Verifiable Outsourcing Computation of Modular Exponentiations with Single Server

Jianxing Cai, Yanli Ren, and Chunshui Huang
(Corresponding author: Yanli Ren)

School of Communication and Information Engineering, Shanghai University
Shanghai 200444, China
(Email: renyanli@shu.edu.cn)

(Received Dec. 5, 2015; revised and accepted Jan. 23 & Feb. 5, 2016)

Abstract

Verifiable computation (VC) allows a computationally weak client to outsource evaluation of a function on many inputs to a powerful but untrusted server. In this paper, we propose an algorithm of verifiable outsourcing computation with single server on modular exponentiation, which has wide applications in public key cryptosystems. We also extend the algorithm to verifiable outsourcing of simultaneous modular exponentiations. The proposed two algorithms improve checkability based on one server compare with the previous ones, where the outsourcer can detect the failure with probability close to 1 if the server misbehaves. The experiments show that our algorithms are the implementations of secure and verifiable outsourcing for single modular exponentiation and simultaneous modular exponentiations.

Keywords: Checkability, modular exponentiation, single server, verifiable outsourcing computation

1 Introduction

Outsourcing computation allows a computation-limited client to outsource the operations on the private data to a powerful server [5]. The rise of cloud computing in recent years has made outsourcing of storage and computation reality [20, 21]. Outsourcing computation will provide an ideal way of freeing up the resources of the client as one of the advantages of cloud computing model [18, 22]. By outsourcing work load to the cloud server, cloud users can use unlimited resources provided by cloud to complete the high cost of computing.

Despite of the tremendous benefits, outsourcing computation also brings an unprecedented security challenge [4, 12]. First of all, due to the opaque of cloud server's internal operational details, there are various malicious motives. It makes the server cannot be completely trusted. For instance, cloud server may be Jerry-rigged in a computation task in order to save resources if clients cannot judge the correctness of the output. Secondly, there may be malicious attacks from external or internal of server such as some software bugs. All of these can make the server return computationally invalid results [5]. Therefore private security and content security of the cloud environment are important problems in our research.

In general, an effective secure outsourcing scheme should have the following properties:

1) The client's data is private for the server;

2) The client can verify the correctness of the output returned by the server.

3) The client can carry out the computation correctly using substantially less effort than computing the result on its own [5, 14, 17].

Dijk et al. [7] proposed outsourcing algorithms of variable-exponent fixed-base and fixed-exponent variable-base exponentiations in a model with one untrusted server. In these algorithms, no variable parts are public before sending to the only untrusted server. They also considered an algorithm of outsourcing variable-exponent variable-base exponentiations. However, in this algorithm, the outsourced base is known to the server. Ma, Li and Zhang [13] described securely outsourcing algorithms of two types of exponentiations in two non-collusion untrusted servers. Hohenberger and Lysyanskaya [11] presented outsource-secure algorithms of variable-exponent variable-base exponentiations in one-malicious model of two untrusted servers. Chen et al. [5] also proposed outsourcing algorithm of modular exponentiation based on two servers, and improved the checkability and efficiency for the outsourcer. Wang et al. [19] constructed an efficient algorithm for batch modular exponentiation based on an untrusted server, but the outsourcer need to execute one modular exponentiation itself when verifying the outsourced result and the checkability is only $1/(n+1)$, where n is the number of modular exponentiations.

In this paper, we first propose a secure verifiable outsourcing algorithm of single modular exponentiation with single server. We also present another outsourcing algorithm for simultaneous modular exponentiations. In the proposed algorithms, the outsourcer could detect any failure with probability close to 1 if the server returns the fault result. The experiments show that the proposed algorithms improve checkability without decreasing efficiency for the outsourcer compare with the previous ones.

The organization of this paper is described as follows: in Section 2, the definitions and security requirements of our algorithms are given. A verify outsourcing algorithm of modular exponentiation is proposed in Section 3. In the following section, we present another outsourcing algorithm for simultaneous modular exponentiations by using single server. The performance evaluation of the proposed algorithms is given in Section 5. We conclude the paper in Section 6.

2 Definitions and Security Requirements

In this section, we review definitions and security requirements of outsource-secure algorithms which have been used in [5, 11, 19].

An algorithm **Alg** includes a trusted part T and an untrusted program U which is invoked by T. We use T^{U} to denote the work that executed by T and U. An adversary A is simulated by a pair of algorithms A=(E,U'), where E denotes the adversarial environment and generates adversarial inputs for **Alg**, and U' represents an adversarial software written by E. The security model is shown in Figure 1.

Definition 1. (Algorithm with outsource-I/O) *An outsourcing algorithm **Alg** takes five inputs and generates three outputs. The first three inputs are generated by an honesty party, and the last two inputs are produced by an adversary environment E. The first input is called the honest, secret input, which is private for both E and U'; the second input is honest and protected, which may be known by E, but is private for U'; the third one is called honest and unprotected input, which is public for both E and U. The last two inputs are maliciously chosen by E, and thus they are known for E. One of them is adversarial and protected, which is protected from U'; the other one is public for E, and we call it the adversarial, unprotected input. Similarly, the first output is secret and unknown for both E and U'; the second one is protected, which means it may be public for E, but protected from U'; the last output is unprotected, which are public for both E and U'.*

As presented in [11], we assume that the two adversaries E, U' can only make direct communication before the execution of T^{U}, and in other cases, they are not be able to communicate directly and must pass message through the outsourcer T. The reason is explained as follows. In the real world, we assume E is a malicious manufacture, and U' is a malicious software produced by E. It is obvious that U' is controlled by E and they can exchange the messages directly before it is sold to T. However, E cannot send instructions to U' directly once T begins to invoke U' as the outsourcing program. During the execution of T^{U}, E can only establish an indirect communication channel with U' through the unprotected inputs and outputs of **Alg**, which means that all messages they communicate with each other must pass through T.

The following outsource-security definition ensures that both E and U' cannot learn nothing about the private 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 as defined above. A pair of algorithms (T, U) is called to be outsource-security if the following conditions holds.*

1) Correctness: T^{U} is a correct execution of **Alg**.
2) Security: For any probabilistic polynomial-time (PPT) adversary A=(E,U'), there exist two PPT simulators $(\mathrm{S}_1, \mathrm{S}_2)$ such that the results of real and ideal experiment are computationally indistinguishable.

Pair One: $EView_{\mathrm{real}} \sim EView_{\mathrm{ideal}}$, which means that the malicious environment E cannot learn anything interesting about the secret inputs and outputs during the execution of T^{U}. Both of the real process and the ideal process proceed in rounds. The notation "$\leftarrow$" denotes the outputs of the procedure in the right hand side.

