New Algorithms for Secure Outsourcing of Modular Exponentiations

Xiaofeng Chen, Jin Li, Jianfeng Ma, Qiang Tang, and Wenjing Lou, *Senior Member, IEEE*

Abstract—With the rapid development of cloud services, the techniques for securely outsourcing the prohibitively expensive computations to untrusted servers are getting more and more attention in the scientific community. Exponentiations modulo a large prime have been considered the most expensive operations in discrete-logarithm-based cryptographic protocols, and they may be burdensome for the resource-limited devices such as RFID tags or smartcards. Therefore, it is important to present an efficient method to securely outsource such operations to (untrusted) cloud servers. In this paper, we propose a new secure outsourcing algorithm for (variable-exponent, variable-base) exponentiation modulo a prime in the two untrusted program model. Compared with the state-of-the-art algorithm, the proposed algorithm is superior in both efficiency and checkability. Based on this algorithm, we show how to achieve outsource-secure Cramer-Shoup encryptions and Schnorr signatures. We then propose the first efficient outsource-secure algorithm for simultaneous modular exponentiations. Finally, we provide the experimental evaluation that demonstrates the efficiency and effectiveness of the proposed outsourcing algorithms and schemes.

Index Terms—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 this new computing environment is the so-called outsourcing paradigm, where the resource-constrained devices can outsource their computation workloads to cloud servers and enjoy the unlimited computation resources in a pay-per-use manner. As a result, the enterprises can avoid large capital outlays in hardware/software deployment and maintenance.

Despite the tremendous benefits, the outsourcing paradigm inevitably introduces some new security concerns and challenges [45], [50]. First, the cloud servers can only be assumed to be semi-trusted, while the computation tasks often contain some sensitive information that should not be exposed to the cloud servers. As such, 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. Note that fully homomorphic encryption could be a potential solution, but the existing schemes are not practical yet. Second, the semi-trusted cloud servers may return invalid results. For example, the servers might contain a software bug that will fail on a constant number of invocations. Moreover, the servers might decrease the amount of the computations due to financial incentives and then return computationally indistinguishable (invalid) results. Therefore, the second security challenge is the *checkability* of the outsourcing computation: the client should have the ability to detect any failures if the cloud servers misbehave. Obviously, the test procedure should never require some other complicated computations, otherwise the outourcing will become meaningless. It must be *far more* efficient than accomplishing the computation task itself.

The problem of secure outsourcing expensive computations has been well studied in the cryptography community. Chaum and Pedersen [18] first 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 [33] 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 [35] presented the first 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 for modular exponentiations in the one-malicious version of two untrusted program model. Compared with the state-of-the-art algorithm [35], the proposed algorithm is superior in both efficiency and checkability. Similar to [35], we also utilize this algorithm as

- *X. Chen is with the State Key Laboratory of Integrated Service Networks (ISN), Xidian University, China. E-mail: xfchen@xidian.edu.cn.*
- *J. Li is with the School of Computer Science and Educational Software, Guangzhou University, China. E-mail: jinli71@gmail.com.*
- *J. Ma is with the School of Computer Science and Technology, Xidian University, China. E-mail: jfma@mail.xidian.edu.cn.*
- *Q. Tang is with SnT, University of Luxembourg. E-mail: qiang.tang@uni.lu.*
- *W. Lou is with the Department of Computer Science, Virginia Polytechnic Institute and State University, USA. E-mail: wjlou@vt.edu.*

Manuscript received 12 Mar. 2013; revised 4 July 2013; accepted 6 July 2013. Date of publication 23 July 2013; date of current version 13 Aug. 2014. Recommended for acceptance by X. Liu.
For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Identifier below.
Digital Object Identifier no. 10.1109/TPDS.2013.180

a subroutine to achieve outsource-secure Cramer-Shoup encryptions and Schnorr signatures. Another contribution of this paper is that we propose the first outsource-secure and efficient algorithm for *simultaneous* modular exponentiations, whose efficiency is (surprisingly) comparable to that of outsourcing only *one* modular exponentiation in [35]. As an application, we present a secure outsourcing algorithm for chameleon signatures. To demonstrate the practical performances, we provide the experimental results for the proposed outsourcing algorithms and schemes.

The main differences between this paper and and the conference version in ESORICS [25] are as follows: Firstly, we present a more clearer explanation for one-malicious version of two untrusted program model in Section 3.1. Secondly, we describe how to extend the proposed algorithm **Exp** to outsource-secure scalar multiplications **SM** on elliptic curves in the Section 3.2, Also, we presented a concrete application of the algorithm **SExp**, e.g., secure outsourcing algorithm for chameleon signatures in Section 5.3. Finally, we provide a thorough experimental evaluation of the proposed outsourcing algorithms and cryptographic schemes in Section 6.

