New Approaches for Secure Outsourcing Algorithm for Modular Exponentiations

Xi-Jun Lin [*], Lin Sun [†], Haipeng Qu [‡]and Xiaoshuai Zhang [§]

January 19, 2016

Abstract: Outsourcing paradigm is one of the most attractive benefits of cloud computing, where computation workloads can be outsourced to cloud servers by the resource-constrained devices, such as RFID tags. With this paradigm, cloud users can avoid setting up their own infrastructures. As a result, some new challenges, such as security and checkability, are inevitably introduced. In this paper, we address the problem of secure outsourcing algorithm for modular exponentiations in the one-malicious version of two untrusted program model. We show that our proposed algorithm is more efficient than the state-of-the-art algorithms. On the other hand, we point out in this paper that the first outsource-secure algorithm for simultaneous modular exponentiations proposed recently is insecure, where the sensitive information can be leaked to the malicious servers. As a result, we propose a new and more efficient algorithm for simultaneous modular exponentiations. We also propose the constructions for outsource-secure Cramer-Shoup encryptions and Schnorr signatures which are also more efficient than the state-of-the-art algorithms.

Key words: secure outsourcing algorithm; modular exponentiations; one-malicious model

1 Introduction

Cloud computing is a new computing paradigm which primarily relies on technologies such as utility computing, Service Oriented Architecture, etc. It enables IT resources and capacities to be provided as services over the Internet. Outsourcing paradigm is one of the most attractive benefits of cloud computing, where computation workloads can be outsourced to cloud servers by the resource-constrained devices, such as RFID tags. With this paradigm, cloud users can avoid setting up their own infrastructures. As a result, some new challenges and security concerns [32, 35] are inevitably introduced, which are shown as follows.

- Security: First of all, the cloud servers are semi-trusted, while the outsourced computations may contain some sensitive information which should not be leaked to the

[*]X.J.Lin is with the Department of Computer Science and Technology, Ocean University of China. Qingdao 266100, P.R.China. email: linxj77@163.com

[†]L.Sun is with the College of Liberal Arts, Qingdao University. Qingdao 266071, P.R.China.

[‡]H.Qu is with the Department of Computer Science and Technology, Ocean University of China. Qingdao 266100, P.R.China.

[§]X.Zhang is with the Department of Computer Science and Technology, Ocean University of China. Qingdao 266100, P.R.China.

servers. Hence, as for the secrecy of the outsourcing computations, it is required that the servers should not learn anything about what it is actually computing (including the secret inputs and the outputs).

- Checkability: Some invalid results may be returned from the cloud servers due to some software bugs. Therefore, the outsourcers should be able to verify any failures caused by the cloud servers misbehaves. It is clear that the verification must be far more efficient than the outsourced computation itself.

1.1 Related Work

The impossibility of secure outsourcing an exponential computation while locally doing only polynomial time work was proved in [2]. In the computer theory community, how to securely outsource different kinds of expensive scientific computations, such as matrix multiplications and quadrature [3], receives considerable attentions. However, the disguise technique used in [3] allows leakage of sensitive information. In [4, 10], how to securely outsource sequence comparisons were studied. The problem of secure outsourcing for widely applicable linear algebra computations was addressed by Benjamin and Atallah [7]. However, homomorphic encryptions which are expensive operations have to be required. With the contribution of so-called weak secret hiding assumption, Atallah and Frikken [1] proposed improved protocols for addressing this problem. Recently, Wang et al. [36] presented efficient approaches for secure outsourcing of linear programming computations.

On the other hand, the problem of securely outsourcing computations receives more and more attentions in the cryptographic community. The first idea with respect to secure outsourcing expensive computations was introduced in the so-called "wallets with observers" by Chaumand Pedersen [17]. Then, Hohenberger and Lysyanskaya [28] presented the first security model for outsourcing cryptographic computations, and the first outsource-secure algorithm for modular exponentiations was also proposed based on two previous approaches of precomputation [13, 21, 30, 33] and server-aided computation [5, 23, 29, 37].

In recent years, how to securely outsource expensive computations to the cloud servers becomes the hot topic of research in the cryptography community due to the advancement of cloud computing. The first algorithm for secure delegation of elliptic-curve pairings based on an untrusted server model was presented in [19]. However, a disadvantage of the algorithm is that some other expensive operations (scalar multiplications and exponentiations) have to be carried out by the outsourcer although any server misbehave can be detected with probability 1. The concept of ringers to solve the problem of verifying computation completion for the "inversion of one-way function" class of outsourcing computations was introduced in [26] based on the assumption that the servers are not trusted by the outsourcers. The other trust problem of retrieving payments were studied in [9, 15, 16, 34]. The notion of verifiable computation to solve the problem of verifiably outsourcing the computation of an arbitrary functions was first formalized by Gennaro et al. [22], which was followed by a plenty of researches [8, 11, 12, 24, 25]. Moreover, a protocol which allows the outsourcer to efficiently check the outputs of the computations with a computationally sound, non-interactive proof was proposed in [22]. From the idea of [22], the first practical verifiable computation scheme for high degree polynomial functions was proposed in [6]. And new approaches were presented by Green et al. [27] for constructing efficient and secure outsourcing attribute-based encryption (ABE), which was followed by Parno et al. [31] to show a construction of a multi-function verifiable computation scheme.

Recently, Chen et al. [18] presented a new outsource-secure algorithm for modular exponentiations in the one-malicious version of two untrusted program model. Compared with the state-of-the-art algorithm [28], their algorithm is more efficient, and the checkability is also superior. Moreover, they presented the first outsource-secure algorithm for simultaneous modular exponentiations.

1.2 Our Contribution

In this paper, we also address the problem of secure outsourcing algorithm for modular exponentiations in the one-malicious version of two untrusted program model. We show that our proposed algorithm **Exp** is more efficient than the state-of-the-art algorithms [18, 28].