- The i-th round of real process consists of the following steps. The three honest inputs $(\mathrm{x_{hs}}, \mathrm{x_{hp}}, \mathrm{x_{hu}})$ are chosen by an honest stateful process I where the environment E cannot access. Then E chooses $\mathrm{estate^i}$, $\mathrm{x^i_{ap}}$, $\mathrm{x^i_{au}}$, $stop^{\mathrm{i}}$ based on its view from the last round and honest inputs $(\mathrm{x^i_{hs}}, \mathrm{x^i_{hp}}, \mathrm{x^i_{hu}})$ given to T^{U}, where $estate^{\mathrm{i}}$ is a random number as reminding what it did next time it is invoked, $\mathrm{x^i_{ap}}$, $\mathrm{x^i_{au}}$ are two adversarial inputs, and $stop^{\mathrm{i}}$ is the Boolean variable which denotes whether round i is the last round. Next, the algorithm $\mathrm{T}^{\mathrm{U}'}$ is implemented on the inputs $(\mathrm{tstate^{i-1}}, \mathrm{x^{j^i}_{hs}}, \mathrm{x^{j^i}_{hp}}, \mathrm{x^{j^i}_{hu}}, \mathrm{x^i_{ap}}, \mathrm{x^i_{au}})$, and generates a new state $tstate^{\mathrm{i}}$ for T, and secret, protected, unprotected outputs $\mathrm{y^i_s}$, $\mathrm{y^i_p}$, $\mathrm{y^i_u}$, where $\mathrm{tstate^{i-1}}$ is T's previously saved state. The oracle U' saves its current state $ustate^{\mathrm{i}}$ based on its previously saved state $ustate^{\mathrm{i-1}}$.
  - $(\mathrm{istate^i}, \mathrm{x^i_{hs}}, \mathrm{x^i_{hp}}, \mathrm{x^i_{hu}}) \leftarrow \mathrm{I}(1^{\mathrm{k}}, \mathrm{istate^{i-1}})$;
  - $(\mathrm{estate^i}, \mathrm{j^i}, \mathrm{x^i_{ap}}, \mathrm{x^i_{au}}, \mathrm{stop^i}) \leftarrow E(1^k, EView^{i-1}_{real}, x^i_{hp}, x^i_{hu})$;
  - $(\mathrm{tstate^i}, \mathrm{ustate^i}, \mathrm{y^i_s}, \mathrm{y^i_p}, \mathrm{y^i_u}) \leftarrow T^{U'ustate^{i-1}} (tstate^{i-1}, x^{j^i}_{hs}, x^{j^i}_{hp}, x^{j^i}_{hu}, x^i_{ap}, x^i_{au})$.

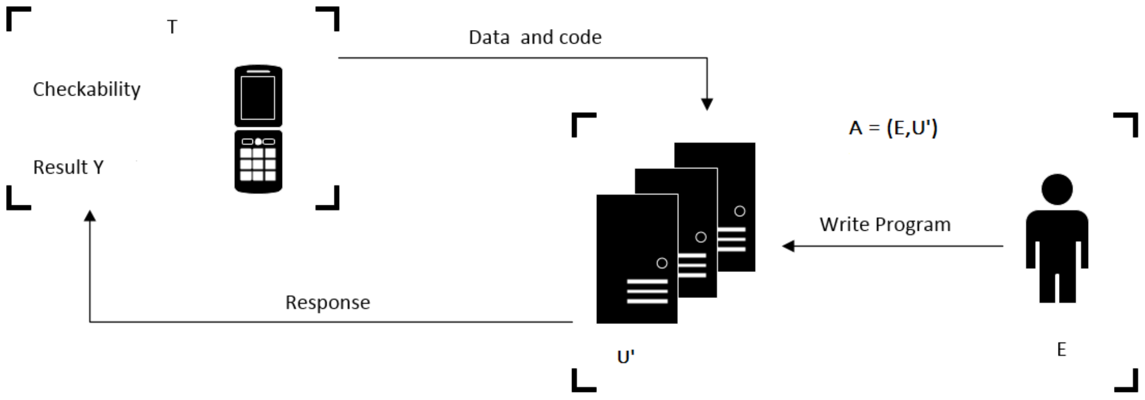

Figure 1: Security model

Thus, the view that the adversarial environment E gets in the i-th round of the real process is $\mathrm{EView}^{\mathrm{i}}_{\mathrm{real}} = (\mathrm{estate}^{\mathrm{i}}, \mathrm{y}^{\mathrm{i}}_{\mathrm{p}}, \mathrm{y}^{\mathrm{i}}_{\mathrm{u}})$ and the overall view is the view in the last round, i.e., $\mathrm{EView}_{\mathrm{real}} = \mathrm{EView}^{\mathrm{i}}_{\mathrm{real}}$ if $\mathrm{stop}^{\mathrm{i}} = \mathrm{TRUE}$.

- The i-th round of ideal process consists of the following steps. In i-th round, the stateful simulator S_1 knows nothing about the secret input $\mathrm{x}^{\mathrm{i}}_{\mathrm{hs}}$, but is given the protected and unprotected outputs that generated by **Alg**. Finally, S_1 outputs some values, either $(\mathrm{y}^{\mathrm{i}}_{\mathrm{p}}, \mathrm{y}^{\mathrm{i}}_{\mathrm{u}})$ or $(\mathrm{Y}^{\mathrm{i}}_{\mathrm{p}}, \mathrm{Y}^{\mathrm{i}}_{\mathrm{u}})$ captured by using a Boolean indicator $\mathrm{ind}^{\mathrm{i}}$. In the whole process, S_1 is allowed to access oracle U' and U' saves its state as in the real experiment.

  – $(\mathrm{istate}^{\mathrm{i}}, \mathrm{x}^{\mathrm{i}}_{\mathrm{hs}}, \mathrm{x}^{\mathrm{i}}_{\mathrm{hp}}, \mathrm{x}^{\mathrm{i}}_{\mathrm{hu}}) \leftarrow \mathrm{I}(1^{\mathrm{k}}, \mathrm{istate}^{\mathrm{i}-1})$.

  – $(\mathrm{estate}^{\mathrm{i}}, \mathrm{j}^{\mathrm{i}}, \mathrm{x}^{\mathrm{i}}_{\mathrm{ap}}, \mathrm{x}^{\mathrm{i}}_{\mathrm{au}}, \mathrm{stop}^{\mathrm{i}}) \leftarrow \mathrm{E}(1^{\mathrm{k}}, \mathrm{EView}^{i-1}_{\mathrm{ideal}}, \mathrm{x}^{\mathrm{i}}_{\mathrm{hp}}, \mathrm{x}^{\mathrm{i}}_{\mathrm{hu}})$.

  – $(\mathrm{astate}^{\mathrm{i}}, \mathrm{y}^{\mathrm{i}}_{\mathrm{s}}, \mathrm{y}^{\mathrm{i}}_{\mathrm{p}}, \mathrm{y}^{\mathrm{i}}_{\mathrm{u}}) \leftarrow \mathrm{Alg}(astate^{i-1}, x^{j^{\mathrm{i}}}_{\mathrm{hs}}, x^{j^{\mathrm{i}}}_{\mathrm{hp}}, x^{j^{\mathrm{i}}}_{\mathrm{hu}}, \mathrm{x}^{\mathrm{i}}_{\mathrm{ap}}, \mathrm{x}^{\mathrm{i}}_{\mathrm{au}})$.

