New Algorithms for Secure Outsourcing of Modular Exponentiations

Xiaofeng Chen[1], Jin Li[2], Jianfeng Ma[3], Qiang Tang[4], and Wenjing Lou[5]

[1] State Key Laboratory of Integrated Service Networks (ISN), Xidian University, Xi'an 710071, P.R. China
xfchen@xidian.edu.cn
[2] School of Computer Science and Educational Software, Guangzhou University, Guangzhou 510006, P.R. China
jinli71@gmail.com
[3] School of Computer Science and Technology, Xidian University, Xi'an 710071, P.R. China
jfma@mail.xidian.edu.cn
[4] APSIA Group, SnT, University of Luxembourg, 6, rue Richard Coudenhove-Kalergi, L-1359 Luxembourg
tonyrhul@gmail.com
[5] Department of Computer Science, Virginia Polytechnic Institute and State University, USA
wjlou@vt.edu

Abstract. Modular exponentiations have been considered the most expensive operation in discrete-logarithm based cryptographic protocols. In this paper, we propose a new secure outsourcing algorithm for exponentiation modular a prime in the one-malicious model. Compared with the state-of-the-art algorithm [33], the proposed algorithm is superior in both efficiency and checkability. We then utilize this algorithm as a subroutine to achieve outsource-secure Cramer-Shoup encryptions and Schnorr signatures. Besides, we propose the first outsource-secure and efficient algorithm for simultaneous modular exponentiations. Moreover, we prove that both the algorithms can achieve the desired security notions.

Keywords: Cloud computing, Outsource-secure algorithms, Modular exponentiation.

1 Introduction

Cloud computing, the long-standing vision of computing as a utility, enables convenient and on-demand network access to a centralized pool of configurable computing resources. One of the most attractive benefits of the cloud computing is the so-called outsourcing paradigm, where the resource-constraint devices can outsource their large computation workloads to the cloud servers in a pay-per-use manner. As a result, the enterprises can avoid large capital outlays in hardware/software deployment and maintenance.

S. Foresti, M. Yung, and F. Martinelli (Eds.): ESORICS 2012, LNCS 7459, pp. 541–556, 2012.

Despite the tremendous benefits, outsourcing computation also inevitably involves in some new security concerns and challenges. Firstly, the cloud servers are not (fully) trusted. Actually, it is impossible to find a trusted server for all outsourcers in cloud paradigm. On the other hand, the computation tasks often contain some sensitive information that should not be exposed to the cloud servers. Therefore, the first security challenge is the *secrecy* of the outsourcing computation: the cloud servers should not learn anything about what it is actually computing (including the *secret* inputs and the outputs). We argue that the encryption can only provide a partial solution to this problem since it is very difficult to perform meaningful computations over the encrypted data. Secondly, the semi-trusted cloud servers may return an invalid result. For example, the servers might contain a software bug that will fail on a constant number of invocation. Moreover, the servers might decrease the amount of the computation due to financial incentives and then return a computationally indistinguishable (invalid) result. Therefore, the second security challenge is the *checkability* of the outsourcing computation: the outsourcer should have the ability to detect any failures if the cloud servers misbehave. Trivially, the test procedure should never be involved in some other complicated computations since the computationally limited devices such as RFID tags or smartcard may be incapable to accomplish the test. At the very least, it must be *far more* efficient than accomplishing the computation task itself (recall the motivation for outsourcing computations).

The problem of secure outsourcing expensive computations has been well studied in the cryptography community. Chaum and Pedersen [17] firstly introduced the idea of "wallets with observers" that allows a piece of hardware installed on the client's device to carry out some computations for each transaction. Golle and Mironov [31] first introduced the concept of ringers to elegantly solve the problem of verifying computation completion for the "inversion of one-way function" class of outsourcing computations. Hohenberger and Lysyanskaya [33] presented the security model for outsourcing cryptographic computations, and proposed the first outsource-secure algorithm for modular exponentiations.

Our Contribution. In this paper, we propose a new secure outsourcing algorithm of modular exponentiation in the one-malicious model. To the best of our knowledge, it seems that the proposed algorithm is the second one for exponentiation modular a prime. Compared with the state-of-the-art algorithm [33], the proposed algorithm is superior in both efficiency and checkability. Similar to [33], we also utilize this algorithm as a subroutine to achieve outsource-secure Cramer-Shoup encryptions and Schnorr signatures. Another main contribution of this paper is the first outsource-secure and efficient algorithm for *simultaneous* modular exponentiations, which efficiency is (surprisingly) comparable to that of outsourcing only *one* modular exponentiation in [33].

