of securely outsourcing computation has received widespread attention due to the development of cloud computing and mobile devices. In this paper, we first propose a secure verifiable outsourcing algorithm of single modular exponentiation based on the one-malicious model of two untrusted servers. The outsourcer could detect any failure with probability 1 if one of the servers misbehaves. We also present the other verifiable outsourcing algorithm for multiple modular exponentiations based on the same model. Compared with the state-of-the-art algorithms, the proposed algorithms improve both checkability and efficiency for the outsourcer. Finally, we utilize the proposed algorithms as two subroutines to achieve outsource-secure polynomial evaluation and ciphertext-policy attributed-based encryption (CP-ABE) scheme with verifiable outsourced encryption and decryption.

Keywords

Cloud computing; Verifiable computation; Outsourcing algorithm; Modular exponentiation

*Corresponding author

†This author is also with the State Key Laboratory of Cryptology, P. O. Box 5159, Beijing 100878, China

ASIA CCS '16, May 30-June 03, 2016, Xi'an, China

DOI: http://dx.doi.org/10.1145/2897845.2897881

1. INTRODUCTION

Verifiable computation (VC) allows a computationally weak client to outsource evaluation of a function on many inputs to a powerful but untrusted server [8, 13]. The client in this model invests a large amount of off-line computation and generates an encoding of its function. Given this encoding and any input, the server performs the computation and responds with a result and a proof that the result is correct. With the server's response, the client can verify if the computation has been carried out correctly using substantially less effort than computing the function directly.

The problem of securely outsourcing computation has received widespread attention due to the rise of cloud computing and the proliferation of mobile devices, such as smart phones and netbooks [2]. For example, a computationally weak device might off-load heavy computations, e.g., a cryptographic operation or a photo manipulation, to a network server. Moreover, a proof of the correctness of the result might be desirable if not necessary.

In the cryptographic community, there is a long history of outsourcing expensive cryptographic operations to a semi-trusted device. Chaum et al. [4] first introduced the concept of "wallets with observers" that allows a piece of hardware installed on the client's device to carry out some computations for each transaction. Hohenberger et al. formalized this model [17], and presented protocols for the computation of modular exponentiations based on two non-colluding servers. Chen et al. [7] proposed an efficient outsourcing algorithm of bilinear pairing in the one-malicious version of two untrusted servers, where the outsourcer only carried out 5 point additions and 4 multiplications without any expensive operations. Lai et al. [20] considered the verifiability of attributed-based encryption (ABE) with outsourced decryption, which guarantees that an outsourcer can efficiently check if the server returns the correct result. Li et al. [21] introduced outsourcing computation into identity-based encryption (IBE) revocation, formalized the security definition

and proposed a scheme to offload all the key-update related operations to a cloud server. Other work targets specific classes of functions, such as one-way function inversion [14], large-scale linear equations [5] and so on.

Hohenberger et al. [17] presented the first security model for outsourcing cryptographic computations, and proposed the first outsource-secure algorithm for modular exponentiations. In their model, the adversarial environment writes the software for the malicious servers, but then does not have direct communication with them once the device starts relying on them. In addition to security, they also provide a framework for quantifying the efficiency and checkability of an outsourcing implementation. In their algorithms, the outsourcer has the ability to detect any failure if the cloud servers misbehave. Chen et al. [6] proposed a new secure outsourcing algorithm for modular exponentiations in the one-malicious version of two untrusted program model. Compared with the algorithm described in [17], the proposed algorithm is better in both efficiency and checkability. They also presented an efficient algorithm for outsourcing simultaneous modular exponentiations. In the proposed algorithms of [6, 17], the checkability is 1/2 and 2/3 respectively, and the outsourcer cannot detect the failure with probability 1 if it is cheated by the server. Therefore, it is possible for the outsourcer to be cheated by the server and the outsourcer cannot check the error successfully in the algorithms of [17] and [6]. Ma et al. presented several outsourcing algorithms for batch modular exponentiations [22], but the bases or exponents of the computations are public for the servers and the algorithms cannot realize the input privacy completely. All the algorithms in [6, 17, 22] are based on two untrusted servers. Dijk et al. [9] presented an outsourcing algorithm for exponentiation based on a single server, but it cannot ensure the privacy of the input since the base of the exponentiation is known for the server. Wang et al. [26] also constructed an efficient algorithm for batch modular exponentiation and realized provable data possession (PDP) [25, 27] based on an untrusted server, but the outsourcer needs to execute one modular exponentiation itself when verifying the outsourced result and the checkability is only for modular exponentiation. Therefore, the algorithm of Wang et al. has no advantage in the efficiency and the checkability for single modular exponentiation.

Our Contributions. In this paper, we first propose a secure verifiable outsourcing algorithm of single modular exponentiation in the one-malicious model of two untrusted servers. The outsourcer could detect any failure with probability 1 if one of the servers returns the fault result. We also present another outsourcing algorithm for multiple modular exponentiations which improves checkability and efficiency for the outsourcer simultaneously compare with the previous one. Finally, we utilize two proposed algorithms as subroutines to achieve private outsourcing of polynomial evaluation and CP-ABE scheme with verifiable outsourced encryption and decryption.

2. SECURITY DEFINITION AND MODEL

In this section, we review the formal security definition and model of an outsourcing algorithm introduced by [17].

2.1 Definition of Outsource-security

An algorithm **Alg** includes a trusted part T and an untrusted program U, and T^U denotes the works that carried out by T invoking U. An adversary A is simulated by a pair of algorithms (E, U'), where E denotes the adversarial environment that submits adversarial inputs for **Alg**, and U' represents an adversarial software written by E. As described in [17], we assume that the two adversaries (E, U') can only make direct communication before the execution of T^U, and in other cases, they can only communicate with each other by passing messages through the outsourcer T. In the real world, a malicious manufacturer E might program its software U' to behave in an adversarial fashion; but once U' is installed behind T's firewall, E should no longer be able to directly send instructions to it [17]. We first introduce the inputs and outputs for an outsourcing algorithm in the following definition.

Definition 1. (Algorithm with outsource-I/O) An algorithm **Alg** takes five inputs, and produces three outputs. The first three inputs are generated by an honest party, and are classified by how much the adversary A knows about them. The first input is called the honest, secret input, which is private for both E and U'; the second is called the honest, protected input, which may be known by E, but is protected from U'; and the third is called the honest, unprotected input, which may be known by both E and U'. The last two inputs are generated by the environment E, including the adversarial, protected input, which is known to E, but protected from U'; and the adversarial, unprotected input, which is public for both E and U'. Similarly, the first output is called secret and private for both E and U'; the second is protected, which may be known to E, but not U'; and the third is unprotected, which may be known by both E and U'.

The following definition of outsource-security ensures that the malicious environment E cannot obtain any information about the secret inputs and outputs of T^U, even if T uses the malicious software U' written by E.

Definition 2. (Outsource-security) Let **Alg** be an algorithm with outsource I/O. A pair of algorithms (T, U) is called an outsource-secure implementation of **Alg** if the following conditions hold.

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

2) Security: For all probabilistic polynomial-time (PPT) adversaries $A = (E, U')$, there exist two PPT simulators (S) such that the following pairs of random variables are computationally indistinguishable.

Pair One: $EVIEW_{real} \sim EVIEW_{ideal}$, which means that the malicious environment E cannot gain anything interesting about the private inputs and outputs during the execution of T^U. Both of the real process and the ideal process proceed in rounds.

The *real* process:

$EVIEW^i_{real} = \{(istate^i, x^i_{hs}, x^i_{hp}, x^i_{hu}) \leftarrow I(1^k, istate^{i-1})\}$;
$(estate^i, j^i, x^i_{ap}, x^i_{au}, stop^i) \leftarrow E(1^k, EVIEW^{i-1}_{real}, x^i_{hp}, x^i_{hu})$;
$(tstate^i, ustate^i, y^i_s, y^i_p, y^i_u) \leftarrow$
$T^{U'(ustate^{i-1})}(tstate^{i-1}, x^{j^i}_{hs}, x^{j^i}_{hp}, x^{j^i}_{hu}, x^i_{ap}, x^i_{au}) :$
$(estate^i, y^i_p, y^i_u)\}$
$EVIEW_{real} = EVIEW^i_{real}$ if $stop^i = TRUE$.