  – $(\mathrm{sstate}^{\mathrm{i}}, \mathrm{ustate}^{\mathrm{i}}, \mathrm{Y}^{\mathrm{i}}_{\mathrm{p}}, \mathrm{Y}^{\mathrm{i}}_{\mathrm{u}}, \mathrm{ind}^{\mathrm{i}}) \leftarrow \mathrm{S}_1^{\mathrm{U}'(ustate^{\mathrm{i}-1})}(sstate^{\mathrm{i}-1}, \mathrm{x}^{\mathrm{j}^{\mathrm{i}}}_{\mathrm{hp}}, x^{j^{\mathrm{i}}}_{\mathrm{hu}}, \mathrm{x}^{\mathrm{i}}_{\mathrm{ap}}, \mathrm{x}^{\mathrm{i}}_{\mathrm{au}}, y^{i}_{p}, y^{i}_{u})$.

  – $(\mathrm{z}^{\mathrm{i}}_{\mathrm{p}}, \mathrm{z}^{\mathrm{i}}_{\mathrm{u}}) = \mathrm{ind}^{\mathrm{i}}(\mathrm{Y}^{\mathrm{i}}_{\mathrm{p}}, \mathrm{Y}^{\mathrm{i}}_{\mathrm{u}}) + (1 - \mathrm{ind}^{\mathrm{i}})(\mathrm{y}^{\mathrm{i}}_{\mathrm{p}}, \mathrm{y}^{\mathrm{i}}_{\mathrm{u}})$.

  Thus, the view that E obtains in the i-th round of the real process is $\mathrm{EView}^{\mathrm{i}}_{\mathrm{ideal}} = (\mathrm{estate}^{\mathrm{i}}, \mathrm{z}^{\mathrm{i}}_{\mathrm{p}}, \mathrm{z}^{\mathrm{i}}_{\mathrm{u}})$ and the overall view is the view in the last round, i.e., $\mathrm{EView}_{\mathrm{ideal}} = \mathrm{EView}^{\mathrm{i}}_{\mathrm{ideal}}$ if $\mathrm{stop}^{\mathrm{i}} = \mathrm{TRUE}$.

Pair Two: $\mathrm{UView}_{\mathrm{real}} \sim \mathrm{UView}_{\mathrm{ideal}}$, which means that the untrusted software U' written by E cannot get anything interesting about the inputs and outputs during the execution of T^{U}.

- On the basis of the real process described in Pair One, the view of the untrusted software U' in the real process is $\mathrm{UView}_{\mathrm{real}} = \mathrm{ustate}^{\mathrm{i}}$ if $\mathrm{stop}^{\mathrm{i}} = \mathrm{TRUE}$.

- The i-th round of ideal process consists of the following steps. In i-th round, the stateful simulator S_2 is given the unprotected outputs that generated by **Alg**.

  – $(\mathrm{istate}^{\mathrm{i}}, \mathrm{x}^{\mathrm{i}}_{\mathrm{hs}}, \mathrm{x}^{\mathrm{i}}_{\mathrm{hp}}, \mathrm{x}^{\mathrm{i}}_{\mathrm{hu}}) \leftarrow \mathrm{I}(1^{\mathrm{k}}, \mathrm{istate}^{\mathrm{i}-1})$.

  – $(\mathrm{estate}^{\mathrm{i}}, \mathrm{j}^{\mathrm{i}}, \mathrm{x}^{\mathrm{i}}_{\mathrm{ap}}, \mathrm{x}^{\mathrm{i}}_{\mathrm{au}}, \mathrm{stop}^{\mathrm{i}}) \leftarrow \mathrm{E}(1^{\mathrm{k}}, \mathrm{estate}^{\mathrm{i}-1}, \mathrm{x}^{\mathrm{i}}_{\mathrm{hp}}, \mathrm{x}^{\mathrm{i}}_{\mathrm{hu}}, \mathrm{y}^{\mathrm{i}-1}_{\mathrm{p}}, y^{i-1}_{u})$.

  – $(\mathrm{astate}^{\mathrm{i}}, \mathrm{y}^{\mathrm{i}}_{\mathrm{s}}, \mathrm{y}^{\mathrm{i}}_{\mathrm{p}}, \mathrm{y}^{\mathrm{i}}_{\mathrm{u}}) \leftarrow \mathrm{Alg}(astate^{i-1}, x^{j^{\mathrm{i}}}_{\mathrm{hs}}, x^{j^{\mathrm{i}}}_{\mathrm{hp}}, x^{j^{\mathrm{i}}}_{\mathrm{hu}}, \mathrm{x}^{\mathrm{i}}_{\mathrm{ap}}, \mathrm{x}^{\mathrm{i}}_{\mathrm{au}})$.

  – $(\mathrm{sstate}^{\mathrm{i}}, \mathrm{ustate}^{\mathrm{i}}) \leftarrow \mathrm{S}_2^{\mathrm{U}'(ustate^{\mathrm{i}-1})}(sstate^{\mathrm{i}-1}, x^{j^{\mathrm{i}}}_{\mathrm{hu}}, \mathrm{x}^{\mathrm{i}}_{\mathrm{au}})$.

  Thus, the view of the untrusted software U' in i-th round of the ideal process is $\mathrm{UView}^{\mathrm{i}}_{\mathrm{ideal}} = (\mathrm{ustate}^{\mathrm{i}})$ and the overall view is the view in the last round, which means $\mathrm{UView}_{\mathrm{ideal}} = \mathrm{UView}^{\mathrm{i}}_{\mathrm{ideal}}$ if $\mathrm{stop}^{\mathrm{i}} = \mathrm{TRUE}$.

Finally, we give the following definition if T^{U} is a correct implementation of **Alg**.

Definition 3. (α-efficient, secure outsourcing) *A pair of algorithms (T, U) is called α-efficient if the running time of T is less than an α-multiplicative factor of that of **Alg** for any inputs x.*

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

Definition 5. ((α, β)-outsource-security) *A pair of algorithms (T, U) is said to be an (α, β)-outsource-secure execution of **Alg** if it is both α-efficient and β-checkable.*

3 Outsourcing Algorithm of Modular Exponentiation with Single Server

Similar to [5], a subroutine named Rand is invoked in our algorithm. The output is a random pair of the form $(\mathrm{b}, \mathrm{g}^{\mathrm{b}} \bmod \mathrm{p})$, and the input is a prime p and a base $\mathrm{g} \in \mathrm{Z}_{\mathrm{p}}^{*}$, where $\mathrm{b} \in \mathrm{Z}_{\mathrm{q}}$. To implement this subroutine, we can use a trusted server to generate a table of random for T and T retrieves a new pair in the table when an invocation is needed. We call this table-lookup method. Another method is generating those random pairs by using EBPV generator [15, 19].