1.1 Related Work

Abadi et al. [2] proved the impossibility of secure outsourcing an exponential computation while locally doing only polynomial time work. Therefore, it is

meaningful only to consider outsourcing expensive polynomial time computations. The theoretical computer science community has devoted considerable attention to the problem of how to securely outsource different kinds of expensive computations. Atallah et al. [3] presented a framework for secure outsourcing of scientific computations such as matrix multiplications and quadrature. However, the solution used the disguise technique and thus allowed leakage of private information. Atallah and Li [4] investigated the problem of computing the edit distance between two sequences and presented an efficient protocol to securely outsource sequence comparisons to two servers. Benjamin and Atallah [8] addressed the problem of secure outsourcing for widely applicable linear algebra computations. However, the proposed protocols required the expensive operations of homomorphic encryptions. Atallah and Frikken [1] further studied this problem and gave improved protocols based on the so-called weak secret hiding assumption. Recently, Wang et al. [45] presented efficient mechanisms for secure outsourcing of linear programming computations.

In the cryptographic community, there are also plenty of research work on the securely outsourcing computations. In 1992, Chaum and Pedersen [17] firstly introduced the notion of wallets with observers, a piece of secure hardware installed on the client's computer to perform some expensive computations. Hohenberger and Lysyanskaya [33] proposed the first outsource-secure algorithm for modular exponentiations based on the two previous approaches of precomputation [15,24,40,42] and server-aided computation [10,29,39,46].

Since the servers (or workers) are not trusted by the outsourcers, Golle and Mironov [31] first introduced the concept of ringers to solve the trust problem of verifying computation completion. The following researchers focused on the other trust problem of retrieving payments [7,19,20,43]. Besides, Gennaro et al. [27] first formalized the notion of verifiable computation to solve the problem of verifiably outsourcing the computation of an arbitrary functions, which has attracted the attention of plenty of researchers [11,13,14,28,30,34,35,38]. Gennaro et al. [27] also proposed a protocol that allowed the outsourcer to efficiently verify the outputs of the computations with a computationally sound, *non-interactive* proof (instead of interactive ones). Benabbas et al. [12] presented the first practical verifiable computation scheme for high degree polynomial functions based on the approach of [27]. In 2011, Green et al. [26] proposed new methods for efficiently and securely outsourcing decryption of attribute-based encryption (ABE) ciphertexts. Based on this work, Parno et al. [41] showed a construction of a multi-function verifiable computation scheme.

1.2 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 modular exponentiations algorithm and its security analysis are given in Section 3. The proposed outsource-secure Cramer-Shoup encryptions and Schnorr

signatures are given in Section 4. The secure and efficient outsourcing algorithm for simultaneous modular exponentiations is given in Section 5. Finally, conclusions will be made in Section 6.

2 Definition of Security

Informally, we say that 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., $\mathsf{Alg} = T^U$ 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 $T^{U'}$. In the following, we introduce the formal definitions for secure outsourcing of a cryptographic algorithm [33].

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 $A = (E, U')$ 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 call 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 A.*

The following definition of outsource-security 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 $A = (E, U')$, there exist probabilistic expected polynomial-time simulators (S_1, S_2) such that the following pairs of random variables are computationally indistinguishable.*
   - *Pair One.* $\text{EVIEW}_{\text{real}} \sim \text{EVIEW}_{\text{ideal}}$*:*
     - *The view that the the adversarial environment E obtains by participating in the following* real *process:*

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

$\mathrm{EVIEW}_{\mathrm{real}} = \mathrm{EVIEW}^i_{\mathrm{real}}$ *if* $\mathrm{stop}^i = \mathrm{TRUE}$.

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 (0) the value of its estate_i *variable as a way of remembering what it did next time it is invoked; (1) 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); (2) the adversarial, protected input x^i_{ap}; (3) the adversarial, unprotected input x^i_{au}; (4) 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* $(\mathrm{tstate}^{i-1}, x^{j^i}_{hs}, x^{j^i}_{hp}, x^{j^i}_{hu}, x^i_{ap}, x^i_{au})$, *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^i_s, protected y^i_p and unprotected y^i_u 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^i_p and y^i_u. The overall view of E in the real process is just its view in the last round (i.e., i for which* $\mathrm{stop}^i = \mathrm{TRUE}$.*).*