In round i, the honest, secret; honest, protected; and honest, unprotected inputs $(x^i_{hs}, x^i_{hp}, x^i_{hu})$ are selected using an honest, stateful process I to which the environment E does not have access. Then E chooses $estate^i, j^i, x^i_{ap}, x^i_{au}, stop^i$ based on its view in the last round $EVIEW^{i-1}_{real}$ and honest inputs $(x^i_{hs}, x^i_{hp}, x^i_{hu})$ given to $T^{U'}$, where $estate^i$ is a

variable as remembering what it did next time it is invoked, x^i_{ap}, x^i_{au} are two adversarial inputs, and $stop^i$ is a Boolean variable determining whether round i is the last round. Next, the algorithm $T^{U'}$ is executed on the inputs $(tstate^{i-1}, x^{j^i}_{hs}, x^{j^i}_{hp}, x^{j^i}_{hu}, x^i_{ap}, x^i_{au})$, and produces a new state $tstate^i$ for T, the secret, protected, and unprotected outputs y^i_s, y^i_p, y^i_u, where $tstate^{i-1}$ is T's previously saved state. The oracle U' saves current state $ustate^i$ based on its previously saved state $ustate^{i-1}$. The view of the real process in round i includes $estate^i$, y^i_p and y^i_u. The overall view of U in the real process is its view in the last round, where $stop^i = TRUE$.

The *ideal* process:
$EVIEW^i_{ideal} = \{(istate^i, x^i_{hs}, x^i_{hp}, x^i_{hu}) \leftarrow I(1^k, istate^{i-1})\}$;
$(estate^i, j^i, x^i_{ap}, x^i_{au}, stop^i) \leftarrow E(1^k, EVIEW^{i-1}_{ideal}, x^i_{hp}, x^i_{hu})$;
$(astate^i, y^i_s, y^i_p, y^i_u) \leftarrow$
$\mathbf{Alg}(astate^{i-1}, x^{j^i}_{hs}, x^{j^i}_{hp}, x^{j^i}_{hu}, x^i_{ap}, x^i_{au})$;
$(sstate^i, ustate^i, Y^i_p, Y^i_u, rep^i) \leftarrow S_1^{U'(ustate^{i-1})}$
$(sstate^{i-1}, \ldots, x^{j^i}_{hp}, x^{j^i}_{hu}, x^i_{ap}, x^i_{au}, y^i_p, y^i_u)$;
$(z^i_p, z^i_u) = rep^i(Y^i_p, Y^i_u) + (1 - rep^i)(y^i_p, y^i_u) : (estate^i, z^i_p, z^i_u)$
$EVIEW_{ideal} = EVIEW^i_{ideal}$ if $stop^i = TRUE$.

In the ideal process, we have a stateful simulator S_1 who is protected from the secret input x^i_{hs}, but given the protected or unprotected outputs that **Alg** produces for round i, decides to either output (y^i_p, y^i_u) generated by **Alg**, or replace them with some other values (Y^i_p, Y^i_u). Note that this is controlled by a Boolean variable rep^i. During the whole process, S_1 is allowed to query oracle U', and U' saves its state as in the real experiment.

Pair Two: $UVIEW_{real} \sim UVIEW_{ideal}$, which means that the untrusted software U' written by E learns nothing about the inputs and outputs during the execution of T^U.

As defined in Pair One, the view that the untrusted software U' in the real process is $UVIEW_{real} = ustate^i$ if $stop^i = TRUE$.

The ideal process:
$UVIEW^i_{ideal} = \{(istate^i, x^i_{hs}, x^i_{hp}, x^i_{hu}) \leftarrow I(1^k, istate^{i-1})\}$;
$(estate^i, j^i, x^i_{ap}, x^i_{au}, stop^i) \leftarrow$
$E(1^k, estate^{i-1}, x^i_{hp}, x^i_{hu}, y^{i-1}_p, y^{i-1}_u)$;
$(astate^i, y^i_s, y^i_p, y^i_u) \leftarrow \mathbf{Alg}(astate^{i-1}, x^{j^i}_{hs}, x^{j^i}_{hp}, x^{j^i}_{hu}, x^i_{ap}, x^i_{au})$;
$(sstate^i, ustate^i) \leftarrow$
$S_2^{U'(ustate^{i-1})}(sstate^{i-1}, x^{j^i}_{hu}, x^i_{au}) : (ustate^i)$;
$UVIEW_{ideal} = UVIEW^i_{ideal}$ if $stop^i = TRUE$.

In the ideal process, we have a stateful simulator S_2 who is only given the unprotected inputs (x^i_{hu}, x^i_{au}). In the whole process, S_2 is allowed to access oracle U' and U' saves its state as in Pair One.

Assume T^U is a correct implementation of **Alg**, we have the following definitions.

Definition 3. (α-efficient, secure outsourcing) A pair of algorithms (T, U) is α-efficient if the running time of T is no more than an α-multiplicative factor of that of **Alg** for any input x.

Definition 4. (β-checkable, secure outsourcing) A pair of algorithms (T, U) is β-checkable if T detects any deviation of U' from its advertised functionality during the execution of $T^{U'(x)}$ with probability no less than β for any input x.

Definition 5. ((α, β)-outsource-security) A pair of algorithms (T, U) are called an (α, β)-outsource-secure execution of **Alg** if they are α-efficient and β-checkable.

2.2 Security Model

Hohenberger et al. [17] first presented two untrusted program models for outsourcing exponentiations modulo a prime. In this model, the adversarial environment E writes two software $U' = (U'_1, U'_2)$, and T installs these software in a manner such that all subsequent communication between any two of E, U'_1 and U'_2 must pass through T. The new adversary attacking T is $A = (E, U'_1, U'_2)$. We assume that at most one of the software misbehaves, but we don't know which one. It is named as the one-malicious version of two untrusted models. In the real-world applications, it is equivalent to buy the two copies of the advertised software from two different vendors and achieve the security as long as one of them is honest [6].

3. VERIFIABLE SECURE OUTSOURCING OF SINGLE MODULAR EXPONENTIATION

In [17], a subroutine named *Rand* is used to speed up the computations. The inputs for *Rand* are a prime p, a base $g \in Z^*_p$, and the output for each invocation is a random, independent pair of the form $(k, g^k \mod p)$, where $k \in Z_q$. In our paper, we define another subroutine named *Rand'*, and the inputs for *Rand'* are same as *Rand*, and the output is a random, independent tuple of the form $(l, l^{-1}, g^{-l} \mod p)$, where $l \in Z_q$. There are two approaches to implement these functionalities. First, a trusted server computes two tables of random, independent tuples in advance and then stores them into the memory of T, which is called the table-lookup method. For each invocation, T needs to retrieves a new tuple in the table. Second, we can apply the EBPV generator, which is secure against adaptive adversaries and runs in time $O(log^2 L)$ for an L-bit exponent [23].

3.1 Verifiable Outsourcing Algorithm

We propose a new secure outsourcing algorithm **VExp** for exponentiation modulo a prime in the one-malicious model. In **VExp** algorithm, T outsources its modular exponentiation computations to U_1 and U_2 by invoking the subroutine *Rand* and *Rand'*. A requirement for **VExp** is that the adversary cannot know any useful information about the inputs and outputs of **VExp**. In the following algorithm, $U_i(x, y) \rightarrow y^x$ denotes that U_i takes (x, y) as input and outputs $y^x \mod p$, where $i = 1, 2$.

Let p, q be two large primes and $q|p-1$. The input of **VExp** is $a \in Z_q$, and $u \in Z^*_p$ such that $u^q \equiv 1 \pmod p$ for an arbitrary base u and an arbitrary power a. The output of **VExp** is $u^a \mod p$. Both of a and p are computationally blinded to U_1 and U_2. The proposed **VExp** algorithm is described as follows:

1) T firstly runs *Rand* to create four blinding pairs (α, g^α), (β, g^β), (t_1, g^{t_1}), (t_2, g^{t_2}), and then runs *Rand'* to obtain two blinding tuples (b, b^{-1}, g^{-b}), (c, c^{-1}, g^{-c}). We denote:

$$v = g^\alpha \mod p,\ \mu = g^\beta \mod p.$$

2) T splits $u^a = (vw)^a = g^{\alpha a} w^a = g^\beta g^\gamma w^a \mod p$, where $w = u/v \mod p$, $\gamma = (\alpha a - \beta) \mod q$.