The inputs of single server exponentiation (SgExp) include a base u and a power a, where $\mathrm{a} \in \mathrm{Z}_{\mathrm{q}}^{*}$ and $\mathrm{u} \in \mathrm{Z}_{\mathrm{p}}^{*}$, and $\mathrm{u}^{\mathrm{q}} = 1 \bmod \mathrm{p}$. Both a and u are secret for the server U. The output of SgExp is $\mathrm{u}^{\mathrm{a}} \bmod \mathrm{p}$ where p and q are two large primes and $\mathrm{q}|\mathrm{p}-1$.

3.1 Outsourcing Algorithm

Here we propose our algorithm SgExp for secure outsourcing of exponentiations. One important security requirement for SgExp is that an adversary can get any useful information about the inputs and outputs of SgExp. Similar to [5] and [11], $\mathrm{U}(\mathrm{x}, \mathrm{y}) \rightarrow \mathrm{y}^{\mathrm{x}}$ express that we invoke the server to compute one time. And (x, y) are the inputs while $\mathrm{y}^{\mathrm{x}} \bmod \mathrm{p}$ are the outputs.

1) First, T invokes the subroutine Rand four times to create four blinding pairs $(\alpha, \mathrm{g}^{\alpha})$, $(\beta, \mathrm{g}^{\beta})$, $(\varepsilon, \mathrm{g}^{\varepsilon})$, $(\theta, \mathrm{g}^{\theta})$. We denote $\mathrm{A} = \mathrm{g}^{\alpha} \bmod \mathrm{p}$, $\mathrm{B} = \mathrm{g}^{\beta} \bmod \mathrm{p}$, $\mathrm{C} = \mathrm{g}^{\varepsilon} \bmod \mathrm{p}$, $\mathrm{D} = \mathrm{g}^{\theta} \bmod \mathrm{p}$.

2) The first logical divisions are

$$\mathrm{u}^{\mathrm{a}} = (\mathrm{Aw})^{\mathrm{a}} = \mathrm{g}^{\mathrm{a}}\alpha \mathrm{w}^{\mathrm{a}} = \mathrm{g}^{\beta}\mathrm{g}^{\gamma}\mathrm{w}^{\mathrm{a}} \bmod \mathrm{p},$$

where $\mathrm{w} = \mathrm{u}/\mathrm{A} \bmod \mathrm{p}$ and $\gamma = (\mathrm{a}\alpha - \beta) \bmod \mathrm{q}$.

In order to verify the correct of the consequence, we do the second divisions by using another two blinding pairs $(\varepsilon, \mathrm{g}^{\varepsilon})$ and $(\theta, \mathrm{g}^{\theta})$ as follows:

$$\mathrm{u}^{\mathrm{a}} = (\mathrm{Cv})^{\mathrm{a}} = \mathrm{g}^{\mathrm{a}\varepsilon}\mathrm{v}^{\mathrm{a}} = \mathrm{g}^{\theta}\mathrm{g}^{\tau}\mathrm{v}^{\mathrm{a}} \bmod \mathrm{p},$$

where $\mathrm{v} = \mathrm{u}/\mathrm{C} \bmod \mathrm{p}$ and $\tau = (\mathrm{a}\varepsilon - \theta) \bmod \mathrm{q}$.

3) T randomly selects i, j and $\mathrm{j} \neq \mathrm{i}$. Let $\mathrm{a}_1 = \mathrm{a} - 2^{\mathrm{i}}$, $a_2 = a - 2^j$, $2^i < a$, $2^j < a$.

4) T runs Rand to obtain eight pairs $(\mathrm{t}_1, \mathrm{g}^{\mathrm{t}_1})$, $(\mathrm{t}_2, \mathrm{g}^{\mathrm{t}_2})$, $(\mathrm{s}_1, \mathrm{g}^{\mathrm{s}_1})$, $(\mathrm{s}_2, \mathrm{g}^{\mathrm{s}_2})$, $(\mathrm{s}_3, \mathrm{g}^{\mathrm{s}_3})$, $(\mathrm{s}_4, \mathrm{g}^{\mathrm{s}_4})$, $(\mathrm{s}_5, \mathrm{g}^{\mathrm{s}_5})$, $(\mathrm{s}_6, \mathrm{g}^{\mathrm{s}_6})$. T then randomly chooses $\mathrm{m}_1, \cdots, \mathrm{m}_{\mathrm{i}-1}, \mathrm{m}_{\mathrm{i}+1}, \cdots, \mathrm{m}_{\mathrm{j}-1}, \mathrm{m}_{\mathrm{j}+1}, \cdots, \mathrm{m}_{\mathrm{n}} \in \mathrm{Z}_{\mathrm{p}}^{*}$.

5) Next, T queries U in random order as:

$$\begin{aligned}
U(\frac{\gamma}{t_1}, g^{t_1}) &\rightarrow g^{\gamma}, \\
U(a_1, wg^{s_1}) &\rightarrow R_{11} = w^{a_1}g^{s_1 a_1}, \\
U(\frac{s_1 a_1 - s_2}{s_3}, g^{s_3}) &\rightarrow R_{12} = g^{s_1 a_1 - s_2}, \\
U(\frac{\tau}{t_2}, g^{t_2}) &\rightarrow g^{\tau}, \\
U(a_2, vg^{s_4}) &\rightarrow R_{21} = v^{a_2}g^{s_4 a_2}, \\
U(\frac{s_4 a_2 - s_5}{s_6}, g^{s_6}) &\rightarrow R_{22} = g^{s_4 a_2 - s_5}, \\
U(2, m_1) &\rightarrow m[1] = m_1^2, \\
U(4, m_1) &\rightarrow m[2] = m_2^4, \\
&\cdots \\
U(2^i, w) &\rightarrow m[i] = w^{2^i}, \\
&\cdots \\
U(2^j, v^{-1}) &\rightarrow m[j] = v^{-2^j}, \\
&\cdots \\
U(2^n, m_n) &\rightarrow m[n] = m_n^{2^n}.
\end{aligned}$$

Note 1. *Considering both the client and the server computing devices are stored in binary, so we don't think the binary conversion cost computing time.*

6) T computes:

$$\begin{aligned}
w^{a_1} &= R_{11}(R_{12}g^{s_2})^{-1}, \\
v^{a_2} &= R_{21}(R_{22}g^{s_5})^{-1},
\end{aligned}$$

and then checks that whether U produces the correct outputs, i.e.,

$$\mathrm{Bg}^{\gamma}\mathrm{w}^{\mathrm{a}_1}\mathrm{m}[\mathrm{i}]\mathrm{m}[\mathrm{j}] \bmod \mathrm{p} = \mathrm{Dg}^{\tau}\mathrm{v}^{\mathrm{a}_2} \bmod \mathrm{p}. \quad (1)$$

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

$$\mathrm{u}^{\mathrm{a}} = \mathrm{Bg}^{\gamma}\mathrm{w}^{\mathrm{a}_1}\mathrm{m}[\mathrm{i}] \bmod \mathrm{p}.$$