- *The* ideal *process:*

$$\begin{aligned}\mathrm{EVIEW}^i_{\mathrm{ideal}} = \{&(\mathrm{istate}^i, x^i_{hs}, x^i_{hp}, x^i_{hu}) \leftarrow I(1^k, \mathrm{istate}^{i-1});\\ &(\mathrm{estate}^i, j^i, x^i_{ap}, x^i_{au}, \mathrm{stop}^i) \leftarrow E(1^k, \mathrm{EVIEW}^{i-1}_{\mathrm{ideal}}, x^i_{hp}, x^i_{hu});\\ &(\mathrm{astate}^i, y^i_s, y^i_p, y^i_u) \leftarrow \mathsf{Alg}(\mathrm{astate}^{i-1}, x^{j^i}_{hs}, x^{j^i}_{hp}, x^{j^i}_{hu}, x^i_{ap}, x^i_{au});\\ &(\mathrm{sstate}^i, \mathrm{ustate}^i, Y^i_p, Y^i_u, \mathrm{rep}^i) \leftarrow S_1^{U'(\mathrm{ustate}^{i-1})}\\ &(\mathrm{sstate}^{i-1}, \cdots, x^{j^i}_{hp}, x^{j^i}_{hu}, x^i_{ap}, x^i_{au}, y^i_p, y^i_u);\\ &(z^i_p, z^i_u) = \mathrm{rep}^i(Y^i_p, Y^i_u) + (1 - \mathrm{rep}^i)(y^i_p, y^i_u) :\\ &(\mathrm{estate}^i, z^i_p, z^i_u)\}\end{aligned}$$

$\mathrm{EVIEW}_{\mathrm{ideal}} = \mathrm{EVIEW}^i_{\mathrm{ideal}}$ *if* $\mathrm{stop}^i = \mathrm{TRUE}$.

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^i_{hs}, but given the non-secret outputs that Alg *produces when run all the inputs for round i, decides to either output the values (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 captured by having the indicator variable* rep^i *be a bit that determines whether y^i_p will be replaced with Y^i_p. In doing so, it is allowed to query oracle U'; moreover, U' saves its state as in the real experiment.*

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

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

  $\text{UVIEW}_{\text{ideal}} = \text{UVIEW}^i_{\text{ideal}}$ *if* $\text{stop}^i = \text{TRUE}$.

  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 an 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.*

3 New and Secure Outsourcing Algorithm of Modular Exponentiations

3.1 Security Model

Hohenberger and Lysyanskaya [33] first presented the so-called *two untrusted program model* for outsourcing exponentiations modulo a prime. In the two untrusted program model, the adversarial environment E writes the code for two (potentially different) programs $U' = (U'_1, U'_2)$. E then gives this software to T, advertising a functionality that U'_1 and U'_2 may or may not accurately compute, and T installs this 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 $\mathcal{A} = (E, U'_1, U'_2)$. Moreover, we assume that at most one of the programs U'_1 and U'_2 deviates from its advertised functionality on a non-negligible

fraction of the inputs, while we cannot know which one and security means that there is a simulator $\mathcal{S}$ for both. This is named as the one-malicious version of two untrusted program model (i.e., "one-malicious model" for the simplicity). 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.

In the security model [33], a subroutine named *Rand* is used in order to speed up the computations. The inputs for *Rand* are a prime p, a base $g \in \mathbb{Z}_p^*$, and possibly some other values, and the outputs for each invocation are a random, independent pair of the form $(b, g^b \mod p)$, where $b \in \mathbb{Z}_q$. There are two approaches to implement this functionality. One is for a trusted server to compute a table of random, independent pairs in advance and then load it into the memory of T. For each invocation of *Rand*, T just retrieves a new pair in the table (the table-lookup method).[1] The other is to apply the well-known preprocessing techniques. By far, the most promising preprocessing algorithm is the EBPV generator [40], which is secure against adaptive adversaries and runs in time $O(\log^2 n)$ for an n-bit exponent. On input a sufficiently large subset of truly random (k, g^k) pairs, EBPV generator outputs a pair (l, g^l) that is statistically close to the uniform distribution. Therefore, we argue that T can never control the output of the subroutine *Rand*, especially the value of l for both of the approaches.

3.2 Outsourcing Algorithm

In this section, we propose 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 the subroutine *Rand*. A requirement for **Exp** is that the adversary $\mathcal{A}$ cannot know any useful information about the inputs and outputs of **Exp**. Similar to [33], $U_i(x, y) \to y^x$ also denotes that U_i takes as inputs (x, y) and outputs $y^x \mod p$, 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 $u^q = 1 \mod p$ (for an arbitrary base u and an arbitrary power a). The output of **Exp** is $u^a \mod p$. Note that a may be secret or (honest/adversarial) protected and u may be (honest/adversarial) protected. Both of a and u are computationally blinded to U_1 and U_2.

To implement this functionality using U_1 and U_2, T firstly runs *Rand* twice to create two blinding pairs (α, g^α) and (β, g^β). We denote $v = g^\alpha \mod p$ and $\mu = g^\beta \mod p$.