3) Next, T queries U_1 and U_2 respectively as below:

$$U_1(b/t_1, wg^{t_1}) \rightarrow D_{11} = w^{b/t_1} g^b,$$
$$U_2(c/t_1, wg^{t_1}) \rightarrow D_{21} = w^{c/t_1} g^c.$$

4) After receiving the outsourcing result from U_1 and U_2, T computes:

$$w^{b/t_1} = D_{11} \cdot g^{-b},\ w^{c/t_1} = D_{21} \cdot g^{-c}.$$

and queries U_1 in random order as:

$$U_1(\gamma/t_2, g^{t_2}) \rightarrow D_{12} = g^{\gamma},$$
$$U_1(at_1/c, w^{c/t_1}) \rightarrow D_{13} = w^a.$$

Similarly, T queries U_2 in random order as:

$$U_2(\gamma/t_2, g^{t_2}) \rightarrow D_{22} = g^{\gamma},$$
$$U_2(at_1/b, w^{b/t_1}) \rightarrow D_{23} = w^a.$$

5) Finally, T verifies that both U_1 and U_2 generate the correct outputs, that is to say,

$$D_{12} = D_{22},\ D_{13} = D_{23}.$$

If not, T outputs "error"; otherwise, T can compute

$$u^a = \mu g^{\gamma} w^a \mod p.$$

Remark 1. In the one-malicious model, the equations

$$D_{12} = D_{22},\ D_{13} = D_{23}$$

implies both U_1 and U_2 generate the correct g^{γ} and w^a since b, c, t_1, t_2 are randomly chosen from Z_q^* and secret for the two servers. The **VExp** algorithm is verifiable with probability 1 for the randomness of b, c, t_1, t_2, where the checkability of the algorithm in [6] is only 2/3, and the malicious server may cheat the outsourcer with probability 1/3. For the outsourcer, we need 12 MM, 5 MInv operations and invoke the *Rand* or *Rand'* six times. So, the proposed scheme improves the checkability of the modular exponentiations though a little computation cost is added.

3.2 Security Analysis

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

Proof. Let $A = (E, U_1', U_2')$ be a PPT adversary that interacts with a PPT algorithm T in the one-malicious model.

First, we prove $EVIEW_{real} \sim EVIEW_{ideal}$, 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.

So, suppose (a, u) is an honest, secret input. The simulator S_1 in the ideal experiment behaves as follows. On receiving the input on round i, S_1 ignores it and instead makes one random query of the form (α_j', β_j') to both U_1' and U_2'. After receiving the outputs of U_1' and U_2', S_1 then submits two random queries of the form (α_j, β_j) to both U_1' and U_2'. Finally, S_1 checks one output $\beta_j'^{\alpha_j'}$ and two outputs $\beta_j^{\alpha_j}$ from each program. If an error is detected, S_1 saves all states and outputs $Y_p^i =$ "$error$", $Y_u^i = \varnothing$, $rep^i = 1$, and thus the final output for ideal process is $(estate^i, "error", \varnothing)$; otherwise, S_1 outputs $Y_p^i = \varnothing$, $Y_u^i = \varnothing$, $rep^i = 0$, and the final output for ideal process is $(estate^i, y_p^i, y_u^i)$.

In addition, we need to show that the inputs to (U_1', U_2') in the real experiment are computationally indistinguishable from that in the ideal one. In the ideal experiment, the inputs are selected uniformly at random. In the real one, each part of all three queries that T makes to any program is generated by invoking the subroutine *Rand* or *Rand'* and thus computationally indistinguishable from random. Therefore, we consider three possible conditions. If (U_1', U_2') both behave honest in round i, $EVIEW_{real}^i \sim EVIEW_{ideal}^i$ since the outputs of **VExp** are not replaced and $rep^i = 0$. If one of (U_1', U_2') is dishonest in round i, the fault must be detected by both T and S_1 with probability 1, resulting in an output of "error". Thus, $EVIEW_{real}^i \sim EVIEW_{ideal}^i$ even when one of (U_1', U_2') misbehaves, so we conclude that $EVIEW_{real} \sim EVIEW_{ideal}$.

Note that if both U_1' and U_2' deviated from their advertised functionalities, this argument would not work. The reason is that while the event that U_1' or U_2' misbehaves is independent of the input (a, u), but the event that both of them misbehaves is not independent of the input (a, u).

Secondly, we prove $UVIEW_{real} \sim UVIEW_{ideal}$, which means that the untrusted software (U_1', U_2') learns nothing during the execution of $(T, (U_1', U_2'))$. In the ideal experiment, the simulator S_2 always behaves as follows: when receiving the input on round i, S_2 ignores it but submits one random query of the form (α_j', β_j') to U_1' and U_2'. After receiving the outputs of U_1' and U_2', S_2 then makes two random queries of the form (α_j, β_j) to U_1' and U_2'. Then S_2 saves its states and those of (U_1', U_2'). Since the honest, secret; or honest, protected; or adversarial, protected inputs are all private for (U_1', U_2'), the simulator S_2 is applicable to all those conditions. As shown in Pair One, the inputs to (U_1', U_2') in the real experiment is computationally indistinguishable from those in the ideal one randomly chosen by S_2. Therefore, $UVIEW_{real}^i \sim UVIEW_{ideal}^i$ for each round i, which means that $UVIEW_{real} \sim UVIEW_{ideal}$.

Theorem 3.2. In the one-malicious model, the algorithm $(T, (U_1, U_2))$ proposed in Section 3.1 is verifiable, that is to say, the outsourcer can test the error with probability 1 if one of the servers outputs the fault result.

Proof. Assume U_2 is a malicious server and U_1 is an honest server. At the end of the algorithm, the outsourcer verifies the result as follows:

$$D_{12} = U_1(\gamma/t_2, g^{t_2}) = U_2(\gamma/t_2, g^{t_2}) = D_{22} \quad (1)$$

$$D_{13} = U_1(at_1/c, w^{c/t_1}) = U_2(at_1/b, w^{b/t_1}) = D_{23} \quad (2)$$

Since the outsourcer sends $\gamma/t_2, g^{t_2}$ to two servers, and U_1 must return true value of g^{γ}, U_2 also computes $g^{\gamma} = (g^{t_2})^{\gamma/t_2}$. Otherwise, the formula (1) cannot pass the verification successfully. Thus, U_2 only possibly cheats the outsourcer during the verification of formula (2).

In our algorithm, the outsourcer splits $u^a = (g^{\beta} g^{\gamma} w^a) \mod p$, and queries $U_1(b/t_1, wg^{t_1})$, $U_2(c/t_1, wg^{t_1})$, where b, c, t_1 are randomly chosen from Z_q. Since U_2 is a malicious server, he may return a fault result $w^{c'/t_1} g^{c'}$ but U_1 returns $w^{b/t_1} g^b$. Next, the outsourcer computes:

$$w^{b/t_1} g^b \cdot (g^b)^{-1} = w^{b/t_1},\ w^{c'/t_1} g^{c'} \cdot (g^c)^{-1} = w^{c'/t_1} g^{c'-c}$$

and queries $U_1(at_1/c, w^{c'/t_1} g^{c'-c})$ and $U_2(at_1/b, w^{b/t_1})$. In order to pass the verification, U_2 must return the same value as U_1. Because U_1 is an honest server, he computes

Table 1: Comparison of the Outsourcing Algorithms for Single Modular Exponentiation

Algorithm	HL [17]	Exp [6]	GExp [26]	**VExp**
MExp(T)	0	0	1	0
MM(T)	9	7	12	13
MInv(T)	5	3	4	3
Invoking subroutine	6	5	7	6
MExp(U)	8	6	4	6
Servers	two	two	one	two
Checkability	1/2	2/3	1/2	1

$$U_1(at_1/c, w^{c'/t_1}g^{c'-c}) \rightarrow w^{\frac{ac'}{c}}g^{\frac{at_1c'}{c}-at_1}.$$

If the formula (2) can be verified successfully, that is,

$$U_1(at_1/c, w^{c'/t_1}g^{c'-c}) = U_2(at_1/b, w^{b/t_1}) = w^{\frac{ac'}{c}}g^{\frac{at_1c'}{c}-at_1}.$$

U_2 must return $w^{\frac{ac'}{c}}g^{\frac{at_1c'}{c}-at_1}$, therefore he must know t_1, b, and then forges c' and computes a. As shown in Section 3.1, t_1, b are randomly chosen from Z_q by the *Rand* and *Rand'*, and thus secret and uniformly distributed for U_2. The outsourcer can verify the returned result with probability 1.

Therefore, U_2 cannot pass the verification of formula (1) and (2) simultaneously if he returns the error result and the proposed algorithm is verifiable with probability 1.