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. Recently, Blanton *et al.* proposed a more efficient scheme for secure outsourcing sequence comparisons [13]. 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.* [49] 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 [18] 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 [35] proposed the first outsource-secure algorithm for modular exponentiations based on the two previous approaches of precomputation [16], [27], [42], [46] and server-aided computation [10], [31], [41], [51]. Chevallier-Mames *et al.* [26] presented the first algorithm for secure delegation of elliptic-curve pairings based on an untrusted server model. Besides, the outsourcer could detect any failures with probability 1 if the server misbehaves. However, an obvious disadvantage of the algorithm is that the outsourcer should carry out some other expensive operations such as scalar multiplications and exponentiations.

Since the servers (or workers) are not trusted by the outsourcers, Golle and Mironov [33] 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], [21], [22], [47]. Besides, Gennaro *et al.* [29] 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], [14], [15], [30], [32], [36], [37], [40]. Gennaro *et al.* [29] 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 [29]. In 2011, Green *et al.* [28] proposed new methods for efficiently and securely outsourcing decryption of attribute-based encryption (ABE) ciphertexts. Based on this work, Parno *et al.* [43] 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. The experimental evaluation of the proposed algorithms is given in Section 6. Finally, conclusions will be made in Section 7.

2 Security Definition and Model

2.1 Definition of Outsource-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 [35].

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 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 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. $EVIEW_{real} \sim EVIEW_{ideal}$:*

     - *The view that the adversarial environment E obtains by participating in the following real process:*

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

     $EVIEW_{real} = EVIEW^i_{real}$ 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

     1. *the value of its $estate_i$ variable as a way of remembering what it did next time it is invoked;*
     2. *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);*
     3. *the adversarial, protected input x^i_{ap};*
     4. *the adversarial, unprotected input x^i_{au};*
     5. *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 $(tstate^{i-1}, x^{j^i}_{hs}, x^{j^i}_{hp}, x^{j^i}_{hu}, x^i_{ap}, x^i_{au})$, 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 $stop^i = TRUE$.).

     - *The ideal* process:

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

     $EVIEW_{ideal} = EVIEW^i_{ideal}$ 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^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. $UVIEW_{real} \sim UVIEW_{ideal}$:*

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

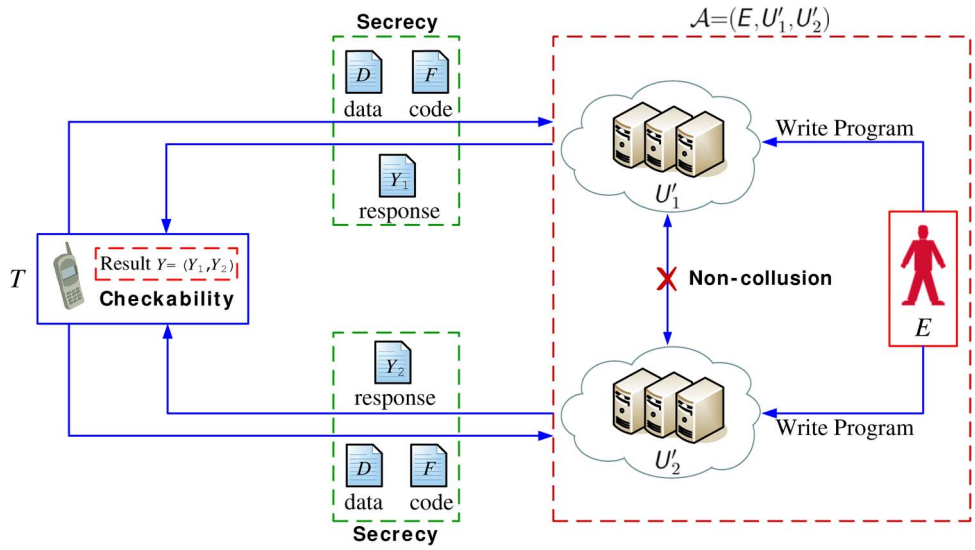


Fig. 1. One-malicious version of two untrusted program model.

- *The ideal process:*

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

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

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.*

Remark 1. It is worth noting that, depending on the β parameter, a secure outourcing algorithm may not provide 100 percent checkability (e.g., [35] and our algorithms). In practice, it is very likely that a client will run the outsourcing algorithm many times with the same servers. If a server cheats frequently, there is a high chance that it will be caught in some instances of the algorithm. Then, the client may seriously punish the servers when a cheating is detected. As a result, the server will not find the incentive to cheat in practice. Nevertheless, it is clear that a larger β is always better. However, there is a tradeoff between the efficiency (α) and chackability (β). For instance, a trivial way to improve the β value is to add a lot of dummy computations to check whether the servers have cheated. This will significantly reduce the efficiency because the servers need to perform a lot of additional computations.