Our trick is a more efficient solution to logically split u and a into random looking pieces that can be computed by U_1 and U_2. The first logical divisions are

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

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

[1] In most applications, the pair cannot be reused. For example, reusing such a pair in Schnorr signature will result in the secret key exposure of the signer.

The second logical divisions are

$$u^a = g^\beta g^\gamma w^a = g^\beta g^\gamma w^{k+l} = g^\beta g^\gamma w^k w^l,$$

where $l = a - k$.

Next, T runs *Rand* to obtain three pairs (t_1, g^{t_1}), (t_2, g^{t_2}), and (t_3, g^{t_3}).

T queries U_1 in random order as
$U_1(t_2/t_1, g^{t_1}) \to g^{t_2}$;
$U_1(\gamma/t_3, g^{t_3}) \to g^\gamma$;
$U_1(l, w) \to w^l$.

Similarly, T queries U_2 in random order as
$U_2(t_2/t_1, g^{t_1}) \to g^{t_2}$;
$U_2(\gamma/t_3, g^{t_3}) \to g^\gamma$;
$U_2(k, w) \to w^k$.

Finally, T checks 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, T outputs "error"; otherwise, T can compute $u^a = \mu g^\gamma w^k w^l$.

Remark 1. In the one-malicious model, the equation $U_1(\gamma/t_3, g^{t_3}) = U_2(\gamma/t_3, g^{t_3})$ implies both U_1 and U_2 produce the correct g^γ. Therefore, the partial computation result g^γ also plays the role of a test query. This is slightly different from the technique in [33] while it indeed improves the efficiency and checkability of the computations.

Remark 2. Trivially, the proposed algorithm **Exp** can be extend to the outsource-secure scalar multiplications on elliptic curves, i.e., aU for any $a \in \mathbb{Z}_q^*$.

3.3 Security Analysis

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 proof is similar to [33]. The correctness is trivial and we only focus on security. Let $\mathcal{A} = (E, U_1', U_2')$ be a PPT adversary that interacts with a PPT algorithm T in the one-malicious model.

Firstly, we prove Pair One $EVIEW_{real} \sim EVIEW_{ideal}$:

If the input (a, u) is anything other than honest, secret, then the simulator S_1 behaves the same way as in the real execution. If (a, u) is an honest, secret input, then the simulator S_1 behaves as follows: On receiving the input on round i, S_1 ignores it and instead makes three random queries of the form (α_j, β_j) to both U_1' and U_2'. S_1 randomly tests two outputs (i.e., $\beta_j^{\alpha_j}$) 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 (i.e., the output for ideal process is $(estate^i$, "error", $\varnothing)$). If no error is detected, S_1 checks the remaining two outputs. If all checks pass, S_1 outputs Y_p^i=$\varnothing$, Y_u^i=$\varnothing$, rep^i=0 (i.e., the output for ideal process is $(estate^i, y_p^i, y_u^i)$); otherwise, S_1 selects a random element r and outputs Y_p^i=r, Y_u^i=$\varnothing$, rep^i=1 (i.e., the output for

ideal process is $(estate^i, r, \varnothing)$). 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 honest in the round i, then $EVIEW^i_{real} \sim EVIEW^i_{ideal}$ (this is because $T^{(U_1', U_2')}$ perfectly executes **Exp** in the real experiment and S_1 simulates with the same outputs in the ideal experiment, i.e., rep^i=0). If one of (U_1', U_2') is dishonest in the round i, 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** is 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. In the ideal experiment, S_1 also simulates with a random value r. Thus, $EVIEW^i_{real} \sim EVIEW^i_{ideal}$ even when one of (U_1', U_2') is dishonest. By the hybrid argument, we conclude that $EVIEW_{real} \sim EVIEW_{ideal}$.

Secondly, we prove Pair Two $UVIEW_{real} \sim UVIEW_{ideal}$:

The simulator S_2 always behaves as follows: On receiving the input on round i, S_2 ignores it and instead makes three random queries of the form (α_j, β_j) 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 round i of the real experiment, T always re-randomizes its inputs to (U_1', U_2'). In the ideal experiment, S_2 always generates random, independent queries for (U_1', U_2'). Thus, for each round i we have $UVIEW^i_{real} \sim UVIEW^i_{ideal}$. 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. The proposed algorithm **Exp** makes 5 calls to *Rand* plus 7 modular multiplication (MM) and 3 modular inverse (MInv) in order to compute $u^a \mod p$ (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, respectively, where n is the bit of the a. On the other hand, it takes roughly $1.5n$ MM to compute $u^a \mod p$ by the square-and-multiply method. 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 all of the three queries that T makes. If U_1 (resp. U_2) fails during any execution of **Exp**, it will be detected with probability $\frac{2}{3}$. □

3.4 Comparison

We compare the proposed algorithm with Hohenberger-Lysyanskaya's algorithm in [33]. We denote by MM a modular multiplication, by MInv a modular inverse,

and by $\text{Rand}^{\text{Invoke}}$ an invocation of the subroutine *Rand*. We omit other operations such as modular additions in both algorithms. Table 1 presents the comparison of the efficiency and the checkability between Hohenberger-Lysyanskaya's algorithm and our proposed algorithm **Exp**.