Theorem 3.3. In the one-malicious model, the algorithm proposed in Section 3.1 is an $(O(\frac{\log^2 L}{L}), 1)$-outsource-secure implementation of **VExp**.

Proof. The proposed algorithm **VExp** makes 6 calls to *Rand* or *Rand'*, and 13 modular multiplication (MM) and 3 modular inverse (MInv) in order to compute $u^a \mod p$. As shown in [17], **VExp** takes $O(\log L)$ or $O(1)$ **MM** using the **EBPV** generator or table-lookup method, respectively, where L is the bit length of a. It is well known that it takes roughly $1.5L$ **MM** to compute $u^a \mod p$ by the square-and-multiply method. Thus, the proposed algorithm is an $(O(\frac{\log^2 L}{L}))$-efficient implementation of **VExp**. On the other hand, it must be detected with probability 1 if U_1 or U_2 fails during any execution of **VExp** from Theorem 3.2.

3.3 Comparison

We compare the outsourcing algorithms for single modular exponentiation with input privacy in Table 1. In Table 1, "**MExp**, **MM**, **MInv**" denote the computation of modular exponentiation, modular multiplication and modular inverse, respectively. We omit other operations such as modular addition in the outsourcing algorithms.

From Table 1, we conclude that the **GExp** algorithm [26] has no advantage for single modular exponentiation since the outsourcer has to execute one modular exponentiation although it is based on a single untrusted server. Compared with the algorithm in [17], the proposed algorithm **VExp** improves checkability and efficiency simultaneously since MInv operation needs more computation cost than MM. The **VExp** algorithm also improves the checkability though a little computation is appended compare with the algorithm in [6]. In detail, the checkability of our algorithm is 1, where it is 1/2 and 2/3 in the algorithm of [17] and [6].

4. VERIFIABLE SECURE OUTSOURCING OF MULTIPLE MODULAR EXPONENTIATIONS

In this section, we propose the other verifiable outsourcing algorithm to compute multiple modular exponentiations $u_1^{a_1}u_2^{a_2}\ldots u_n^{a_n} \mod p (n \geq 2)$, which is applicative in many cryptographic schemes, such as multivariate polynomial evaluation [19], attributed-based encryption [24], provable data possession (PDP) [25, 27], and so on.

4.1 Verifiable Outsourcing Algorithm

We propose a verifiable outsourcing algorithm of multiple modular exponentiations **VMExp** in the one-malicious model.

Let p, q be two large primes and $q|p-1$. Given n arbitrary bases $u_1, u_2, \ldots, u_n \in Z_p^*$ and n exponents $a_1, a_2, \ldots, a_n \in Z_q$ such that the order of $u_1, u_2, \ldots, u_n$ is q. The proposed algorithm **VMExp** is described as follows:

1) T firstly runs *Rand* to create four blinding pairs (α, g^α), (β, g^β), (t_1, g^{t_1}), (t_2, g^{t_2}), and then runs *Rand'* to create two blinding tuples (b, b^{-1}, g^{-b}), (c, c^{-1}, g^{-c}). We denote:

$$v = g^\alpha \mod p,\ \mu = g^\beta \mod p.$$

2) T splits:

$$\begin{aligned} u_1^{a_1}u_2^{a_2}\ldots u_n^{a_n} &= (vw_1)^{a_1}(vw_2)^{a_2}\ldots(vw_n)^{a_n} \\ &= g^{\alpha(a_1+\ldots+a_n)}w_1^{a_1}\ldots w_n^{a_n} \\ &= g^\beta g^\gamma w_1^{a_1}w_2^{a_2}\ldots w_n^{a_n} \end{aligned}$$

where $w_1 = u_1/v, w_2 = u_2/v, \ldots, w_n = u_n/v$,

$$\gamma = [(a_1 + \ldots + a_n)\alpha - \beta] \mod q.$$

3) Next, T randomly chooses $i \in \{1, 2, \ldots, n\}$ and queries U_1 in random order as:

$$U_1(b/t_1, w_i g^{t_1}) \rightarrow D_{111} = w_i^{b/t_1}g^b,$$
$$U_1(b/t_1, (\Pi_{j=1,j\neq i}^n w_j)g^{t_1}) \rightarrow D_{112} = (\Pi_{j=1,j\neq i}^n w_j)^{b/t_1}g^b.$$

Similarly, T queries U_2 in random order as:

$$U_2(c/t_1, w_i g^{t_1}) \rightarrow D_{211} = w_i^{c/t_1}g^c,$$
$$U_2(c/t_1, (\Pi_{j=1,j\neq i}^n w_j)g^{t_1}) \rightarrow D_{212} = (\Pi_{j=1,j\neq i}^n w_j)^{c/t_1}g^c.$$

4) After receiving D_{111}, D_{112} and D_{211}, D_{212} from U_1 and U_2 respectively, T computes:

$$w_i^{b/t_1} = D_{111}g^{-b},\ (\Pi_{j=1,j\neq i}^n w_j)^{b/t_1} = D_{112}g^{-b},$$
$$w_i^{c/t_1} = D_{211}g^{-c},\ (\Pi_{j=1,j\neq i}^n w_j)^{c/t_1} = D_{212}g^{-c}.$$

Then T queries U_1 in random order as:

$$U_1(\gamma/t_2, g^{t_2}) \rightarrow D_{12} = g^\gamma,$$
$$U_1(a_1 - c/t_1, w_1) \rightarrow D_{131} = w_1^{a_1-c/t_1},$$
$$\ldots,$$
$$U_1(\tfrac{a_i t_1}{c}, w_i^{c/t_1}) \rightarrow D_{13i} = w_i^{a_i},$$
$$\ldots,$$
$$U_1(a_n - c/t_1, w_n) \rightarrow D_{13n} = w_n^{a_n-c/t_1}.$$

Similarly, T queries U_2 in random order as:

$$U_2(\gamma/t_2, g^{t_2}) \to D_{22} = g^{\gamma},$$
$$U_2(a_1 - b/t_1, w_1) \to D_{231} = w_1^{a_1 - c/t_1},$$
$$\ldots,$$
$$U_2(\tfrac{a_i t_1}{b}, w_i^{b/t_1}) \to D_{23i} = w_i^{a_i},$$
$$\ldots,$$
$$U_2(a_n - b/t_1, w_n) \to D_{23n} = w_n^{a_n - b/t_1}.$$

5) T computes:

$$\begin{aligned} D_{13} &= D_{131} \ldots D_{13i} \ldots D_{13n} (\prod_{j=1, j\neq i}^{n} w_j)^{c/t_1} \\ &= w_1^{a_1 - c/t_1} \ldots w_i^{a_i} \ldots w_n^{a_n - c/t_1} (\prod_{j=1, j\neq i}^{n} w_j)^{c/t_1} \\ &= w_1^{a_1} w_2^{a_2} \ldots w_n^{a_n} \end{aligned}$$

$$\begin{aligned} D_{23} &= D_{231} \ldots D_{23i} \ldots D_{23n} (\prod_{j=1, j\neq i}^{n} w_j)^{b/t_1} \\ &= w_1^{a_1 - b/t_1} \ldots w_i^{a_i} \ldots w_n^{a_n - b/t_1} (\prod_{j=1, j\neq i}^{n} w_j)^{b/t_1} \\ &= w_1^{a_1} w_2^{a_2} \ldots w_n^{a_n} \end{aligned}$$

and then verifies :

$$D_{12} = D_{22},\ D_{13i} = D_{23i},\ D_{13} = D_{23}.$$

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

$$u_1^{a_1} u_2^{a_2} \ldots u_n^{a_n} = g^{\beta} g^{\gamma} w_1^{a_1} w_2^{a_2} \ldots w_n^{a_n} \mod p.$$

Remark 2. In fact, outsourcing multiple modular exponentiations $u_1^{a_1} u_2^{a_2} \ldots u_n^{a_n}$ can be executed by invoking single modular exponentiation u^a for n times, which requires $13n$ MM and $3n$ MInv operations for the outsourcer, where the **VMExp** algorithm only needs $4n + 13$ MM and 3 MInv operations. Thus, the proposed **VMExp** algorithm is much more efficient than invoking single modular exponentiation directly for $n \geq 2$.

4.2 Security Analysis

Theorem 4.1. In the one-malicious model, the proposed algorithm $(T, (U_1, U_2))$ is an outsource-secure implementation of **VMExp**, where the input $(a_1, \ldots, a_n; u_1, \ldots, u_n)$ may be honest, secret; honest, protected; or adversarial, protected.