Another contribution of this paper is that we point out that the first outsource-secure algorithm for simultaneous modular exponentiations proposed in [18] is insecure. The sensitive information can be leaked to the malicious servers. As a result, we propose a new outsource-secure algorithm **SExp** for simultaneous modular exponentiations. Compared with the algorithm of outsourcing only one modular exponentiation in [28] and the algorithm in [18], our algorithm is surprisingly more efficient.

Similar to [18, 28], we also propose the constructions for outsource-secure Cramer-Shoup encryptions and Schnorr signatures. The difference is that the approach used in our proposal is by combining **Exp** and **SExp** which can achieve higher efficiency.

1.3 Organization

The rest of the paper is organized as follows: Some security definitions for outsourcing computation are given in Section 2. The proposed new outsource-secure algorithm for modular exponentiations and its security proof are presented in Section 3. In Section 4, we first present the cryptanalysis of the algorithm for simultaneous modular exponentiations proposed in [18], and then propose our algorithm. The proposed outsource-secure Cramer-Shoup encryptions and Schnorr signatures are given in Section 5, which is followed by the last section to conclude our work.

2 Preliminaries

2.1 Definition of Outsource-Security

Roughly speaking, T securely outsources some work to U, and (T, U) is an outsource-secure implementation of a cryptographic algorithm Alg if 1) T and U implement Alg, i.e., $Alg = T^U$ and 2) suppose that T is given oracle access to an adversary U' (instead of U) that records all of its computation over time and tries to act maliciously, U' cannot learn anything interesting about the input and output of $T^{U'}$. The formal definitions for secure outsourcing of a cryptographic algorithm [28] is shown as follows.

Definition 1 (Algorithm with Outsource-I/O) *An algorithm Alg obeys the outsource input/output specification if it 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 $\mathcal{A} = (E, U')$ knows about them, where E is the adversarial environment that submits adversarially chosen inputs to Alg, and U' is the adversarial software operating in place of oracle U. The* *first input is called the honest, secret input, which is unknown to 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. In addition, there are two adversarially-chosen inputs generated by the environment E: the adversarial, protected input, which is known to E, but protected from U'; and the adversarial, unprotected input, which may be known by E and U. Similarly, the first output called secret is unknown to 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 parties of $\mathcal{A}$.*

The following definition of outsource-security proposed in [28] ensures that the malicious environment E cannot gain any knowledge of 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 said to be an outsource-secure implementation of Alg if:*

1. *Correctness: T^U is a correct implementation of Alg.*
2. *Security: For all probabilistic polynomial-time adversaries $\mathcal{A} = (E, U')$, there exist probabilistic expected polynomial-time simulators (S_1, S_2) such that the following pairs of random variables are computationally indistinguishable.*
   - *Pair One. $EVIEW_{real} \sim EVIEW_{ideal}$:*
     - *The view that the adversarial environment E obtains by participating in the following real process:*

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

$EVIEW_{real} \sim EVIEW^i_{real}$ if $stop^i = TRUE$.

The real process proceeds in rounds. In round i, the honest (secret, protected, and unprotected) inputs $(x^i_{hs}, x^i_{hp}, x^i_{hu})$ are picked using an honest, stateful process I to which the environment E does not have access. Then E, based on its view from the last round, chooses

(a) the value of its $estate^i$ variable as a way of remembering what it did next time it is invoked;

(b) which previously generated honest inputs $(x^i_{hs}, x^i_{hp}, x^i_{hu})$ to give to $T^{U'}$ (note that E can specify the index j^i of these inputs, but not their values);

(c) the adversarial, protected input x^i_{ap};

(d) the adversarial, unprotected input x^i_{au};

(e) the Boolean variable $stop^i$ that determines whether round i is the last round in this process.

Next, the algorithm $T^{U'}$ *is run on the inputs* $(tstate^{i-1}, x_{hs}^{j^i}, x_{hp}^{j^i}, x_{hu}^{j^i}, x_{ap}^{i}, x_{au}^{i})$, *where* $tstate^{i-1}$ *is* T*'s previously saved state, and produces a new state* $tstate^i$ *for* T*, as well as the secret* y_s^i*, protected* y_p^i *and unprotected* y_u^i *outputs. The oracle* U' *is given its previously saved state,* $ustate^{i-1}$*, as input, and the current state of* U' *is saved in the variable* $ustate^i$*. The view of the real process in round* i *consists of* $estate^i$*, and the values* y_p^i *and* y_u^i*. The overall view of* E *in the real process is just its view in the last round (i.e.,* i *for which* $stop^i = TRUE$*.).*

– *The ideal process:*

$$\begin{array}{rcl} EVIEW_{ideal}^i & = & \{(istate^i, x_{hs}^i, x_{hp}^i, x_{hu}^i) \\ & \leftarrow & I(1^k, istate^{i-1}); \\ & & (estate^i, j^i, x_{ap}^i, x_{au}^i, stop^i) \\ & \leftarrow & E(1^k, EVIEW_{ideal}^{i-1}, x_{hp}^i, x_{hu}^i); \\ & & (astate^i, y_s^i, y_p^i, y_u^i) \\ & \leftarrow & Alg(astate^{i-1}, x_{hs}^{j^i}, x_{hp}^{j^i}, x_{hu}^{j^i}, x_{ap}^i, x_{au}^i); \\ & & (sstate^i, ustate^i, Y_p^i, Y_u^i, replace^i) \\ & \leftarrow & S^{U'(ustate^{i-1})}(sstate^{i-1}, \cdots, x_{hp}^{j^i}, x_{hu}^{j^i}, x_{ap}^i, x_{au}^i, y_p^i, y_u^i); \\ & & (z_p^i, z_u^i) = replace^i(Y_p^i, Y_u^i) + (1 - replace^i)(y_p^i, y_u^i) : \\ & & (estate^i, z_p^i, z_u^i)\} \end{array}$$