Table 1. Comparison of the two algorithms

	Algorithm [33]	Algorithm **Exp**
MM	9	7
MInv	5	3
Invoke(*Rand*)	6	5
Invoke(U_1)	4	3
Invoke(U_2)	4	3
Checkability	$\frac{1}{2}$	$\frac{2}{3}$

Compared with Hohenberger-Lysyanskaya's algorithm, the proposed algorithm **Exp** is superior in both efficiency and checkability. More precisely, **Exp** requires only 7 MM, 3 MInv, 5 invocation of *Rand*, and 3 invocation of U_1 and U_2 for each modular exponentiation. Note that the modular exponentiation is the most basic operation in discrete-logarithm based cryptographic protocols, and millions of such computations may be outsourced to the server every day. Thus, our proposed algorithm can save huge of computational resources for both the outsourcer T and the servers U_1 and U_2.

4 Secure Outsourcing Algorithms for Encryption and Signatures

In this section, we propose two secure outsourcing algorithms for Cramer-Shoup encryption scheme [18] and Schnorr signature scheme [42].

4.1 Outsource-Secure Cramer-Shoup Encryptions

The proposed outsource-secure Cramer-Shoup encryption scheme 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 subroutine *Rand* and generates the ciphertext C as follows:

1. T runs *Rand* to obtain a pair $(k, r = g^k \mod p)$.
2. T firstly runs **Exp** to obtain $\mathbf{Exp}(k, W) \to s$, $\mathbf{Exp}(k, Z) \to t$ and then computes $e = mt$, and $h = H(r, s, e)$.
3. T runs **Exp** to obtain $\mathbf{Exp}(k, X) \to \alpha$, $\mathbf{Exp}(kh, Y) \to \beta$ and then computes $\gamma = \alpha\beta$.
4. T outputs the ciphertext $C = (r, s, e, \gamma)$.

– **Decryption:** On input the verification key y, the message m, and the signature $\sigma = (e, s)$, the outsourcer T runs the subroutine **Exp** and verifies the signature σ as follows:
1. T computes $h = H(r, s, e)$.
2. T runs **Exp** to obtain $\mathbf{Exp}(w, r) \to \psi_1$ and $\mathbf{Exp}(x + yh, r) \to \psi_2$.
3. If and only if $s = \psi_1$ and $\gamma = \psi_2$, T runs **Exp** to obtain $\mathbf{Exp}(z, r) \to t$ computes $m = et^{-1}$.
4. T outputs m.

Remark 3. We present a secure outsourcing algorithm for Cramer-Shoup encryption scheme CS1b. Compared with [33], we do not use a new subroutine *Rand*′ that produces a triple $(b, g^b \mod p, g'^b \mod p)$, while our algorithm requires one more invocation of **Exp** (only) for encryption. Trivially, we could present outsouce-secure Cramer-Shoup encryption scheme CS1a (running either *Rand* or *Rand*′).

4.2 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 $g^q = 1 \mod p$. Define a cryptographic secure hash function $H : \{0, 1\}^* \to \mathbb{Z}_p$. The system parameters are $SP = \{p, q, g, H\}$.
– **Key Generation:** On input 1^l, run the key generation algorithm to obtain the signing/verification key pair (x, y), here $y = g^{-x} \mod p$.
– **Signature Generation:** On input the singing key x and a message m, the outsourcer T runs the subroutine *Rand* and generates the signature σ as follows:
1. T runs *Rand* to obtain a pair $(k, r = g^k \mod p)$.
2. T computes $e = H(m||r)$ and $s = k + xe \mod q$.
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 **Exp** and verifies the signature σ as follows:
1. T runs **Exp** to obtain $\mathbf{Exp}(s, g) \to \psi_1$ and $\mathbf{Exp}(e, y) \to \psi_2$.
2. T computes $r' = \psi_1\psi_2 \mod p$ and $e' = H(m||r')$.
3. T outputs 1 if and only if $e' = e$.

Remark 4. The proposed outsource-secure Schnorr signature scheme is basically same as that in [33]. Note that the subroutine **Exp** is only used for the signature verification.