Proof. The proof is similar to Theorem 3.1, and the difference is that the real experiment may be corrupted. Let $A = (E, U_1', U_2')$ be a PPT adversary that interacts with a PPT algorithm T in the one-malicious model.

First, we prove $EVIEW_{real} \sim EVIEW_{ideal}$, which means that the environment E learns nothing during the execution of $(T, (U_1, U_2))$. If the input $(a_1, \ldots, a_n; u_1, \ldots, u_n)$ 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 $(a_1, \ldots, a_n; u_1, \ldots, u_n)$ is an honest, secret input.

So, suppose $(a_1, \ldots, a_n; u_1, \ldots, u_n)$ is an honest, secret input. The simulator S_1 in the ideal experiment behaves as follows. On receiving the input on round i, S_1 ignores it and instead makes two random query of the form (α_j', β_j') to both U_1' and U_2'. After receiving the outputs of U_1' and U_2', S_1 then submits $n + 1$ random queries of the form (α_j, β_j) to both U_1' and U_2'. Finally, S_1 checks two outputs $\beta_j'^{\alpha_j'}$ and two random outputs $\beta_j^{\alpha_j}$ from each program. If an error is detected, S_1 outputs $Y_p^i = error$, $Y_u^i = \varnothing$, $rep^i = 1$. If no error is detected, S_1 verifies the remaining $n - 1$ outputs. If all checks pass, S_1 outputs $Y_p^i = \varnothing$, $Y_u^i = \varnothing$, $rep^i = 0$; else, S_1 chooses a random element $r \in Z_p^*$, and outputs $Y_p^i = r$, $Y_u^i = \varnothing$, $rep^i = 1$. In either condition, S_1 saves its own states and those of (U_1', U_2').

In addition, we need to show that the inputs to (U_1', U_2') in the real experiment are computationally indistinguishable from those in the ideal one. In the ideal experiment, the inputs are selected uniformly at random. In the real one, each part of all $n + 3$ queries that T makes to any program is generated by invoking the subroutine *Rand* or *Rand'* and thus computationally indistinguishable from random. Therefore, we consider three possible conditions. If (U_1', U_2') both behave honest in round i, $EVIEW_{real}^i \sim EVIEW_{ideal}^i$ since the outputs of **VMExp** are not replaced and $rep^i = 0$. If one of (U_1', U_2') is dishonest in round i, the fault must be detected by both T and S_1 with probability $1 - \frac{1}{2n(n+1)}$, resulting in an output of "error". Thus, $EVIEW_{real}^i \sim EVIEW_{ideal}^i$ also holds even when one of (U_1', U_2') misbehaves in the round i, so we conclude that $EVIEW_{real} \sim EVIEW_{ideal}$.

Secondly, we prove $EVIEW_{real} \sim EVIEW_{ideal}$, which means that the untrusted software (U_1', U_2') learns nothing during the execution of $(T, (U_1', U_2'))$. In the ideal experiment, the simulator S_2 always behaves as follows: when receiving the input on round i, S_2 ignores it but submits two random query of the form (α_j', β_j') to U_1' and U_2'. After receiving the outputs of U_1' and U_2', S_2 then makes $n + 1$ random queries of the form (α_j, β_j) to U_1' and U_2'. Then S_2 saves its states and those of (U_1', U_2'). Since the honest, secret; or honest, protected; or adversarial, protected inputs are all private for (U_1', U_2'), the simulator S_2 is applicable to all those conditions. As we know, E can easily distinguish the real and ideal experiments since the outputs of the ideal experiment are never corrupted, but he cannot send the information to (U_1', U_2') since they cannot communicate each other during the execution of T^U. In addition, the inputs to (U_1', U_2') in the real experiment are computationally indistinguishable from those in the ideal one which are randomly chosen by S_2. Therefore, $UVIEW_{real}^i \sim UVIEW_{ideal}^i$ for each round i. In all, we conclude that $UVIEW_{real} \sim UVIEW_{ideal}$.

Theorem 4.2. In the one-malicious model, the proposed algorithm $(T, (U_1, U_2))$ is verifiable, which means that the outsourcer can detect the error with probability $1 - \frac{1}{2n(n+1)}$ if one of the servers outputs the fault result.

Proof. Similarly, we assume that U_2 is a malicious server and U_1 is an honest server. At the end of the algorithm, the outsourcer verifies the result as follows:

$$D_{12} = U_1(\gamma/t_2, g^{t_2}) = U_2(\gamma/t_2, g^{t_2}) = D_{22} \quad (3)$$

$$D_{13i} = U_1(a_i t_1/c, w_i^{c/t_1}) = U_2(a_i t_1/b, w_i^{b/t_1}) = D_{23i} \quad (4)$$

$$\begin{aligned} D_{13} &= D_{131} \ldots D_{13i-1} D_{13i} \ldots D_{13n} (\textstyle\prod_{j=1, j\neq i}^{n} w_j)^{c/t_1} \\ &= D_{231} \ldots D_{23i-1} D_{23i} \ldots D_{23n} (\textstyle\prod_{j=1, j\neq i}^{n} w_j)^{b/t_1} \\ &= D_{23} \end{aligned} \quad (5)$$

As proven in Theorem 3.2, U_2 must return the same value as U_1 during the verification of the formula (3) and (4). Thus, U_2 only possibly cheats the outsourcer during the verification of formula (5).

In our algorithm, the outsourcer splits

$$u_1^{a_1}u_2^{a_2}\dots u_n^{a_n} = (g^{\beta}g^{\gamma}w_1^{a_1}w_2^{a_2}\dots w_n^{a_n}) \mod p,$$

and queries

$$U_1(b/t_1, (\prod_{j=1,j\neq i}^{n} w_j)g^{t_1}),\ U_2(c/t_1, (\prod_{j=1,j\neq i}^{n} w_j)g^{t_1}),$$

where b, c, t_1 are randomly chosen from Z_q^*. Since U_2 is a malicious server, he may return a fault result D'_{212}, but U_1 returns true result $D_{112} = (\prod_{j=1,j\neq i}^{n} w_j)^{b/t_1}g^b$. Next, the outsourcer computes:

$$D_{112}(g^b)^{-1} = (\prod_{j=1,j\neq i}^{n} w_j)^{b/t_1},\ D'_{212}(g^c)^{-1},$$

and queries U_1, U_2 about $D_{131}, \dots, D_{13i-1}, D_{13i+1}, \dots, D_{13n}$ and $D_{231}, \dots, D_{23i-1}, D_{23i+1}, \dots, D_{23n}$, respectively. Because U_1 is an honest server, he computes the true values of $D_{131}, \dots, D_{13i-1}, D_{13i+1}, \dots, D_{13n}$. If the formula (5) can be verified successfully, that is to say,

$$\begin{aligned} D_{13} &= D_{131}\dots D_{13i-1}D_{13i}D_{13i+1}\dots D_{13n}D'_{212}g^{-c} \\ &= D'_{231}\dots D'_{23i-1}D_{23i}D'_{23i+1}\dots D'_{23n}D_{112}g^{-b} \\ &= D'_{23}, \end{aligned}$$

which means

$$\frac{D'_{212}}{D'_{231}\dots D'_{23i-1}D'_{23i+1}\dots D'_{23n}} = \frac{D_{23i}D_{112}g^{c-b}}{D_{131}\dots D_{13i-1}D_{13i}\dots D_{13n}}.$$

As we know, U_2 can compute the true values of $D_{231}, \dots, D_{23i-1}, D_{23i+1}, \dots, D_{23n}$ if he behaves honestly, and

$$\begin{aligned} D_{13} &= D_{131}\dots D_{13i-1}D_{13i}D_{13i+1}\dots D_{13n}D_{212}g^{-c} \\ &= D_{231}\dots D_{23i-1}D_{23i}D_{23i+1}\dots D_{23n}D_{112}g^{-b} \\ &= D_{23}, \end{aligned}$$

so U_2 can obtain

$$\frac{D_{212}}{D_{231}\dots D_{23i-1}D_{23i+1}\dots D_{23n}} = \frac{D_{23i}D_{112}g^{c-b}}{D_{131}\dots D_{13i-1}D_{13i}\dots D_{13n}}.$$

Therefore, U_2 can forge fault results of

$$D_{212} \text{ and } D_{231}\dots D_{23i-1}D_{23i+1}\dots D_{23n},$$

which are denoted as D'_{212} and $D'_{231}\dots D'_{23i-1}D'_{23i+1}\dots D'_{23n}$, to make the formula (5) hold and generate the wrong value of D_{13} if he knows two values of

$$D_{212} \text{ and } D_{231}\dots D_{23i-1}D_{23i+1}\dots D_{23n}$$

from queries in random order executed by the outsourcer.