$EVIEW_{ideal} = EVIEW_{ideal}^i$ *if* $stop^i = TRUE$*:*
The ideal process also proceeds in rounds. In the ideal process, we have a stateful simulator S_1 *who, shielded from the secret input* x_{hs}^i*, but given the non-secret outputs that* Alg *produces when run all the inputs for round* i*, decides to either output the values* (y_p^i, y_u^i) *generated by* Alg*, or replace them with some other values* (Y_p^i, Y_u^i)*. Note that this is captured by having the indicator variable* $replace^i$ *be a bit that determines whether* y_p^i *will be replaced with* Y_p^i*. In doing so, it is allowed to query oracle* U'*; moreover,* U' *saves its state as in the real experiment.*

- *Pair Two.* $UVIEW_{real} \sim UVIEW_{ideal}$*:*
  – *The view that the untrusted software* U' *obtains by participating in the real process described in* Pair One. $UVIEW_{real} = ustate^i$ *if* $stop^i = TRUE$*.*
  – *The ideal process:*

$$\begin{array}{rcl} UVIEW_{ideal}^i & = & \{(istate^i, x_{hs}^i, x_{hp}^i, x_{hu}^i) \\ & \leftarrow & I(1^k, istate^{i-1}); \\ & & (estate^i, j^i, x_{ap}^i, x_{au}^i, stop^i) \\ & \leftarrow & E(1^k, estate^{i-1}, x_{hp}^i, x_{hu}^i, y_p^i, y_u^i); \\ & & (astate^i, y_s^i, y_p^i, y_u^i) \\ & \leftarrow & Alg(astate^{i-1}, x_{hs}^{j^i}, x_{hp}^{j^i}, x_{hu}^{j^i}, x_{ap}^i, x_{au}^i); \\ & & (sstate^i, ustate^i) \leftarrow S_2^{U'(ustate^{i-1})}(sstate^{i-1}, x_{hu}^{j^i}, x_{au}^i) \\ & & : (ustate^i)\} \end{array}$$

$UVIEW_{ideal} = UVIEW_{ideal}^i$ *if* $stop^i = TRUE$*:*

In the ideal process, we have a stateful simulator S_2 who, equipped with only the unprotected inputs (x^i_{hu}, x^i_{au}), queries U'. As before, U' may maintain state.

Definition 3 (α-Efficient, Secure Outsourcing) *A pair of algorithms (T, U) is said to be an α-efficient implementation of Alg if 1) T^U is a correct implementation of Alg and 2) $\forall$ inputs x, the running time of T is no more than an α-multiplicative factor of the running time of Alg.*

Definition 4 (β-Checkable, Secure Outsourcing) *A pair of algorithms (T, U) is said to be a β-checkable implementation of Alg if 1) T^U is a correct implementation of Alg and 2) $\forall$ inputs x, if U' deviates from its advertised functionality during the execution of $T^{U'}(x)$, T will detect the error with probability no less than β.*

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

As mentioned in [18, 28], a secure outourcing algorithm may not provide 100 percent checkability (β). However, it is very practical that a user may run the outsourcing algorithm many times with the same servers. In this case, the probability of being caught will be high if a server cheats frequently. It is clear that a larger β is always better. However, there is a tradeoff between the efficiency (α) and chackability (β).

2.2 One-Malicious Model

In the two untrusted program model for outsourcing exponentiations modulo a prime proposed by Hohenberger and Lysyanskaya [28], the adversarial environment E writes the code for two (potentially different) softwares $U' = (U'_1, U'_2)$. E gives this software to T, advertising a functionality that U'_1 and U'_2 may or may not accurately compute. Note that T controls the channel among E, U_1 and U_2. That is, all communication between any two of E, U'_1 and U'_2 must pass through T. In this case, $\mathcal{A} = (E, U'_1, U'_2)$ denotes the adversary attacking T. Moreover, it is assumed that at most one of U'_1 and U'_2 deviates from its advertised functionality on a non-negligible fraction of the inputs, while we do not know which one. The security means that there is a simulator S for both. This is named as the one malicious version of two untrusted program model (i.e., "one-malicious model" for the simplicity). As mentioned in [18], one-malicious model can be viewed as a special case of refereed delegation of computation model [14] when the number of servers $n = 2$.

3 New and Secure Outsourcing Algorithm of Modular Exponentiations

In [18, 28], to speed up the computations a subroutine *Rand* is used, the inputs for which are a prime p, a generator $g \in \mathbb{Z}_p^*$, and possibly some other values. For each invocation *Rand* outputs a random, independent pair of the form (b, g^b), where $b \in \mathbb{Z}_q$ and $q | p - 1$.

Two approaches can be used to implement the functionality of *Rand*. One is the so-called table-lookup method, i.e., a table of random, independent pairs, which is loaded into the memory of T, is generated in advance by a trusted entity. For each invocation of

Rand, T just retrieves a new pair from the table. The other is to apply the well-known preprocessing algorithms, such as the EBPV generator [30]. The EBPV generator, secure against adaptive adversaries, runs in time $O(log^2 n)$ for an n-bit exponent. It is conjectured that with a sufficiently large subset of truly random (k, g^k) pairs the output distribution of the EBPV generator is statistically close to the uniform distribution. Hence, it should be noted that the output of the *Rand* can never be controlled by T.

3.1 The Proposed Algorithm for Outsource-Secure Exponentiation Modulo a Prime

Here, we present a new secure outsourcing algorithm **Exp** for exponentiation modulo a prime in the one-malicious model. In **Exp**, T outsources its modular exponentiation computations to U_1 and U_2 by invoking *Rand*. The security requirement is that the adversary $\mathcal{A}$ cannot know any useful information about the inputs and outputs of **Exp**. Similar to [18, 28], $U_i(x, y) \to y^x$ also denotes that U_i takes as inputs (x, y) and outputs $y^x \pmod p$, where $i = 1, 2$.

Let p, q be two large primes and $q|p-1$. The input of **Exp** is $a \in \mathbb{Z}_q^*$, and $u \in \mathbb{Z}_p^*$ such that $u^q = 1 \pmod p$ (for an arbitrary base u and an arbitrary power a). The output of **Exp** is $u^a \pmod p$. Note that a may be secret or (honest/adversarial) protected and u may be (honest/adversarial) protected.