5 Outsource-Secure Algorithm of Simultaneous Modular Exponentiations

In this section, we focus on simultaneous modular exponentiations $u_1^a u_2^b \mod p$, which play an important role in many cryptographic primitives such as chameleon hashing [5,6,21,22,36,44] and trapdoor commitment [9,16,23,25,32]. Trivially, a simultaneous modular exponentiation can be carried out by invoking 2 modular exponentiations. This requires roughly $3n$ MM, where n is the bit of a and b. However, the computation cost is only $1.75n$ MM (i.e., roughly 1.17 modular exponentiation) if we use the Algorithm 14.88 of [37].

In the following, we propose an efficient outsource-secure algorithm of simultaneous modular exponentiations **SExp** in the one-malicious model.

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 $u_1^a u_2^b \mod p$.

Similarly, T firstly runs *Rand* twice to create two blinding pairs (α, g^α) and (β, g^β). We denote $v = g^\alpha \mod p$ and $\mu = g^\beta \mod p$.

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 $s = b - t$.

Next, T runs *Rand* to obtain three pairs (t_1, g^{t_1}), (t_2, g^{t_2}), and (t_3, g^{t_3}).

T queries 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, T queries U_2 in random order as
$U_2(t_2/t_1, g^{t_1}) \to g^{t_2}$;
$U_2(\gamma/t_3, g^{t_3}) \to g^\gamma$;
$U_2(l, w_1) \to w_1^l$;
$U_2(s, w_2) \to w_2^s$.

Finally, T checks 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, T outputs "error"; otherwise, T can compute $u_1^a u_2^b = \mu g^\gamma w_1^k w_1^l w_2^t w_2^s$.

Note that **SExp** requires only 10 MM, 4 MInv, 5 invocation of *Rand*, and 4 invocation of U_1 and U_2 for each modular exponentiation. Therefore, the computation cost of **SExp** is much less than that of double running **Exp**. Moreover, it is even comparable to that of outsourcing *one* modular exponentiation [33].

Table 2. Efficiency comparison for two algorithms

	Algorithm [33]	Algorithm **SExp**
MM	9	10
MInv	5	4
Invoke(*Rand*)	6	5
Invoke(U_1)	4	4
Invoke(U_2)	4	4
Checkability	$\frac{1}{2}$	$\frac{1}{2}$

Table 2 presents the comparison of the efficiency and the checkability between Hohenberger-Lysyanskaya's *Exp* algorithm and our proposed algorithm **SExp**.

Similar to theorem 3.2, 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****.*

6 Conclusions

In this paper, we propose two outsource-secure and efficient algorithms for modular exponentiations and simultaneous modular exponentiations, which are the most basic and expensive operations in many discrete-logarithm cryptosystems. Compared with the algorithm [33], the proposed algorithm is superior in both efficiency and checkability.

The security model of our outsourcing algorithms requires the outsourcer to interact with two non-colluding cloud servers (the same as [33]). Therefore, an interesting open problem is whether there is an efficient algorithm for secure outsourcing modular exponentiation using only one untrusted cloud sever.

Acknowledgments. We are grateful to the anonymous referees for their invaluable suggestions. This work is supported by the National Natural Science Foundation of China (Nos. 60970144 and 61100224), and China 111 Project (No. B08038). Besides, Lou's work was supported by US National Science Foundation under grant CNS-1155988.

References

1. Atallah, M.J., Frikken, K.B.: Securely outsourcing linear algebra computations. In: Proceedings of the 5th ACM Symposium on Information, Computer and Communications Security (CCS), pp. 48–59 (2010)
2. Abadi, M., Feigenbaum, J., Kilian, J.: On hiding information from an oracle. In: Proceedings of the 19th Annual ACM Symposium on Theory of Computing (STOC), pp. 195–203 (1987)