3.2 Security Analysis

Lemma 1. (Correctness): *In a single untrusted model, the algorithm (T, U') presented in Section 3.1 is a correct implementation of SgExp.*

Proof. As described in Section 3.1, we know that $\mathrm{R}_{11} = \mathrm{w}^{\mathrm{a}_1}\mathrm{g}^{\mathrm{s}_1\mathrm{a}_1}, \mathrm{R}_{12} = \mathrm{g}^{\mathrm{s}_1\mathrm{a}_1 - \mathrm{s}_2}$. So, $\mathrm{R}_{11}(\mathrm{R}_1 2\mathrm{g}^{\mathrm{s}_2})^{-1} = \mathrm{w}^{\mathrm{a}_1}$. In addition, $\mathrm{w}^{\mathrm{a}} = \mathrm{w}^{\mathrm{a}_1 + 2^{\mathrm{i}}} = \mathrm{w}^{\mathrm{a}_1}\mathrm{w}^{2^{\mathrm{i}}} = \mathrm{w}^{\mathrm{a}_1}\mathrm{m}[\mathrm{i}]$. Therefore,

$$\begin{aligned}
\mathrm{u}^{\mathrm{a}} &= g^{\beta}g^{\gamma}w^{a} \bmod p \\
&= g^{\beta}g^{\gamma}w^{a_1}m[i].
\end{aligned}$$

Similarly, T compute:

$$\begin{aligned}
v^{a_2} &= R_{21}(R_{22}g^{s_5})^{-1}, \\
u^{a} &= g^{\theta}g^{\tau}v^{z} \bmod p \\
&= g^{\theta}g^{\tau}v^{a_2}m[j]^{-1}.
\end{aligned}$$

□

Theorem 1. (Security): *In single untrusted program model, the algorithms (T, U) are an outsource-secure implementation of SgExp, where the inputs (a, u) may be honest, secret; honest, protected; or adversarial, protected.*

Proof. Firstly, we show Pair One $\mathrm{EView_{real}} \sim \mathrm{EView_{ideal}}$ holds, which means the adversarial environment E leans nothing during the execution of (T, U'). □

If the input (a, u) is honest, protected or adversarial, protected, the simulator $\mathrm{S_1}$ behaves same as in the real execution. Thus, it needs only to consider the case where (a, u) is an honest, secret input.

So, suppose (a, u) is an honest, secret input. The simulator $\mathrm{S_1}$ in the ideal experiment behaves as follows. When receiving the input in the i-th round, $\mathrm{S_1}$ ignores it and instead submits 6 random queries with the form $(\mathrm{a_j}, \mathrm{u_j})$ and n random queries with the form $(2^{\mathrm{k}}, \mathrm{u'_j})(\mathrm{k} \in 1, 2, \cdots, \mathrm{n})$ to U'. Then $\mathrm{S_1}$ tests 6 outputs $(\mathrm{u_j^{a_j}})$ and 2 random outputs $(\mathrm{u'}_{\mathrm{j}}^{2^{\mathrm{k}}})$ from U'. If an error is detected, $\mathrm{S_1}$ outputs $\mathrm{Y_p^i}$ = "error", $\mathrm{Y_u^i}$ = "ϕ", ind^i=1. If no error is checked, $\mathrm{S_1}$ verifies the remaining outputs. If all checks pass, $\mathrm{S_1}$ outputs $\mathrm{Y_p^i}$"ϕ", $\mathrm{Y_u^i}$ = "ϕ", $ind^i = 0$; else, $\mathrm{S_1}$ chooses a random element $\mathrm{r} \in \mathrm{Z_p^*}$, and outputs $\mathrm{Y_p^i}$ = "r", $\mathrm{Y_u^i}$ = "ϕ", $ind^i = 1$. In either condition, $\mathrm{S_1}$ saves its own states and those of U'.

As same as [11], we need to show that the input distribution to U' in the real experiment is computationally distinguished from that in the ideal one. In the ideal experiment, the inputs are all chosen randomly and uniformly distributed. While in the real experiment, each part of all queries that T makes to U' is generated by invoking the subroutine Rand and thus computationally indistinguishable from random. Thus, we have three possible cases to consider. If U' behaves honest in the i-th round, $\mathrm{EView_{real}^i} \sim \mathrm{EView_{ideal}^i}$, because the outputs of SgExp in the ideal experiment are not replaced. If U' misbehaves in the i-th round, it will be caught by both T and $\mathrm{S_1}$ with probability $1 - 1/n^2$, and then the experiment outputs "error"; otherwise, the outputs of SgExp are corrupted by U'. In the real experiment, the outputs generated by U' is multiplied together with random values produced by invoking Rand, thus the corrupted outputs of SgExp look random to E. While in the ideal experiment, the outputs of U' are replaced by a random value r. So, we conclude that $\mathrm{EView_{real}^i} \sim \mathrm{EView_{ideal}^i}$ also holds even if U' is dishonest in the i-th round. In all, by the hybrid argument we have $\mathrm{EView_{real}} \sim \mathrm{EView_{ideal}}$.

Secondly, we show Pair Two $\mathrm{UView_{real}} \sim \mathrm{UView_{ideal}}$ holds, that is to say, the adversarial software U' leans nothing during the execution of (T, U').

In the ideal experiment, the simulator $\mathrm{S_2}$ behaves as follows. On receiving the inputs in the i-th round, $\mathrm{S_2}$ ignore them but submits 6 random queries of the form $(\mathrm{a_j}, \mathrm{u_j})$ and n random queries of the form $(2^{\mathrm{k}}, \mathrm{u'_j})$ $(k \in \{1, 2, \cdots, n\})$ to U', and then $\mathrm{S_2}$ saves its own states and that of U'. Since all three kinds of inputs are unknown to U', the simulator $\mathrm{S_2}$ is applicable to all those cases. 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' since they cannot communicate each other during the execution of $\mathrm{T^U}$. In addition, the input distribution to U' in the real experiment is computationally indistinguishable from that in the ideal experiment randomly chosen by $\mathrm{S_2}$. Thus, $\mathrm{UView_{real}^i} \sim \mathrm{UView_{ideal}^i}$ holds for each round i. In all, we conclude that $\mathrm{UView_{real}} \sim \mathrm{UView_{ideal}}$.

Theorem 2. *In one untrusted model, the algorithm (T, U) presented in Section 3.1 is an* $(\mathrm{O}((\log^2 \mathrm{n})/\mathrm{n}),\ 1 - 1/n^2)$*-outsource-secure implementation of SgExp, where m is the bit length of a.*

Proof. In order to compute $\mathrm{u^a} \bmod \mathrm{p}$, SgExp requires 12 calls to Rand, 22 modular multiplication and 8 modular inverse. It takes $O(1)$ or $O(\log^2 n)$ modular multiplication by using table-lookup method or the EBPV generator, where n is the bit of q. As we know, it takes roughly $1.5n$ modular multiplication to compute $\mathrm{u^a} \bmod \mathrm{p}$ by the square-and-multiply method. Therefore, the algorithm (T, U) is an $O((\log^2 n)/n)$-efficient execution of SgExp. □

On the other side, U cannot cheat the outsourcer to accept an error result unless that he knows i and j. Since i and j are randomly chosen from $1, 2, \cdots, \mathrm{n}$, the outsourcer can verify an outsourcing result with probability $1 - 1/n^2$.