As shown in Section 4.1, the outsourcer first executes two queries D_{211}, D_{212} and then $n+1$ queries $D_{22}, D_{231}, \dots, D_{23i-1}, D_{23i+1}, \dots, D_{23n}$ to U_2. It is obvious that U_2 can guess D_{212} from D_{211}, D_{212} truly with probability 1/2. In addition, U_2 can obtain the true value of

$$D_{231}\dots D_{23i-1}D_{23i+1}\dots D_{23n}$$

from $n+1$ queries $D_{22}, D_{231}, \dots, D_{23i-1}, D_{23i+1}, \dots, D_{23n}$ with probability $\frac{1}{n(n+1)}$ since he can guess D_{22}, D_{23i} from $n+1$ queries correctly and computes the product of other $n-1$ queries with probability $\frac{1}{n(n+1)}$. Therefore, the malicious server U_2 can cheat the outsourcer with probability $\frac{1}{2n(n+1)}$ and thus the outsourcer can verify the returned result with probability $1-\frac{1}{2n(n+1)}$.

Therefore, U_2 can pass the verification of formula (3), (4) and (5) simultaneously if he returns the error results with probability $\frac{1}{2n(n+1)}$ and the proposed algorithm is verifiable with probability $1-\frac{1}{2n(n+1)}$.

Theorem 4.3. In the one-malicious model, the proposed algorithm $(T, (U_1, U_2))$ is an $(O(\frac{\log^2 L+n}{nL}), 1-\frac{1}{2n(n+1)})$ -outsource-secure implementation of **VMExp**.

Proof. As we know, the proposed algorithm **VMExp** makes 6 calls to *Rand* or *Rand'* and $4n+13$ modular multiplication (MM) and 3 modular inverse (MInv) in order to compute $u_1^{a_1}u_2^{a_2}\dots u_n^{a_n}$, so **VMExp** algorithm takes $O(n)$ or $O(\log^2 L+n)$ MM using table-lookup method or the **EBPV** generator, respectively, where L is the bit length of $a_1, \dots, a_n$. As shown in [16], it takes roughly $0.2nL$ MM to compute $u_1^{a_1}u_2^{a_2}\dots u_n^{a_n}$ by joint sparse form (JSF) using windows method with window width 4. Therefore, the proposed algorithm **VMExp** is an $(O(\frac{\log^2 L+n}{nL}))$-efficient implementation of multiple modular exponentiations. On the other hand, we know that it must be detected with probability $1-\frac{1}{2n(n+1)}$ if U_1 or U_2 misbehaves during any execution of **VMExp** from Theorem 4.2.

4.3 Comparison

In Table 2, we compare the outsourcing algorithms for multiple modular exponentiations with input privacy proposed in [26] and this paper.

Table 2: Comparison of the Outsourcing Algorithms for Multiple Modular Exponentiations

Algorithm	GExp [26]	**VMExp**
MExp(T)	1	0
MM(T)	$4n+9$	$4n+13$
MInv(T)	$2n+2$	3
Invoking subroutine	7	6
MExp(U)	$2n+2$	$2n+6$
Servers	one	two
Checkability	$\frac{1}{n+1}$	$1-\frac{1}{2n(n+1)}$

As described in Table 1, "MExp, MM, MInv" denote the computation of modular exponentiation, modular multiplication and modular inverse, respectively.

From Table 2, we conclude that **VMExp** algorithm greatly improves the checkability and efficiency of outsourcing computation for multiple modular exponentiations by using two servers in the one-malicious model. Compare with the algorithm presented in [26], we need no MExp operations, and only need executing $4n+13$ MM, 3 MInv, 6 invocations of *Rand* or *Rand'*, and $2n+6$ queries to U_1 and U_2 for n modular exponentiations. Moreover, the **VMExp** algorithm can verify the result returned by the server with probability $1-\frac{1}{2n(n+1)}$ where the checkability is only $\frac{1}{n+1}$ in the algorithm of [26].

5. APPLICATIONS

In this section, we introduce two applications of the proposed **VExp** and **VMExp** algorithm, including private outsourcing of univariate and multivariate polynomial evalua-

tion and CP-ABE scheme with verifiable outsourced encryption and decryption.

5.1 Private Outsourcing Scheme of Polynomial Evaluation

Polynomial evaluation is an important tool in constructing many cryptographic protocols, such as proof of retrievability [18], verifiable keyword search [3] and so on. In general, the evaluation of polynomials does not need high computation complexity, but the magnitude of data prevents the client to evaluate them itself when the polynomial or the input are derived from large datasets.

Benabbas et al. [2] presented the outsourcing scheme of polynomial evaluation, which can protect the polynomials themselves, but cannot realize the input privacy . Fiore et al. constructed an outsourcing scheme of polynomial evaluation with public verifiability, which means that anyone can verify the correctness of computing result returned from the server. However, the scheme may reveal the input or the polynomial [10]. Recently, Zhang et al. described a verifiable outsourcing scheme of polynomial evaluation using multilinear maps [11, 12], and the polynomial function and the input are also private for the server [28]. All of the outsourcing schemes are based on homomorphic encryption (HE) scheme [1], which is costly for the outsourcer with weak computation ability in practice.

A direct application of **VExp** or **VMExp** algorithm is to evaluate the univariate or multivariate polynomials over large datasets without using HE scheme. The proposed schemes are as follows:

Assume the outsourcer T wants to evaluate the univariate polynomial of d degree $f(x) = \sum_{i=0}^{d} f_i x^i$, it first runs

$$\mathbf{VExp}(i; x) \to f(i) = x^i \text{ for } 0 \le i \le d,$$

and then computes $f(x) = \sum_{i=0}^{d} f_i x^i$.

Similarly, if T wants to evaluate the polynomial of n variables and d degree

$$f(x_1, \ldots, x_n) = \sum_{0 \le i_1 + \ldots + i_n \le d} f_{i_1,\ldots,i_n} x_1^{i_1} \ldots x_n^{i_n},$$

it first runs

$$\mathbf{VMExp}(i_1, \ldots, i_n; x_1, \ldots, x_n) \to f(i_1, \ldots, i_n) = x_1^{i_1} \ldots x_n^{i_n}$$

for $0 \le i_1 + \ldots + i_n \le d$, and then computes

$$f(x_1, \ldots, x_n) = \sum_{0 \le i_1 + \ldots + i_n \le d} f(i_1, \ldots, i_n).$$

Remark 3. The proposed schemes above are verifiable since **VExp** and **VMExp** algorithm are verifiable with probability 1 or close to 1. Moreover, it realizes input privacy and function privacy simultaneously without HE scheme. The disadvantage is that the proposed schemes are interactive based on two servers while the scheme described in [28] is non-interactive and can verify the outsourcing result with probability 1 based on one server since an HE scheme is required.

5.2 CP-ABE scheme with verifiable outsourced encryption and decryption

Sahai and Waters [24] introduced the notion of attribute-based encryption (ABE) for complex access-control over encrypted data. The main efficiency drawback of the most existing ABE schemes is that encryption and decryption are expensive for resource-limited devices due to modular exponentiation and pairing operations. Green et al. [15] and Lai [20] proposed ABE schemes with outsourced decryption that largely eliminates the decryption overhead for users. Currently, there is no ABE scheme with outsourcing encryption and decryption simultaneously proposed. Now we present a verifiable ABE scheme with outsourcing encryption and decryption based on two servers U_1, U_2 in the one-malicious model using **VExp** and **VMExp** algorithm.

The proposed outsourcing scheme of CP-ABE consists of the following algorithms:

Setup(λ, D): Take as input a security parameter λ and a small universe description $D = \{1, 2, \ldots, l\}$. It first runs $G(\lambda)$ to obtain (p, G, G_T, e), where G and G_T are cyclic groups of prime order p and e is a bilinear map from G and G to G_T, and randomly chooses $g \in G$, $\alpha, a \in Z_p^*$. It also chooses random $s_i \in Z_p^*$ for each attribute $i \in D$, and computes $T_i = g^{s_i}$. The public parameters are published as:

$$PK = (G, G_T, e, g, g^\alpha, e(g,g)^\alpha, T_i (i \in D)),$$

and the master secret key is $MSK = \alpha$.