3. Atallah, M.J., Pantazopoulos, K.N., Rice, J.R., Spafford, E.H.: Secure outsourcing of scientific computations. Advances in Computers 54, 216–272 (2001)
4. Atallah, M.J., Li, J.: Secure outsourcing of sequence comparisons, International Journal of Information Security, 277–287 (2005)
5. Ateniese, G., de Medeiros, B.: Identity-Based Chameleon Hash and Applications. In: Juels, A. (ed.) FC 2004. LNCS, vol. 3110, pp. 164–180. Springer, Heidelberg (2004)
6. Ateniese, G., de Medeiros, B.: On the Key Exposure Problem in Chameleon Hashes. In: Blundo, C., Cimato, S. (eds.) SCN 2004. LNCS, vol. 3352, pp. 165–179. Springer, Heidelberg (2005)
7. Blanton, M.: Improved Conditional E-Payments. In: Bellovin, S.M., Gennaro, R., Keromytis, A.D., Yung, M. (eds.) ACNS 2008. LNCS, vol. 5037, pp. 188–206. Springer, Heidelberg (2008)
8. Benjamin, D., Atallah, M.J.: Private and cheating-free outsourcing of algebraic computations. In: Proceeding of the 6th Annual Conference on Privacy, Security and Trust (PST), pp. 240–245 (2008)
9. Brassard, G., Chaum, D., Crepeau, C.: Minimum disclosure proofs of knowledge. Journal of Computer and System Sciences 37(2), 156–189 (1988)
10. Beaver, D., Feigenbaum, J., Kilian, J., Rogaway, P.: Locally random reductions: Improvements and applications. Journal of Cryptology 10(1), 17–36 (1997)
11. Ben-Or, M., Goldwasser, S., Kilian, J., Wigderson, A.: Multi-prover interactive proofs: How to remove intractability assumptions. In: Proceedings of the ACM Symposium on Theory of Computing (STOC), pp. 113–131 (1988)
12. Benabbas, S., Gennaro, R., Vahlis, Y.: Verifiable Delegation of Computation over Large Datasets. In: Rogaway, P. (ed.) CRYPTO 2011. LNCS, vol. 6841, pp. 111–131. Springer, Heidelberg (2011)
13. Blum, M., Luby, M., Rubinfeld, R.: Program result checking against adaptive programs and in cryptographic settings. DIMACS Series in Discrete Mathematics and Theoretical Computer Science, pp. 107–118 (1991)
14. Blum, M., Luby, M., Rubinfeld, R.: Self-testing/correcting with applications to numerical problems. Journal of Computer and System Science, 549–595 (1993)
15. Boyko, V., Peinado, M., Venkatesan, R.: Speeding up Discrete Log and Factoring Based Schemes via Precomputations. In: Nyberg, K. (ed.) EUROCRYPT 1998. LNCS, vol. 1403, pp. 221–235. Springer, Heidelberg (1998)
16. Di Crescenzo, G., Ostrovsky, R.: On Concurrent Zero-Knowledge with Preprocessing (Extended Abstract). In: Wiener, M. (ed.) CRYPTO 1999. LNCS, vol. 1666, pp. 485–502. Springer, Heidelberg (1999)
17. Chaum, D., Pedersen, T.P.: Wallet Databases with Observers. In: Brickell, E.F. (ed.) CRYPTO 1992. LNCS, vol. 740, pp. 89–105. Springer, Heidelberg (1993)
18. Cramer, R., Shoup, V.: Design and analysis of practical public-key encryption schemes secure against adaptive chosen ciphertext attack. SIAM Journal of Computing 33, 167–226 (2003)
19. Carbunar, B., Tripunitara, M.: Conditional Payments for Computing Markets. In: Franklin, M.K., Hui, L.C.K., Wong, D.S. (eds.) CANS 2008. LNCS, vol. 5339, pp. 317–331. Springer, Heidelberg (2008)
20. Carbunar, B., Tripunitara, M.: Fair payments for outsourced computations. In: Proceedings of the 7th Annual IEEE Communications Society Conference on Sensor, Mesh and Ad Hoc Communications and Networks (SECON), pp. 529–537. IEEE (2010)