3.3 Comparison

We make a comparison between our scheme SgExp and the schemes in [5, 19] in Table 1, where n is the bit length of q.

Note that the scheme [19] needs one modular exponentiation operation and has no advantage for single modular exponentiation. The proposed scheme is superior in checkability, and it is more efficient than that of the scheme in [19]. Our scheme only uses one server but the scheme in [5] is based on two untrusted server though we need more modular multiplications and modular inversions for the outsourcer and more queries to U. Therefore, the proposed scheme improves the checkability based on only one server.

4 Outsourcing Algorithm of Simultaneous Modular Exponentiations

We also extend our scheme to an outsourcing algorithm of simultaneous modular exponentiations (SmExp) $\mathrm{u_1^a u_2^b} \bmod \mathrm{p}$ which has important applications in many cryptographic schemes such as trapdoor commitment [3, 8, 9, 16] and chameleon hashing [1, 2, 6, 10].

4.1 Outsourcing Algorithm

There are two bases $\mathrm{u_1}, \mathrm{u_2} \in \mathrm{Z_p^*}$ and two powers $\mathrm{a}, \mathrm{b} \in \mathrm{Z_q^*}$. We need to compute $\mathrm{u_1^a u_2^b} \bmod \mathrm{p}$.

Table 1: Comparison of outsourcing single exponentiation

	GExp[8]	Exp[1]	SgExp
Rand	7	5	12
Modular Multiplications	12	7	22
Modular Exponentiation	1	0	0
Modular Inversions	4	3	9
Queries to U	4	6	n+6
Privacy	$\surd$	$\surd$	$\surd$
Checkability	1/2	2/3	$1 - 1/n^2$
The number of servers	Single server	Two servers	Single server

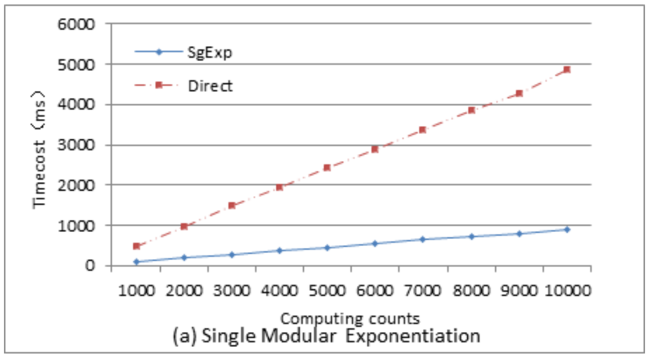

Figure 2: Simulation for SgExp algorithm

1) First, T invokes the subroutine Rand four times to create four blinding pairs (α, g^α), (β, g^β), $(\varepsilon, g^\varepsilon)$, (θ, g^θ). We denote:

$$\begin{aligned} A &= g^\alpha \bmod p, \\ B &= g^\beta \bmod p, \\ C &= g^\varepsilon \bmod p, \\ D &= g^\theta \bmod p. \end{aligned}$$

2) The first logical divisions are represented as below:

$$\begin{aligned} u_1^a u_2^b &= (Aw_1)^a (Aw_2)^b \\ &= g^\beta g^\gamma w_1^a w_2^b, \end{aligned}$$

where $\mathrm{w_1 = u_1/A, w_2 = u_2/A}$, and $\gamma = \alpha(\mathrm{b+a}) - \beta$. Similarly, T executes the second divisions as follows:

$$\begin{aligned} u_1^a u_2^b &= (Cv_1)^a (Cv_2)^b \\ &= g^\theta g^\tau v_1^a v_2^b, \end{aligned}$$

where $\mathrm{v_1 = u_1/C, v_2 = u_2/C}$, and $\tau = \varepsilon(\mathrm{a+b}) - \theta$.

3) T randomly selects k, h, i, j and $\mathrm{k < h < i < j}$. Let $\mathrm{a_1 = a - 2^k, a_2 = a - 2^h, b_1 = b - 2^i, b_2 = b - 2^j}$.

4) T runs Rand for ten times to obtain ten pairs $\mathrm{(t_1, g^{t_1})}$, $\mathrm{(t_2, g^{t_2})}$, $\mathrm{(s_1, g^{s_1})}$, $\mathrm{(s_2, g^{s_2})}$, $\mathrm{(s_3, g^{s_3})}$, $\mathrm{(s_4, g^{s_4})}$, $\mathrm{(s_5, g^{s_5})}$, $\mathrm{(s_6, g^{s_6})}$, $\mathrm{(s_7, g^{s_7})}$, $\mathrm{(s_8, g^{s_8})}$. T randomly generated n-4 integers $\mathrm{m_1, m_2, \cdots, m_{k-1}, m_{k+1}, \cdots, m_{h-1}, m_{h+1}, \cdots m_{i-1}, m_{i+1}, \cdots, m_{j-1}, m_{j+1}, \cdots, m_n \in Z_p^*}$.

5) T queries U in random order as:

$$\begin{aligned}
U(\frac{\gamma}{t_1}, g^{t_1}) &\rightarrow g^\gamma, \\
U(a_1, w_1 g^{s_1}) &\rightarrow R_{11} = w_1^{a_1} g^{s_1 a_1}, \\
U(a_1, w_1 g^{s_1}) &\rightarrow R_{11} = w_1^{a_1} g^{s_1 a_1}, \\
U(\frac{s_1 a_1 - s_2}{s_3}, g^{s_3}) &\rightarrow R_{12} = g^{s_1 a_1 - s_2}, \\
U(b_1, w_2 g^{s_1}) &\rightarrow R_{21} = w_2^{b_1} g^{s_1 b_1}, \\
U(\frac{s_1 b_1 - s_2}{s_4}, g^{s_4}) &\rightarrow R_{22} = g^{s_1 b_1 - s_2}, \\
U(\frac{\tau}{t_2}, g^{t_2}) &\rightarrow g^\tau, \\
U(a_2, v_1 g^{s_5}) &\rightarrow R_{31} = v_1^{a_2} g^{s_5 a_2}, \\
U(\frac{s_5 a_2 - s_6}{s_7}, g^{s_7}) &\rightarrow R_{32} = g^{s_5 a_2 - s_6}, \\
U(b_2, v_2 g^{s_1}) &\rightarrow R_{41} = v_2^{b_2} g^{s_1 b_2}, \\
U(\frac{s_5 a_2 - s_6}{s_8}, g^{s_8}) &\rightarrow R_{42} = g^{s_5 a_2 - s_6}, \\
U(2, m_1) &\rightarrow m[1] = m_1^2, \\
U(4, m_1) &\rightarrow m[2] = m_2^4, \\
&\cdots \\
U(2^k, w_1) &\rightarrow m[k] = w_1^{2^k}, \\
&\cdots \\
U(2^h, v_1^{-1}) &\rightarrow m[h] = v_1^{-2^h}, \\
&\cdots \\
U(2^i, w_2) &\rightarrow m[i] = w_2^{2^i}, \\
&\cdots \\
U(2^j, v_2) &\rightarrow m[j] = v_2^{2^j}, \\
&\cdots \\
U(2^n, m_n) &\rightarrow m[n] = m_n^{2^n}.
\end{aligned}$$