KeyGen(PK, MSK, S): For a set of attributes S, picks random $t \in Z_p^*$, and computes:

$$K = g^\alpha g^{at},\ K_0 = g^t,\ K_i = T_i^t (i \in S),$$

So the private key $SK_S = (S, K, K_0, K_i (i \in S))$.

Encrypt(PK, M, A): Take as input the public parameters PK, a message $M \in G_T$, and an LSSS access structure $A = (A, \rho)$, where A is an $l \times n$ matrix and ρ is a map from each row A_i of A to an attribute $\rho(i)$, it randomly chooses a vector $\vec{v} = (s, v_2, \ldots, v_n) \in (Z_p^*)^n$. For each row A_i of A, the outsourcer T generates the ciphertext as follows:

1) T runs

$$\mathbf{VExp}(s; e(g,g)^\alpha) \to C_1'' = e(g,g)^{\alpha s},$$
$$\mathbf{VExp}(s; g) \to C_1' = g^s,$$

and computes $C_1 = M \cdot C_1''$.

2) T runs *Rand* for l times to obtain $(r_i, g^{r_i}), i \in \{1, \ldots, l\}$, and sets $D_i = g^{r_i}$.

3) T runs

$$\mathbf{VMExp}(A_i \vec{v}, -r_i; g^a, T_{\rho(i)}) \to C_i = g^{aA_i\vec{v}} T_{\rho(i)}^{-r_i},$$

where $i \in \{1, \ldots, l\}$.

The final ciphertext $CT = ((A, \rho), C_1, C_1', C_i, D_i)$.

Decrypt(PK, SK_S, CT): Take as input the public keys PK, a private key $SK_S = (S, K, K_0, K_i (i \in S))$ for a set of attributes S, and a ciphertext $CT = ((A, \rho), C_1, C_1', C_i, D_i)$ for an access structure $A = (A, \rho)$. If S does not satisfy the access structure A, it output $\perp$. Else, it computes constant $\omega_i \in Z_p^*$ such that $\sum_{i \in I} \omega_i A_i = (1, 0, \ldots, 0)$, where $I = \{i : \rho(i) \in S\}$, and $I \subset \{1, 2, \ldots, l\}$. Then it decrypts:

$$\begin{aligned} M &= C_i \cdot \frac{\prod_{i \in I} (e(C_i, K_0) e(K_{\rho(i)}, D_i))^{\omega_i}}{e(C_1', K)} \\ &= M \cdot e(g,g)^{\alpha s} \cdot \frac{\prod_{i \in I} (e(g,g)^{atA_i v \omega_i})}{e(g,g)^{\alpha s} e(g,g)^{ats}}. \end{aligned}$$

If a user T' wants to outsource the decryption of the ciphertext, it executes the following algorithms:

GenTK$_{out}(PK, SK_S)$: Take as input the public parameters PK, a private key $SK_S = (S, K, K_0, K_i (i \in S))$ for a set of attributes S. The outsourcer T' randomly chooses $z_1, z_2 \in Z_p^*$, runs **VExp** algorithm as follows:

$$\mathbf{VExp}(1/z_1; K) \to K'_1 = K^{1/z_1},$$
$$\mathbf{VExp}(1/z_2; K) \to K'_2 = K^{1/z_2},$$
$$\mathbf{VExp}(1/z_1; K_0) \to K'_{10} = K_0^{1/z_1},$$
$$\mathbf{VExp}(1/z_2; K_0) \to K'_{20} = K_0^{1/z_2},$$
$$\mathbf{VExp}(1/z_1; K_i) \to K'_{1i} = K_i^{1/z_1}, i \in S,$$
$$\mathbf{VExp}(1/z_2; K_i) \to K'_{2i} = K_i^{1/z_2}, i \in S,$$

and sets two transformation keys and retrieving keys as:

$$TK_{S1} = (S, K'_1 = K_{1/z_1}, K'_{10} = K_0^{1/z_1}, K'_{1i} = K_i^{1/z_1}),$$
$$TK_{S2} = (S, K'_2 = K_{1/z_2}, K'_{20} = K_0^{1/z_2}, K'_{2i} = K_i^{1/z_2}),$$
$$RK_{S1} = z_1, RK_{S2} = z_2.$$

Then T' sends TK_{S1}, TK_{S2} to the servers U_1, U_2, respectively.

Transformout$_{out}(PK, CT, TK_S)$: Take as input the public parameters PK, a ciphertext $CT = ((A, \rho), C_1, C'_1 C_i, D_i)$ for an access structure $A = (A, \rho)$ and a transformation key $TK_{S1} = (S, K'_1, K'_{10}, K'_{1i})$. The server U_1 computes:

$$\begin{aligned} T'_1 &= \frac{e(C'_1, K'_1)}{\prod_{i\in I}(e(C_i, K'_{10})e(K'_{1,\rho(i)}, D_i))^{\omega_i}} \\ &= \frac{e(g,g)^{\alpha s/z_1} e(g,g)^{ats/z_1}}{\prod_{i\in I} e(g,g)^{atA_i \vec{v}\omega_i/z_1}} \\ &= e(g,g)^{\alpha s/z_1}, \end{aligned}$$

and outputs the transformed ciphertext as

$$CT'_1 = (T_1 = C_1, T'_1).$$

Similarly, the server U_2 computes:

$$T'_2 = \frac{e(C'_1, K'_2)}{\prod_{i\in I}(e(C_i, K'_{20})e(K'_{2,\rho(i)}, D_i))^{\omega_i}} = e(g,g)^{\alpha s/z_2},$$

and outputs the transformed ciphertext as

$$CT'_2 = (T_2 = C_1, T'_2).$$

Decrypt$_{out}(PK, CT, CT'_1, CT'_2, RK_{S1}, RK_{S2})$: Take the public keys PK, a ciphertext $CT = ((A, \rho), C_1, C'_1, C_i, D_i)$, two transformed ciphertexts $CT'_1 = (T_1 = C_1, T'_1), CT'_2 = (T_2 = C_1, T'_2)$, and two retrieving keys $RK_{S1} = z_1, RK_{S2} = z_2$ for a set of attributes S as input. If $T_1 \neq C_1$ or $T_2 \neq C_1$, the outsourcer T' outputs $\perp$. Else, T' computes

$$M = T_1/(T'_1)^{z_1}, M' = T_2/(T'_2)^{z_2}.$$

If $M = M'$, outputs the message M, else, it outputs $\perp$.

Remark 4. In the one-malicious model, it is obvious that the result of outsourcing encryption is verifiable because of the verifiability of **VExp** and **VMExp** algorithm. The result of the outsourcing decryption is also verifiable since the outsourcer T' can check the error by verifying $T_1/(T'_1)^{z_1} = T_2/(T'_2)^{z_2}$ if one of the servers outputs the fault result.

6. PERFORMANCE EVALUATION

In this section, we provide an experimental evaluation of the proposed outsourcing algorithms. Our experiment is simulated on two machines with Intel Core i5 Processor running at 3.1GHz with 4G memory (cloud server), and Pentium T4300 Processor running at 2.1GHz with 2G memory (the outsourcer), respectively. The programming language is JAVA.

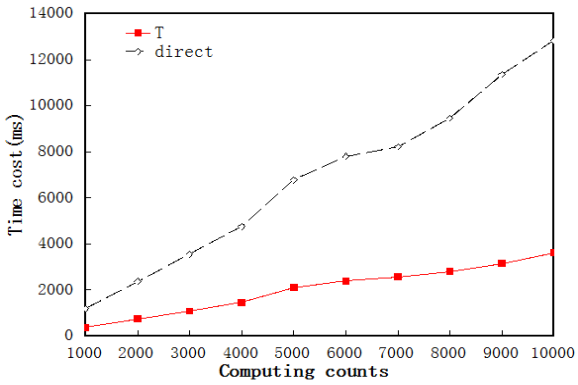

Figure 1: Simulation for VExp Algorithm.

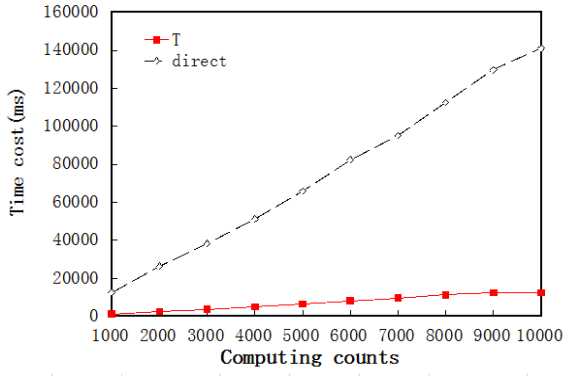

Figure 2: Simulation for VMExp Algorithm.