21. Chen, X., Zhang, F., Kim, K.: Chameleon Hashing Without Key Exposure. In: Zhang, K., Zheng, Y. (eds.) ISC 2004. LNCS, vol. 3225, pp. 87–98. Springer, Heidelberg (2004)
22. Chen, X., Zhang, F., Susilo, W., Mu, Y.: Efficient Generic On-Line/Off-Line Signatures Without Key Exposure. In: Katz, J., Yung, M. (eds.) ACNS 2007. LNCS, vol. 4521, pp. 18–30. Springer, Heidelberg (2007)
23. Fischlin, M., Fischlin, R.: Efficient Non-malleable Commitment Schemes. In: Bellare, M. (ed.) CRYPTO 2000. LNCS, vol. 1880, pp. 413–431. Springer, Heidelberg (2000)
24. Even, S., Goldreich, O., Micali, S.: On-line/Off-line digital signatures. Journal of Cryptology 9(1), 35–67 (1996)
25. Gennaro, R.: Multi-trapdoor Commitments and Their Applications to Proofs of Knowledge Secure Under Concurrent Man-in-the-Middle Attacks. In: Franklin, M. (ed.) CRYPTO 2004. LNCS, vol. 3152, pp. 220–236. Springer, Heidelberg (2004)
26. Green, M., Hohenberger, S., Waters, B.: Outsourcing the Decryption of ABE Ciphertexts. In: Proceedings of the 20th USENIX Conference on Security (2011), http://static.usenix.org/events/sec11/tech/full-papers/Green.pdf
27. Gennaro, R., Gentry, C., Parno, B.: Non-interactive Verifiable Computing: Outsourcing Computation to Untrusted Workers. In: Rabin, T. (ed.) CRYPTO 2010. LNCS, vol. 6223, pp. 465–482. Springer, Heidelberg (2010)
28. Goldwasser, S., Kalai, Y.T., Rothblum, G.N.: Delegating computation: interactive proofs for muggles. In: Proceedings of the ACM Symposium on the Theory of Computing (STOC), pp. 113–122 (2008)
29. Girault, M., Lefranc, D.: Server-Aided Verification: Theory and Practice. In: Roy, B. (ed.) ASIACRYPT 2005. LNCS, vol. 3788, pp. 605–623. Springer, Heidelberg (2005)
30. Goldwasser, S., Micali, S., Rackoff, C.: The knowledge complexity of interactive proof-systems. SIAM Journal on Computing 18(1), 186–208 (1989)
31. Golle, P., Mironov, I.: Uncheatable Distributed Computations. In: Naccache, D. (ed.) CT-RSA 2001. LNCS, vol. 2020, pp. 425–440. Springer, Heidelberg (2001)
32. Garay, J., MacKenzie, P., Yang, K.: Strengthening Zero-knowledge Protocols Using Signatures. In: Biham, E. (ed.) EUROCRYPT 2003. LNCS, vol. 2656, pp. 177–194. Springer, Heidelberg (2003)
33. Hohenberger, S., Lysyanskaya, A.: How to Securely Outsource Cryptographic Computations. In: Kilian, J. (ed.) TCC 2005. LNCS, vol. 3378, pp. 264–282. Springer, Heidelberg (2005), http://www.cs.jhu.edu/
34. Kilian, J.: A note on efficient zero-knowledge proofs and arguments. In: Proceedings of the ACM Symposium on Theory of Computing (STOC), pp. 723–732 (1992)
35. Kilian, J.: Improved Efficient Arguments. In: Coppersmith, D. (ed.) CRYPTO 1995. LNCS, vol. 963, pp. 311–324. Springer, Heidelberg (1995)
36. Krawczyk, H., Rabin, T.: Chameleon hashing and signatures. In: Proceeding of the 7th Annual Network and Distributed System Security Symposium (NDSS), pp. 143–154 (2000)
37. Menezes, A., van Oorschot, P., Vanstone, S.: Handbook of Applied Cryptography. CRC Press (1996)
38. Micali, S.: CS proofs. In: Proceedings of the 35th Annual Symposium on Foundations of Computer Science (FOCS), pp. 436–453 (1994)
39. Matsumoto, T., Kato, K., Imai, H.: Speeding up Secret Computations with Insecure Auxiliary Devices. In: Goldwasser, S. (ed.) CRYPTO 1988. LNCS, vol. 403, pp. 497–506. Springer, Heidelberg (1990)

40. Nguyen, P.Q., Shparlinski, I.E., Stern, J.: Distribution of modular sums and the security of server aided exponentiation. In: Proceedings of the Workshop on Comp. Number Theory and Crypt., pp. 1–16 (1999)
41. Parno, B., Raykova, M., Vaikuntanathan, V.: How to Delegate and Verify in Public: Verifiable Computation from Attribute-Based Encryption. In: Cramer, R. (ed.) TCC 2012. LNCS, vol. 7194, pp. 422–439. Springer, Heidelberg (2012)
42. Schnorr, C.P.: Efficient signature generation for smart cards. Journal of Cryptology 4(3), 239–252 (1991)
43. Shi, L., Carbunar, B., Sion, R.: Conditional E-Cash. In: Dietrich, S., Dhamija, R. (eds.) FC 2007 and USEC 2007. LNCS, vol. 4886, pp. 15–28. Springer, Heidelberg (2007)
44. Shamir, A., Tauman, Y.: Improved Online/Offline Signature Schemes. In: Kilian, J. (ed.) CRYPTO 2001. LNCS, vol. 2139, pp. 355–367. Springer, Heidelberg (2001)
45. Wang, C., Ren, K., Wang, J.: Secure and practical outsourcing of linear programming in cloud computing. In: Proceedings of the 30th IEEE International Conference on Computer Communications (INFOCOM), pp. 820–828 (2011)
46. Wu, W., Mu, Y., Susilo, W., Huang, X.: Server-Aided Verification Signatures: Definitions and New Constructions. In: Baek, J., Bao, F., Chen, K., Lai, X. (eds.) ProvSec 2008. LNCS, vol. 5324, pp. 141–155. Springer, Heidelberg (2008)