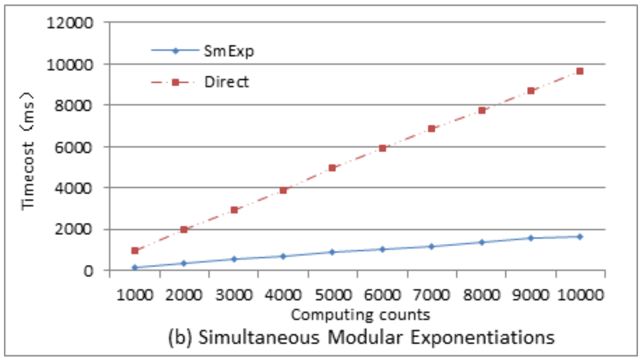

Figure 3: Simulation for SmExp algorithm

6) T computes:

$$\begin{aligned} w_1^{a_1} &= R_{11}(R_{12}g^{s_2})^{-1}, \\ w_2^{b_1} &= R_{21}(R_{22}g^{s_2})^{-1}, \\ v_1^{a_2} &= R_{31}(R_{32}g^{s_6})^{-1}, \\ v_2^{b_2} &= R_{41}(R_{42}g^{s_6})^{-1}. \end{aligned}$$

7) Finally, T checks whether U produces the correct outputs, i.e.,

$$Bg^{\gamma}w_1^{a_1}w_2^{b_1}m[k]m[i]m[h] = Dg^{\tau}v_1^{a_2}v_2^{b_2}m[j] \bmod p.$$

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

$$u_1^a u_2^b = Bg^{\gamma}w_1^{a_1}w_2^{b_1}m[k]m[i] \bmod p.$$

4.2 Security Analysis

Lemma 2. (Correctness): *In a single untrusted program model, the algorithm (T, U) presented in Section 4.1 is a correct implementation of SmExp.*

As described in Section 4.1, we know that

$$\begin{aligned} R_{11} &= w_1^{a_1}g^{s_1a_1}, \\ R_{12} &= g^{s_1a_1-s_2}, \\ R_{21} &= w_2^{b_1}g^{s_1b_1}, \\ R_{22} &= g^{s_1b_1-s_2}. \end{aligned}$$

So,

$$\begin{aligned} R_{11}(R_{12}g^{s_2})^{-1} &= \mathrm{w}_1^{a_1}, \\ R_{21}(R_{22}g^{s_2})^{-1} &= \mathrm{w}_2^{b_1}. \end{aligned}$$

In addition,

$$\begin{aligned} w_1^a &= w_1^{a_1+2^k} \\ &= w_1^{a_1}w_1^{2^k} \\ &= w_1^{a_1}m[k], \\ w_2^b &= w_2^{b_1+2^i} \\ &= w_2^{b_1}w_2^{2^i} \\ &= w_1^{b_1}m[i]. \end{aligned}$$

Therefore,

$$\begin{aligned} u_1^a u_2^b &= g^{\beta}g^{\gamma}w_1^a w_2^b \bmod p \\ &= g^{\beta}g^{\gamma}w_1^{a_1}w_2^{b_1}m[k]m[i] \bmod p. \end{aligned}$$

Similarly, we compute:

$$\begin{aligned} v_1^{a_2} &= R_{31}(R_{32}g^{s_2})^{-1}, \\ v_2^{b_2} &= R_{41}(R_{42}g^{s_2})^{-1}, \\ u_1^a u_2^b &= g^{\theta}g^{\tau}v_1^a v_2^b \bmod p \\ &= Dg^{\tau}v_1^{a_1}v_2^{b_1}m[j]m[h]^{-1} \bmod p. \end{aligned}$$

As same as Theorems 1 and 2, we can easily prove the following theorems.

Theorem 3. (Security): *In one untrusted model, the algorithm (T, U) presented in Section 4.1 is an outsource-secure implementation of SmExp, where the inputs* $(\mathrm{a}, \mathrm{b}; \mathrm{u}_1\mathrm{u}_2)$ *may be honest, secret; honest, protected; or adversarial, protected.*

Theorem 4. *In one untrusted model, the algorithm (T, U) presented in Section 4.1 is an $(O((\log^2 n)/n)$, $1 - 1/n^4)$-outsource-secure execution of SmExp, where n is the bit length of q.*

Proof. As Theorem 2, the algorithm (T, U) presented in Section 4.1 is an $O((\log^2 n)/n)$-efficient implementation of SmExp. On the other hand, U cannot cheat the outsourcer to accept an error result unless that he knows k, h, i and j. Since k, h, i and j are randomly chosen from $\{1, 2, \cdots, \mathrm{n}\}$, the outsourcer can verify an outsourcing result with probability $1 - 1/n^4$. □

4.3 Efficiency

We also make a comparison among the proposed SmExp algorithm and other outsourcing algorithms of simultaneous exponentiation. The comparison is given in Table 2.

Note that the proposed SmExp algorithm is more efficient then the GExp algorithm since no modular exponentiation is needed. For the outsourcer, the SmExp algorithm improves the checkability based on only one server though it needs more modular multiplications and modular inversions.

5 Performance Evaluation

The experimental evaluation of the proposed outsourcing algorithms will be provided in this section. Our experiment is simulated on two Windows machines with Intel(R) Core(TM) i5-3470 CPU running at 3.20 GHz and 4G memory (local user), and Intel(R) Core(TM) i7-3770 CPU running at GHz and 16G memory (cloud server), respectively. We choose C^{++} as the programming language. The Multiple Precision Integers and Rational Library (MPIR) are used in our experiments.

Table 2: Comparison of outsourcing simultaneous exponentiation

	GExp[8]	SExp[1]	SmExp
Rand	8	5	14
Modular Multiplications	17	10	38
Modular Exponentiation	1	0	0
Modular Inversions	4	4	11
Queries to U	6	8	n+10
Privacy	$\surd$	$\surd$	$\surd$
Checkability	1/3	2/3	$1-1/n^4$
Security Model	Single server	Two servers	Single server

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

$$\begin{aligned} p &= 8df2a494492276aa3d25759bb06869cbeac0d83a \\ &\quad fb8d0cf7cbb8324f0d7882e5d0762fc5b7210eaf \\ &\quad c2e9adac32ab7aac49693dfbf83724c2ec0736ee \\ &\quad 31c80291. \\ q &= c773218c737ec8ee993b4f2ded30f48edace915f. \end{aligned}$$

In Figure 2 and Figure 3, we provide the simulation of SgExp and SmExp algorithm, which means that the fault can be found with probability close to 1 if the server misbehaves. It is obvious that the time cost for the outsourcer T is much smaller than that for directly computing single modular exponentiation and simultaneously modular exponentiations since that a number of computations have been delegated to the server. Therefore, the proposed SgExp and SmExp algorithm are the implementations of secure and verifiable outsourcing for single modular exponentiation and simultaneously modular exponentiations.