Our trick is a more efficient solution to firstly randomize u and then logically split a into random looking pieces. The proposed algorithm **Exp**(a, u) is given as follows:

1. Run *Rand* thrice to create three blinding pairs (α, g^α), (β, g^β) and (γ, g^γ). We denote $v = g^\alpha$, $\mu = g^\beta$ and $\lambda = g^\gamma$.

2. The first step is to randomize u with v and μ as follows:

$$\begin{cases} w_1 = vu \\ w_2 = \mu u \end{cases} \tag{1}$$

3. The second step is to logically split a into two random looking pieces $l = \frac{\gamma + a\beta}{\beta - \alpha}$ and $k = a - l$. Clearly, l and k satisfy the following equations

$$\begin{cases} l + k = a \\ \alpha l + \beta k = -\gamma \end{cases} \tag{2}$$

4. Next, pick four random numbers $d, e \in \mathbb{Z}_q^*$ and $f, h \in \mathbb{Z}_p^*$.

5. Query U_1 in random order as
   $U_1(l, w_1) \to w_1^l$;
   $U_1(d, f) \to f^d$;
   $U_1(e, h) \to h^e$.
   Similarly, query U_2 in random order as
   $U_2(k, w_2) \to w_2^k$;
   $U_2(d, f) \to f^d$;

$U_2(e, h) \rightarrow h^e$.

6. Finally, check that both U_1 and U_2 produce the correct outputs, i.e., $U_1(d, f) = U_2(d, f)$ and $U_1(e, h) = U_2(e, h)$. If not, output "error"; otherwise, output $\lambda w_1^l w_2^k$.

Clearly, $\lambda w_1^l w_2^k = g^\gamma (g^\alpha u)^l (g^\beta u)^k = g^\gamma g^{\alpha l + \beta k} u^{l+k} = g^\gamma g^{-\gamma} u^a = u^a$.

Note that in the one-malicious model, the above two test queries (i.e., $U_1(d, f) = U_2(d, f)$ and $U_1(e, h) = U_2(e, h)$) imply both U_1 and U_2 produce the correct outputs. Similar to [18], **Exp** can be trivially extended to outsource-secure scalar multiplications on elliptic curves, whose details are omitted here.

3.2 Security Proof

Theorem 1 *In the one-malicious model, the algorithms (T, U_1, U_2) are an outsource-secure implementation of* **Exp***, where the input (a, u) may be honest, secret; or honest, protected; or adversarial, protected.*

Proof. The correctness property is straight-forward, and we only focus on security. The proof is very similar to [18, 28]. Let $\mathcal{A} = (E, U_1', U_2')$ be a probabilistic polynomial time adversary that interacts with a probabilistic polynomial time algorithm T in the one-malicious model.

Firstly, we prove *Pair One* $EVIEW_{real} \sim EVIEW_{ideal}$ (The external adversary, E, learns nothing.):

If the input (a, u) is anything other than honest, secret, then the simulation is trivial: the simulator S_1 just behaves the same way as in the real execution.

If (a, u) is an honest, secret input, S_1 behaves as follows: Upon receiving the input on round i, S_1 ignores it, and instead makes three random queries of the form $(\alpha_j \in \mathbb{Z}_q, \beta_j \in \mathbb{Z}_p^*)$ to both U_1' and U_2'. S_1 randomly tests two outputs from each program (i.e., $\beta_j^{\alpha_j}$). If an error is detected, S_1 saves all states and outputs $Y_p^i = \text{"}error\text{"}$, $Y_u^i = \phi$, $replace^i = 1$. If no error is detected, S_1 checks the remaining two outputs. If all checks pass, S_1 outputs $Y_p^i = \phi$, $Y_u^i = \phi$, $replace^i = 0$; otherwise, S_1 picks a random number $r \in \mathbb{Z}_p^*$ and outputs $Y_p^i = r$, $Y_u^i = \phi$, $replace^i = 1$. In either case, S_1 saves the appropriate states.

The input distributions to (U_1', U_2') in the real and ideal experiments are computationally indistinguishable. In the ideal experiment, the inputs are chosen uniformly at random. In the real experiment, each part of all three queries that T makes to any one program is independently re-randomized and thus computationally indistinguishable from random. If (U_1', U_2') behave honestly in the i-th round, then $EVIEW_{real}^i \sim EVIEW_{ideal}^i$ (this is because $T^{(U_1', U_2')}$ perfectly executes **Exp** in the real experiment and S_1 chooses not to replace the output of **Exp** in the ideal experiment). If one of (U_1', U_2') gives an incorrect output in the i-th round, then it will be detected by both T and S_1 with probability $\frac{2}{3}$, resulting in an output of "error"; otherwise, the output of **Exp** will be successfully corrupted (with probability $\frac{1}{3}$). In the real experiment, the three outputs generated by (U_1', U_2') are multiplied together along with a random value, thus a corrupted output of **Exp** will look incorrect, but random to E. In the ideal experiment, S_1 also simulates with a random value $r \in \mathbb{Z}_p^*$. Thus, $EVIEW_{real}^i \sim EVIEW_{ideal}^i$ even when one of (U_1', U_2') behaves dishonestly in the i-th round. By the hybrid argument, we conclude that $EVIEW_{real} \sim EVIEW_{ideal}$.

Secondly, we prove *Pair Two* $UVIEW_{real} \sim UVIEW_{ideal}$ (The untrusted software, (U_1', U_2'), learns nothing.):

The simulator S_2 always behaves as follows: Upon receiving the input on the i-th round, S_2 ignores it, and instead makes three random queries of the form $(\alpha_j \in \mathbb{Z}^*q, \beta_j \in \mathbb{Z}_p^*)$ to both U_1' and U_2'. Then S_2 saves its states and the states of (U_1', U_2'). E can easily distinguish between these real and ideal experiments (note that the output in the ideal experiment is never corrupted). However, E cannot communicate this information to (U_1', U_2'). This is because in the i-th round of the real experiment, T always re-randomizes its inputs to (U_1', U_2'). In the ideal experiment, S_2 always creates random, independent queries for (U_1', U_2'). Thus, for each round i we have $UVIEW_{real}^i \sim UVIEW_{ideal}^i$. By the hybrid argument, we conclude that $UVIEW_{real} \sim UVIEW_{ideal}$. □

Theorem 2 *In the one-malicious model, the algorithms* $(T, (U_1, U_2))$ *are an* $(O(\frac{log^2 n}{n}, \frac{2}{3})$-*outsource-secure implementation of* **Exp**.