The parameters of p and q are same to Federal Information Processing Standards for DSA (FIPS-186-2) [6]. That is, p is a 512-bit prime and $q|p-1$ is a 160-bit prime.

In Fig.1, we provide the simulation of VExp algorithm, where the fault can be found with probability 1 if one of the servers misbehaves. We also present the simulation of **VMExp** algorithm for $n = 10$ in Fig.2, which shows that the outsourcer can check the error with probability 0.9955 if one of the servers returns the false result when computing $u_1^{a_1} u_2^{a_2} \ldots u_{10}^{a_{10}}$. It is obvious that the time cost for T is much smaller than that for directly computing single modular exponentiation and multiple modular exponentiations since that a number of computations have been delegated to two servers. Therefore, the proposed **VExp** and **VMExp** algorithm are the implementations of secure verifiable outsourcing single modular exponentiation and multiple modular exponentiations.

In Table 3, we compare the evaluation time for the outsourcing algorithms of single modular exponentiation proposed in [6, 17, 26] and this paper, respectively. From Table 3, we conclude that for the outsourcer T, the **VExp** algorithm is superior to HL and GExp algorithm in efficiency, and it appends little computation cost to improve the checkability compared with **Exp** algorithm. We also compare the time for the outsourcing algorithms of multiple modular exponentiations proposed in [26] and this paper in Table 4. Similarly, for the outsourcer T, the proposed **VMExp** algorithm is more efficient than the GExp algorithm. Thus,

Table 3: Time Comparison for Single Modular Exponentiation Algorithms

	HL [17]	Exp [6]	GExp [26]	**VExp**
T(ms)	4.206	2.570	4.359	3.038
U_1(ms)	4.601	3.529	3.219	3.549
U_2(ms)	4.549	3.575	0	3.558

Table 4: Time Comparison for Multiple Modular Exponentiations Algorithms

n	**VMExp**(T)(ms)	GExp(T)(ms) [26]
2	2.296	7.160
3	3.014	7.455
4	3.494	7.642
5	3.699	7.802
6	3.961	8.096
7	4.184	8.330
8	4.447	8.723
9	4.884	8.902
10	5.011	9.376

the proposed **VExp** and **VMExp** algorithm improves the checkability and efficiency for the outsourcer simultaneously based on two servers in the one-malicious model.

7. CONCLUSION

In this paper, we propose two verifiable outsource-secure algorithms for single modular exponentiation and multiple modular exponentiations. The security model of our proposed algorithms is based on two non-colluding servers, and the outsourcer can detect any failure with probability 1 or close to 1 if one of the servers misbehaves. Compare with the previous ones, the proposed algorithms improve both the checkability and efficiency for the outsourcer.

8. ACKNOWLEDGMENTS

This work was supported by the National Natural Science Foundation of China (Grant No. 61472235, 61572309, U1536108), Doctoral Fund of Ministry of Education of China (Grant No. 20120073110094), and the Innovation Program of Shanghai Municipal Education Commission (Grant No. 14YZ020).

9. REFERENCES

[1] M. Barbosa and P. Farshim. Delegatable homomorphic encryption with applications to secure outsourcing of computation. In *CT-RSA 2012*, pages 296–312. Springer, 2012.

[2] S. Benabbas, R. Gennaro, and Y. Vahlis. Verifiable delegation of computation over large datasets. In *CRYPTO 2011*, pages 111–131. Springer, 2011.

[3] D. Boneh, G. Crescenzo, and R. Ostrovsky. Public key encryption with keyword search. In *CRYPTO 2004*, pages 506–522. Springer, 2004.

[4] D. Chaum and T. Pedersen. Wallet databases with observers. In *CRYPTO 1992*, pages 89–105. Springer, 1993.

[5] X. Chen, X. Huang, J. Li, J. Ma, W. Lou, and D. Wong. New algorithms for secure outsourcing of large-scale systems of linear equations. *IEEE Transactions on Information Forensics and Security*, 10(1):69–78, 2015.

[6] X. Chen, J. Li, J. Ma, Q. Tang, and W. Lou. New algorithms for secure outsourcing of large-scale systems of linear equations. *IEEE Transactions on Parallel and Distributed Systems*, 25(9):2386–2396, 2014.

[7] X. Chen, W. Susilo, J. Li, D. Wong, and et al. Efficient algorithms for secure outsourcing of bilinear pairings. *Theoretical Computer Science*, 562:112–121, 2015.

[8] K. Chung, Y. Kalai, and S.Vadhan. Improved delegation of computation using fully homomorphic encryption. In *CRYPTO 2010*, pages 483–501. Springer, 2010.

[9] M. Dijk, D. Clarke, B. Gassend, G. Suh, and S. Devadas. Speeding up exponentiation using an untrusted computational resource. *Designs, Codes and Cryptography*, 39(2):253–273, 2006.

[10] D. Fiore and R. Gennaro. Publicly verifiable delegation of large polynomials and matrix computations with applications. In *ACM CCS 2012*, pages 501–512. ACM, 2012.

[11] S. Garg, C. Gentry, and S. Halevi. Candidate multilinear maps from ideal lattices. In *ACM CCS 2012*, pages 1–17. Springer, 2013.

[12] S. Garg, C. Gentry, S. Halevi, A. Sahai, and B. Waters. Attribute-based encryption for circuits from multilinear maps. In *CRYPTO 2013*, pages 479–499. Springer, 2013.

[13] R. Gennaro, C. Gentry, and B. Parno. Non-interactive verifiable computing: outsourcing computation to untrusted workers. In *CRYPTO 2010*, pages 465–482. Springer, 2010.

[14] P. Golle and I. Mironov. Uncheatable distributed computations. In *CT-RSA 2001*, pages 425–440. Springer, 2001.

[15] M. Green, S. Hohenberger, and B. Waters. Outsourcing the decryption of abe ciphertexts. In *Proceedings of the 20th USENIX conference on Security*, page 34. Springer, 2011.

[16] D. Hankerson, S. Vanstone, and A. Menezes. *Guide to elliptic curve cryptography*. Springer Science and Business Media, NEW YORK, 2004.

[17] S. Hohenberger and A. Lysyanskaya. How to securely outsource cryptographic computations. In *TCC 2005*, pages 264–282. Springer, 2005.

[18] A. Juels and B. Kaliski. Proofs of retrievability for large files. In *ACM CCS 2007*, pages 584–597. ACM, 2007.

[19] A. Kate, G. Zaverucha, and I. Goldberg. Constant-size commitments to polynomials and their applications. In *ASIACRYPT 2010*, pages 177–194. Springer, 2010.

[20] J. Lai, R. Deng, C. Guan, and J. Weng. Attribute-based encryption with verifiable outsourced decryption. *IEEE Transactions on Information Forensics and Security*, 8(8):1343–1354, 2013.

[21] J. Li, J. Li, X. Chen, and C. Jia. Identity-based encryption with outsourced revocation in cloud computing. *IEEE Transactions on Computers*, 64(2):425–437, 2015.

[22] X. Ma, J. Li, and F. Zhang. Outsourcing computation of modular exponentiations in cloud computing. *Cluster Computing*, 16:787–796, 2013.

[23] P. Nguyen, I. Shparlinski, and J. Stern. Distribution of modular sums and the security of server aided exponentiation. In *Proceedings of the Workshop on Cryptography and Computational Number Theory*, pages 1–16. Springer, 1999.

[24] A. Sahai and B. Waters. Fuzzy identity-based encryption. In *EUROCRYPT 2005*, pages 457–473. Springer, 2005.

[25] C. Wang, S. Chow, Q. Wang, K. Ren, and W. Lou. Privacy-preserving public auditing for secure cloud storage. *IEEE Transactions on Computers*, 62(2):362–375, 2013.

[26] Y. Wang, Q. Wu, D. Wong, and B. Qin. Securely outsourcing exponentiations with single untrusted program for cloud storage. In *ESORICS 2014*, pages 326–343. Springer, 2014.

[27] J. Yuan and S. Yu. Proofs of retrievability with public verifiability and constant communication cost in cloud. In *Proceeding of International Workshop on Security in Cloud Computing*, pages 19–26. ACM, 2013.

[28] L. Zhang and R. Naini. Private outsourcing of polynomial evaluation and matrix multiplication using multilinear maps. In *CANS 2013*, pages 329–348. Springer, 2013.