In addition, we provide the evaluation time of the outsourcer for single modular exponentiation and simultaneously modular exponentiations proposed in [5, 19] and our paper. We show the result in Table 3. From Table 3, we conclude that for the outsourcer, the proposed SgExp and SmExp algorithm are more efficient than GExp proposed in [19]. The proposed algorithms improve the checkability based on one server though more time is needed than Exp and SExp of [5].

6 Conclusions

In this paper, we first propose an outsourcing algorithm for single modular exponentiation based on one server, and then extend the algorithm to secure outsourcing of simultaneous modular exponentiations. In the proposed two algorithms, the outsourcer can verify the outsourcing result efficiently and detect the error with probability close to 1. Our algorithms are superior in checkability and more efficient than that of the previous ones based on one untrusted server.

Acknowledgments

The work described in this paper was supported by the Natural Science Foundation of China (Grant No. 61373151, 61472235, 61572309, U1536108), and the Innovation Program of Shanghai Municipal Education Commission (Grant No. 14YZ020).

References

[1] G. Ateniese, B. de Medeiros, "On the key exposure problem in chameleon hashes," in *Security in Communication Networks (SCN'05)*, LNCS 3352, pp. 165–179, Springer, 2005.

[2] G. Ateniese, B. de Medeiros, "Identity-based chameleon hash and applications," in *International Conference on Financial Cryptography (FC'04)*, LNCS 3110, pp. 164–180, Springer, 2004.

[3] G. Brassard, D. Chaum, C. Cepeau, "Minimum disclosure proofs of knowledge," *Journal of Computer and System Sciences*, vol. 37, no. 2, pp. 156–189, 1988.

[4] Z. Cao, L. Liu, "A note on two schemes for secure outsourcing of linear programming," *International Journal of Network Security*, vol. 19, no. 2, pp. 323–326, 2017.

[5] X. Chen, J. Li, J. Ma, et al., "New algorithms for secure outsourcing of modular exponentiations," *IEEE Transactions on Parallel and Distributed Systems*, vol. 25, no. 9, pp. 2386–2396, 2014.

[6] X. Chen, F. Zhang, W. Susilo, et al., "Efficient generic on-line/off-line signatures without key exposure," in *Applied Cryptography and Network Security (ACNS'07)*, LNCS 4521, pp. 18–30, Springer, 2007.

[7] M. Van Dijk, D. Clarke, B. Gassend, et al., "Speeding up exponentiation using an untrusted computational resource," *Designs, Codes and Cryptography*, vol. 39, no. 2, pp. 253–273, 2006.

[8] X. Dong, "A multi-secret sharing scheme based on the CRT and RSA," *International Journal of Electronics and Information Engineering*, vol. 2, no. 1, pp. 47–51, 2015.

Table 3: Time comparison for modular exponentiation

T/ms	GExp[8]	Exp(SExp)[1]	Our scheme
Single	0.511	0.031	0.091
Simultaneous	0.536	0.056	0.169

[9] J. A. Garay, P. MacKenzie, K. Yang, "Strengthening zero-knowledge protocols using signatures," in *International Conference on the Theory and Applications of Cryptographic Techniques*, pp. 177–194, 2003.

[10] W. R. Ghanem, M. Shokir, and M. Dessoky, "Defense Against Selfish PUEA in Cognitive Radio Networks Based on Hash Message Authentication Code," *International Journal of Electronics and Information Engineering*, vol. 4, no. 1, pp. 12–21, 2016.

[11] S. Hohenberger, A. Lysyanskaya, "How to securely outsource cryptographic computations," in *Theory of Cryptography (TCC'05)*, LNCS 3378, pp. 264–282, Springer, 2005.

[12] K. Li and H. Ma, "Outsourcing decryption of multi-authority ABE ciphertexts," *International Journal of Network Security*, vol. 16, no. 4, pp. 286–294, 2014.

[13] X. Ma, J. Li, F. Zhang, "Outsourcing computation of modular exponentiations in cloud computing," *Cluster Computing*, vol. 16, no. 4, pp. 787–796, 2013.

[14] X. Ma, F. Zhang, J. Li, "Verifiable evaluation of private polynomials," in *Fourth IEEE International Conference on Emerging Intelligent Data and Web Technologies (EIDWT'13)*, pp. 451–458, 2013.

[15] P. Q. Nguyen, I. E. Shparlinski, J. Stern, "Distribution of modular sums and the security of the server aided exponentiation," in *Cryptography and Computational Number Theory*, pp. 331–342, 2001.

[16] T. P. Pedersen, "Non-interactive and information-theoretic secure verifiable secret sharing," in *Annual International Cryptology Conference (CRYPTO'88)*, LNCS 576, pp. 129–140, Springer, 1991.

[17] K. Peng, "Critical survey of existing publicly verifiable secret sharing schemes," *IET Information Security*, vol. 6, no. 4, pp. 249–257, 2012.

[18] J. Singh, "Cyber-attacks in cloud computing: A case study," *International Journal of Electronics and Information Engineering*, vol. 1, no. 2, pp. 78–87, 2014.

[19] Y. Wang, Q. Wu, D. S. Wong, et al., "Securely outsourcing exponentiations with single untrusted program for cloud storage," in *19th European Symposium on Research in Computer Security (ESORICS'14)*, LNCS 8712, pp. 326–343, Springer, 2014.

[20] Z. Wang, Y. Lu, G. Sun, "A policy-based deduplication mechanism for securing cloud storage," *International Journal of Electronics and Information Engineering*, vol. 2, no. 2, pp. 70–79, 2015.

[21] F. Zhang, S. Rehaneh, "Private outsourcing of polynomial evaluation and matrix multiplication using multilinear maps," in *Cryptology and Network Security*, LNCS 8257, pp. 329–348, Springer, 2013.

[22] M. Zareapoor, P. Shamsolmoali, and M. A. Alam, "Establishing safe cloud: Ensuring data security and performance evaluation," *International Journal of Electronics and Information Engineering*, vol. 1, no. 2, pp. 88–99, 2014.

Biography

Jianxing Cai is a master candidate of Shanghai University. His research interests include public key cryptography and verifiable outsourcing computation.

Yanli Ren is an associate professor in School of Communication and Information Engineering at Shanghai University. She was awarded a MS degree in applied mathematics in 2005 from Shanghai Normal University, China, and a PhD degree in computer science and technology in 2009 from Shanghai Jiao Tong University, China. Her research interests include applied cryptography, secure computing, and network security.

Chunshui Huang is a master of Shanghai University. His research interests include public key cryptography and verifiable outsourcing computation.