Proof. It takes the proposed algorithm **Exp** roughly $1.5n$ MM to compute u^a by the square-and-multiply method for an arbitrary base to an arbitrary power. Moreover, **Exp** makes 3 calls to *Rand* plus 6 modular multiplication (MM) and 1 modular inverse (MInv) in order to compute u^a (we omit other operations such as modular additions). Also, **Exp** takes $O(log^2 n)$ or $O(1)$ MM using the EBPV generator or table-lookup method for *Rand*, respectively, where n is the bit of the a. Thus, the algorithms $(T, (U_1, U_2))$ are an $O(\frac{log^2 n}{n})$-efficient implementation of **Exp**.

On the other hand, U_1 (resp. U_2) cannot distinguish the two test queries from the three real queries T makes. If U_1 (resp. U_2) fails during any execution of **Exp**, it will be detected with probability $\frac{2}{3}$. □

3.3 Efficiency

Here, we compare our proposal with the state-of-the-art algorithms [18, 28]. Let MM denote a modular multiplication, $MInv$ denote a modular inverse, Invoke(*Rand*) denote an invocation of *Rand*, and Invoke(U_1) and Invoke(U_2) denote the invocation of U_1 and U_2 respectively. Other operations such as modular additions are omitted. The comparison of the efficiency and the checkability is presented in Table 1.

Compared with the other two algorithms, our proposal is more efficient, which requires only 6 MM, 1 MInv, 3 invocation of *Rand*, and 3 invocation of U_1 and U_2 for each modular exponentiation. As the basic operation in most cryptographic protocols, millions of the modular exponentiations may be performed by the cloud server every day. Thus, our proposal can save huge of such computational resources.

Table 1: Efficiency Comparison

	Algorithm [28]	Algorithm [18]	Ours
MM	9	7	6
MInv	5	3	1
Invoke(*Rand*)	6	5	3
Invoke(U_1)	4	3	3
Invoke(U_2)	4	3	3
Checkability	$\frac{1}{2}$	$\frac{2}{3}$	$\frac{2}{3}$

4 New and Secure Outsource Algorithm of Simultaneous Modular Exponentiations

The simultaneous modular exponentiations $u_1^a u_2^b \pmod p$ plays an important role in many cryptographic primitives. Chen et al. first introduced the problem of outsource-secure algorithm *SExp* of simultaneous modular exponentiations [18]. However, we point out that their algorithm is insecure. And then we propose a new outsource-secure algorithm of simultaneous modular exponentiations.

4.1 Cryptanalysis of Chen et al.'s Algorithm

Here, we recall Chen et al.'s outsource-secure algorithm of simultaneous modular exponentiations, and then show our cryptanalysis.

Let p, q be two large primes and $q|p-1$. Given two arbitrary bases $u_1, u_2 \in \mathbb{Z}_p^*$ and two arbitrary powers $a, b \in \mathbb{Z}_q^*$ such that the order of u_1 and u_2 is q. The output of *SExp* is $u_1^a u_2^b \pmod p$.

1. Run *Rand* twice to create two blinding pairs (α, g^α) and (β, g^β). We denote $v = g^\alpha$ and $\mu = g^\beta$.
2. The first logical divisions are
$$u_1^a u_2^b = (vw_1)^a (vw_2)^b = g^\beta g^\gamma w_1^a w_2^b,$$
where $w_1 = u_1/v$, $w_2 = u_2/v$, and $\gamma = (a+b)\alpha - \beta$.
The second logical divisions are
$$u_1^a u_2^b = g^\beta g^\gamma w_1^a w_2^b = g^\beta g^\gamma w_1^k w_1^l w_2^t w_2^s,$$
where $l = a - k$ and $s = b - t$.
3. Next, run *Rand* to obtain three pairs $(t_1, g^{t_1}), (t_2, g^{t_2})$ and (t_3, g^{t_3}).
4. Query U_1 in random order as
$U_1(t_2/t_1, g^{t_1}) \to g^{t_2}$;
$U_1(\gamma/t_3, g^{t_3}) \to g^\gamma$;
$U_1(k, w_1) \to w_1^k$;
$U_1(t, w_2) \to w_2^t$.
Similarly, query U_2 in random order as
$U_2(t_2/t_1, g^{t_1}) \to g^{t_2}$;
$U_2(\gamma/t_3, g^{t_3}) \to g^\gamma$;
$U_2(l, w_1) \to w_1^l$;
$U_2(s, w_2) \to w_2^s$.
5. Finally, check that both U_1 and U_2 produce the correct outputs, i.e., $g^{t_2} = U_1(t_2/t_1, g^{t_1}) = U_2(t_2/t_1, g^{t_1})$ and $U_1(\gamma/t_3, g^{t_3}) = U_2(\gamma/t_3, g^{t_3})$. If not, output "error"; otherwise, compute $u_1^a u_2^b = \mu g^\gamma w_1^k w_1^l w_2^t w_2^s$.

In the original paper, Chen et al. claimed that the above algorithm is secure under one-malicious model.

Cryptanalysis

We stress here again that the security of the outsourcing computation requires that U_1' and U_2' should not learn anything about what it is actually computing (including the secret inputs and the outputs). On the other hand, it is allowed that E and U_1' (or U_2') develop a joint strategy in advance [28] (once they begin interacting with T, they no longer have a direct communication channel).

The weakness of the algorithm [18] is due to w_1 and w_2. Note that given u_1 and u_2, for any w_1 and w_2, w_1/w_2 is always equal to u_1/u_2. Since w_1, w_2 are inputs for both U_1' and U_2', we assume without loss of generality that U_1' is malicious.

Suppose that E is a malicious software vendor. Let u_1 and u_2 be partial public keys of its competing software vendor. E then computes

$$\phi = u_1/u_2, \psi = u_2/u_1,$$

which are embedded into U_1'. In fact, when *SExp* is invoked, U_1' has the ability to recognize these public keys of the competing software vendor.

Given (a, b, u_1, u_2) T will invoke *SExp* for computing $u_1^a u_2^b$. Let (c_i, d_i) be the inputs to U_1' in the i-th round. We know that for all invocations of U_1', there must exist two numbers d_i and d_j $(i \neq j)$ such that $d_i = w_1, d_j = w_2$ or $d_i = w_2, d_j = w_1$, that is, $d_i/d_j = \phi$ or $d_i/d_j = \psi$.

Hence, when $U_1'(c_i, d_i)$ is invoked, U_1' can compute $\delta = d_i/d_j$ for each d_j where $j < i$ (note that U_1' can record all inputs from T). And then it checks whether or not $\delta = \phi$ or $\delta = \psi$ holds. If it is the case, U_1' knows that it is performing the work with the public key of the competing software vendor.

Similarly, E can embed a list of public keys of its competing software vendors. Then, the information with respect to all these vendors will be leaked to the malicious software, which contradicts to the basic security requirement for the outsourcing computation.

4.2 The Proposed Algorithm

Now, we propose a new outsource-secure algorithm of simultaneous modular exponentiations. Compared with Chen et al.'s algorithm [18], the security is assured. Moreover, our proposal is more efficient. More in details, our proposal **SExp** is as follows.

1. Run *Rand* thrice to create three blinding pairs (α, g^α), (β, g^β) and (γ, g^γ). We denote $v = g^\alpha$, $\mu = g^\beta$ and $\lambda = g^\gamma$.

2. The first logical divisions are
$$u_1^a u_2^b = g^{-\pi} w_1^a w_2^b,$$
where $w_1 = u_1 v$, $w_2 = u_2 \mu$, and $\pi = a\alpha + b\beta$.

   The second logical divisions are
$$u_1^a u_2^b = g^{-\pi} w_1^a w_2^b = g^\gamma g^x w_1^k w_1^l w_2^t w_2^s = \lambda g^x w_1^k w_1^l w_2^t w_2^s,$$

where $x = q - \pi - \gamma$, $l = a - k$ and $s = b - t$.

3. Next, pick two random numbers $d \in \mathbb{Z}_q^*$ and $f \in \mathbb{Z}_p^*$, and run *Rand* to obtain a pair (t_1, g^{t_1}).

4. Query U_1 in random order as
   $U_1(x/t_1, g^{t_1}) \rightarrow g^x$;
   $U_1(d, f) \rightarrow f^d$;
   $U_1(k, w_1) \rightarrow w_1^k$;
   $U_1(t, w_2) \rightarrow w_2^t$.
   Similarly, query U_2 in random order as
   $U_2(x/t_1, g^{t_1}) \rightarrow g^x$;
   $U_2(d, f) \rightarrow f^d$;
   $U_2(l, w_1) \rightarrow w_1^l$;
   $U_2(s, w_2) \rightarrow w_2^s$.

5. Finally, check that both U_1 and U_2 produce the correct outputs, i.e., $U_1(x/t_1, g^{t_1}) = U_2(x/t_1, g^{t_1})$ and $U_1(d, f) = U_2(d, f)$. If not, output "error"; otherwise, compute

$$u_1^a u_2^b = \lambda g^x w_1^k w_1^l w_2^t w_2^s.$$

Similar to Theorem 1 and 2, we can easily prove the following theorem:

Theorem 3 *In the one-malicious model, the algorithms $(T, (U_1, U_2))$ are an $O(\frac{log^2 n}{n}), \frac{1}{2})$-outsource-secure implementation of* **SExp**.

4.3 Efficiency

Note that **SExp** requires only 10 MM, 1 MInv, 4 invocation of *Rand*, and 4 invocation of U_1 and U_2 for each modular exponentiation. Table 2 presents the comparison of the efficiency and the checkability with Hohenberger-Lysyanskaya's algorithm of outsourcing one modular exponentiation (Surprisingly, our proposal is even more efficient) and Chen et al.'s algorithm.

Table 2: Efficiency Comparison

	Algorithm [28]	Algorithm [18]	Ours
MM	9	10	10
MInv	5	4	1
Invoke(*Rand*)	6	5	4
Invoke(U_1)	4	4	4
Invoke(U_2)	4	4	4
Checkability	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{2}$

5 Secure Outsourcing Algorithms for Encryption and Signatures

In this section, we propose two secure outsourcing algorithms for Cramer-Shoup encryption scheme [20] and Schnorr signature scheme [33].

5.1 Outsource-Secure Cramer-Shoup Encryptions

The proposed outsource-secure Cramer-Shoup encryption scheme CS1b consists of the following efficient algorithms:

- *System Parameters Generation*: Let $\mathbb{G}$ be an abelian group of a large prime order q. Let g be a generator of $\mathbb{G}$. Define a cryptographic secure hash function $H : \mathbb{G}^3 \to \mathbb{Z}_q$. The system parameters are $SP = \{\mathbb{G}, q, g, H\}$.
- *Key Generation*: On input 1^l, run the key generation algorithm to obtain the secret/public key pair (SK, PK), here $SK = (w, x, y, z) \in_R \mathbb{Z}_q^* \times \mathbb{Z}_q^3$, $PK = (W, X, Y, Z) = (g^w, g^x, g^y, g^z)$.
- *Encryption*: On input the public key PK and a message $m \in \mathbb{G}$, the outsourcer T runs the subroutines *Rand*, **Exp** and **SExp**, and generates the ciphertext C as follows:
  1. T runs *Rand* to obtain a pair $(k, r = g^k)$.
  2. T firstly runs **Exp** to obtain **Exp**$(k, W) \to s$, **Exp**$(k, Z) \to t$ and then computes $e = mt$, and $h = H(r, s, e)$.
  3. T runs **SExp** to obtain **SExp**$(k, kh, X, Y) \to X^k Y^{kh} = \gamma$.
  4. T outputs the ciphertext $C = (r, s, e, \gamma)$.
- *Decryption*: On input the secret key SK, and the ciphertext $C = (r, s, e, \gamma)$, the outsourcer T' runs the subroutine **Exp** and computes the message m as follows:
  1. T' computes $h = H(r, s, e)$.
  2. T' runs **Exp** to obtain **Exp**$(w, r) \to \psi_1$ and **Exp**$(x + yh, r) \to \psi_2$.
  3. If and only if $s = \psi_1$ and $\gamma = \psi_2$, T' runs **Exp** to obtain **Exp**$(q - z, r) \to t$ and then computes $m = et$.

Compared with [18], we use both **Exp** and **SExp** as subroutines, and the efficiency is improved greatly which is shown in Table 3. Note that the checkability in *Encryption* is lower than [18]. The solution is to make two additional test queires in **SExp**, which will improve the checkability to $\frac{2}{3}$. Clearly, our proposal is still more efficient even in this case.

5.2 Outsource-Secure Schnorr Signatures

The proposed outsource-secure Schnorr signature scheme consists of the following efficient algorithms:

- *System Parameters Generation*: Let p and q be two large primes that satisfy $q|p-1$. Let g be an element in $\mathbb{Z}_p$ such that $g^q = 1 \pmod{p}$. Define a cryptographic secure hash function $H : \{0,1\}^* \to \mathbb{Z}_p$. The system parameters are $SP = \{p, q, g, H\}$.

Table 3: Efficiency Comparison

	Algorithm [18] (Enc/Dec)	Ours (Enc/Dec)
MM	30/23	24/20
MInv	12/10	3/3
Invoke($Rand$)	21/15	11/9
Invoke(U_1)	12/9	10/9
Invoke(U_2)	12/9	10/9
Checkability	$\frac{2}{3}$ / $\frac{2}{3}$	$\frac{3}{5}$ / $\frac{2}{3}$

- *Key Generation*: On input 1^l, run the key generation algorithm to obtain the signing/verification key pair (x, y), here $y = g^{-x} \pmod p$.
- *Signature Generation*: On input the singing key x and a message m, the outsourcer T runs the subroutine $Rand$, and generates the signature as follows:
  1. T runs $Rand$ to obtain a pair $(k, r = g^k)$.
  2. T computes $e = H(m\|r)$ and $s = k + xe \pmod q$.
  3. T outputs the signature $\sigma = (e, s)$.
- *Signature Verification*: On input the verification key y, the message m, and the signature $\sigma = (e, s)$, the outsourcer T' runs the subroutine **SExp**, and verifies the signature as follows:
  1. T' runs **SExp** to obtain $\textbf{SExp}(s, e, g, y) \to g^s y^e = r'$.
  2. T' computes $e' = H(m\|r')$.
  3. T' outputs 1 if and only if $e' = e$.

The efficiency comparison is given in Table 4. Similarly, to improve the checkability to $\frac{2}{3}$ two additional test queires can be made in **SExp**. Clearly, our proposal is still more efficient even in this case.

Table 4: Efficiency Comparison of Signature Verification

	Algorithm [28]	Algorithm [18]	Ours
MM	19	15	6
MInv	10	6	1
Invoke($Rand$)	12	10	3
Invoke(U_1)	8	6	3
Invoke(U_2)	8	6	3
Checkability	$\frac{1}{2}$	$\frac{2}{3}$	$\frac{1}{2}$

6 Conclusion

In this paper,we propose two outsource-secure and efficient algorithms for modular exponentiations and simultaneous modular exponentiations. Compared with the state-of-the-art algorithms [18, 28], the proposed algorithm is more efficient. Also, we point out that the algorithm for simultaneous modular exponentiations in [18] is insecure. Moreover, we propose the constructions for outsource-secure Cramer-Shoup encryptions and Schnorr signatures.

References

[1] M.J. Atallah and K.B. Frikken, "Securely Outsourcing Linear Algebra Computations," in Proc. 5th ACM Symp. Inf.,Comput.Commun. Secur., 2010, pp. 48-59.

[2] M. Abadi, J. Feigenbaum, and J. Kilian, "On Hiding Information from an Oracle," in Proc. 19th Annu. ACM Symp. Theory Comput., 1987, pp. 195-203.

[3] M.J. Atallah, K.N. Pantazopoulos, J.R. Rice, and E.H. Spafford, "Secure Outsourcing of Scientific Computations," Adv. Comput., vol. 54, pp. 215-272, 2002.

[4] M.J. Atallah and J. Li, "Secure Outsourcing of Sequence Comparisons," Int'l J. Inf. Secur., vol. 4, no. 4, pp. 277-287, Oct. 2005.

[5] D. Beaver, J. Feigenbaum, J. Kilian, and P. Rogaway, "Locally Random Reductions: Improvements and Applications," J. Cryptol., vol. 10, no. 1, pp. 17-36, Dec. 1997.

[6] S. Benabbas, R. Gennaro, and Y. Vahlis, "Verifiable Delegation of Computation Over Large Datasets," in Proc. Crypto, 2011, vol. LNCS 6841, pp. 111-131.

[7] D. Benjamin and M.J. Atallah, "Private and Cheating-Free Outsourcing of Algebraic Computations," in Proc. 6th Annu. Conf. Privacy, Secur. Trust, 2008, pp. 240-245.

[8] M. Ben-Or, S. Goldwasser, J. Kilian, and A. Wigderson, "Multi-Prover Interactive Proofs: How to Remove Intractability Assumptions," in Proc. ACM Symp. Theory Comput., 1988, pp. 113-131.

[9] M. Blanton, "Improved Conditional E-Payments," in Proc. ACNS, 2008, vol. LNCS 5037, pp. 188-206.

[10] M. Blanton, M.J. Atallah, K.B. Frikken, and Q. Malluhi, "Secure and Efficient Outsourcing of Sequence Comparisons," in Proc. ESORICS, 2012, vol. LNCS 7459, pp. 505-522.

[11] M. Blum, M. Luby, and R. Rubinfeld, "Program Result Checking Against Adaptive Programs and in Cryptographic Settings," Proc. DIMACS Series Discrete Math. Theoretical Comput. Sci., 1991, pp. 107-118.

[12] M. Blum, M. Luby, and R. Rubinfeld, "Self-Testing/Correcting with Applications to Numerical Problems," J. Comput. Syst. Sci., vol. 47, no. 3, pp. 549-595, Dec. 1993.

[13] V. Boyko, M. Peinado, and R. Venkatesan, "Speeding Up Discrete Log and Factoring Based Schemes via Precomputations," in Proc. Eurocrypt, 1998, vol. LNCS 1403, pp. 221-232.

[14] R. Canetti, B. Riva, and G. Rothblum, "Practical Delegation of Computation using Multiple Servers," in Proc. 18th ACM Conf. Comput. Commun. Secur., 2011, pp. 445-454.

[15] B. Carbunar and M. Tripunitara, “Conditioal Payments for Computing Markets,” in Proc. CANS, 2008, vol. LNCS 5339, pp. 317-331.

[16] B. Carbunar and M. Tripunitara, “Fair Payments for Outsourced Computations,” in Proc. SECON, 2010, pp. 529-537.

[17] D. Chaum and T. Pedersen, “Wallet Databases with Observers,” in Proc. Crypto 1992, 1993, vol. LNCS 740, pp. 89-105.

[18] X. Chen, J. Li, J. Ma, Q. Tang and W. Lou, “New algorithms for secure outsourcing of modular exponentiations,” Parallel and Distributed Systems, IEEE Transactions on, 2014, 25(9). pp. 2386-2396.

[19] B. Chevallier-Mames, J. Coron, N. McCullagh, D. Naccache, and M. Scott, “Secure Delegation of Elliptic-Curve Pairing,” in Proc. CARDIS, 2010, vol. LNCS 6035, pp. 24-35.

[20] R. Cramer and V. Shoup, “Design and Analysis of Practical Public-Key Encryption Schemes Secure Against Adaptive Chosen Ciphertext Attack,” SIAMJ. Comput., vol. 33,no. 1,pp. 167-226, 2004.

[21] S. Even, O. Goldreich, and S. Micali, “On-Line/Off-Line Digital Signatures,” J. Cryptol., vol. 9, no. 1, pp. 35-67, 1996.

[22] R. Gennaro, C. Gentry, and B. Parno, “Non-Interactive Verifiable Computing: Outsourcing Computation to Untrusted Workers,” in Proc. Crypto, 2010, vol. LNCS 6223, pp. 465-482.

[23] M. Girault and D. Lefranc, “Server-Aided Verification: Theory and Practice,” in Proc. ASIACRYPT, 2005, vol. LNCS 3788, pp. 605-623.

[24] S. Goldwasser, Y.T. Kalai, and G.N. Rothblum, “Delegating Computation: Interactive Proofs for Muggles,” in Proc. ACM Symp. Theory Comput., 2008, pp. 113-122.

[25] S. Goldwasser, S. Micali, and C. Rackoff, “The Knowledge Complexity of Interactive Proof-Systems,” SIAM J. Comput., vol. 18, no. 1, pp. 186-208, Feb. 1989.

[26] P. Golle and I. Mironov, “Uncheatable Distributed Computations,” in Proc. CT-RSA, 2001, vol. LNCS 2020, pp. 425-440.

[27] M. Green, S. Hohenberger, and B. Waters, “Outsourcing the Decryption of ABE Ciphertexts,” in Proc. 20th USENIX Conf. Secur., 2011.

[28] S. Hohenberger and A. Lysyanskaya, “How to Securely Outsource Cryptographic Computations,” in Proc. TCC, 2005, vol. LNCS 3378, pp. 264-282, Springer-Verlag: New York, NY, USA.

[29] T. Matsumoto, K. Kato, and H. Imai, “Speeding up Secret Computations with Insecure Auxiliary Devices,” in Proc. Crypto, 1988, vol. LNCS 403, pp. 497-506.

[30] P.Q. Nguyen, I.E. Shparlinski, and J. Stern, “Distribution of Modular Sums and the Security of Server-Aided Exponentiation,” in Proc. Workshop Comput. Number Theory Crypt., 1999, pp. 1-16.

[31] B. Parno, M. Raykova, and V. Vaikuntanathan, "How to Delegate and Verify in Public: Verifiable Computation from Attribute-Based Encryption," in Proc. TCC, 2012, vol. LNCS 7194, pp. 422-439.

[32] K. Ren, C. Wang, and Q. Wang, "Security Challenges for the Public Cloud," IEEE Internet Comput., vol. 16, no. 1, pp. 69-73. 2012.

[33] C.P. Schnorr, "Efficient Signature Generation for Smart Cards," J. Cryptol., vol. 4, no. 3, pp. 161-174, 1991.

[34] L. Shi, B. Carbunar, and R. Sion, "Conditional E-Cash," in Proc. FC, 2007, vol. LNCS 4886, pp. 15-28.

[35] C. Wang, N. Cao, K. Ren, and W. Lou, "Enabling Secure and Efficient Ranked Keyword Search Over Outsourced Cloud Data," IEEE Trans. Parallel Distrib. Syst., vol. 23,no. 8, pp.1467-1479, Aug. 2012.

[36] C. Wang, K. Ren, and J. Wang, "Secure and Practical Outsourcing of Linear Programming in Cloud Computing," in Proc. 30th IEEE Int'l Conf. Comput. Commun., 2011, pp. 820-828.

[37] W. Wu, Y. Mu, W. Susilo, and X. Huang, "Server-Aided Verification Signatures: Definitions and New Constructions," in Proc. ProvSec, 2008, vol. LNCS 5324, pp. 141-155.