2.2 Security Model

Hohenberger and Lysyanskaya [35] first presented the so-called *two untrusted program model* for outsourcing exponentiations modulo a prime. In this 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), as shown in Fig. 1.

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. Recently, Canetti, Riva, and Rothblum [19] introduced the so-called refereed delegation of computation model, where the outsourcer delegates the computation to $n \geq 2$ servers. In case the servers make

contradictory claims about the computation results, the outsourcer can engage in a protocol with each of the servers, at the end of which the outsourcer can efficiently determine the true claim under the assumption that at least one of the servers is honest (while the outsourcer does not know which is honest). Obviously, one-malicious model can be viewed as a special case of refereed delegation of computation model when the number of servers $n = 2$.

3 New and Secure Outsourcing Algorithm of Modular Exponentiations

In the construction from [35], 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 $(b, g^b \bmod p)$, 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 [42], 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.

3.1 Outsourcing Algorithm

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 [35], $U_i(x, y) \rightarrow y^x$ also denotes that U_i takes as inputs (x, y) and outputs $y^x \bmod p$, 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 $u^q = 1 \bmod p$ (for an arbitrary base u and an arbitrary power a). The output of **Exp** is $u^a \bmod p$. 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. The proposed algorithm **Exp** is given as follows:

1. 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 $v = g^\alpha \bmod p$ and $\mu = g^\beta \bmod p$.
2. The main 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

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

where $w = u/v \bmod p$ and $\gamma = a\alpha - \beta \bmod q$. The second logical divisions are

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

where $l = a - k \bmod q$.

3. Next, T runs $Rand$ to obtain three pairs (t_1, g^{t_1}), (t_2, g^{t_2}), and (t_3, g^{t_3}).
4. T queries U_1 in random order as

$U_1(t_2/t_1, g^{t_1}) \rightarrow g^{t_2}$;
$U_1(\gamma/t_3, g^{t_3}) \rightarrow g^\gamma$;
$U_1(l, w) \rightarrow w^l$. Similarly, T queries U_2 in random order as

$U_2(t_2/t_1, g^{t_1}) \rightarrow g^{t_2}$;
$U_2(\gamma/t_3, g^{t_3}) \rightarrow g^\gamma$;
$U_2(k, w) \rightarrow w^k$.

5. 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 2. 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 [35] while it indeed improves the efficiency and checkability of the computations.

Remark 3. The proposed algorithm **Exp** can also be extend to outsource-secure scalar multiplications **SM** on elliptic curves, i.e., aU for any $a \in \mathbb{Z}_q^*$ and any $U \in \mathbb{G}$, where $\mathbb{G}$ is a cyclic additive point group of an elliptic curve E defined over a finite field $\mathbb{GF}(q)$.

Trivially, the subroutine $Rand$ is defined as $Rand(G) \rightarrow (b, bG)$, where $b \in \mathbb{Z}_q$. T firstly runs $Rand$ twice to create two blinding pairs $(\alpha, \alpha G)$ and $(\beta, \beta G)$. The logical divisions are $aU = \beta G + \gamma G + kW + lW$, where $W = U - V$, $V = \alpha G$, $\gamma = a\alpha - \beta$, and $l = a - k$.

Next, T runs $Rand$ to obtain three pairs $(t_1, t_1 G)$, $(t_2, t_2 G)$, and $(t_3, t_3 G)$.

T queries U_1 in random order as

$U_1(t_2/t_1, t_1 G) \rightarrow t_2 G$;
$U_1(\gamma/t_3, t_3 G) \rightarrow \gamma G$;
$U_1(l, W) \rightarrow lW$.

Similarly, T queries U_2 in random order as

$U_2(t_2/t_1, t_1 G) \rightarrow t_2 G$;
$U_2(\gamma/t_3, t_3 G) \rightarrow \gamma G$;
$U_2(k, W) \rightarrow kW$.

Finally, T checks that both U_1 and U_2 produce the correct outputs, i.e., $t_2 G = U_1(t_2/t_1, t_1 G) = U_2(t_2/t_1, t_1 G)$ and $U_1(\gamma/t_3, t_3 G) = U_2(\gamma/t_3, t_3 G)$. If not, T outputs ''error''; otherwise, T can compute $aU = \beta G + \gamma G + kW + lW$.

3.2 Security Analysis

Theorem 3.1. *In the one-malicious model, the algorithms* $(T, (U_1, U_2))$ *are an outsource-secure implementation of* **Exp**,

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.

TABLE 1
Comparison of the Two Algorithms

	Algorithm [35]	Algorithm **Exp**
MM	9	7
MInv	5	3
Invoke($Rand$)	6	5
Invoke(U_1)	4	3
Invoke(U_2)	4	3
Checkability	$\frac{1}{2}$	$\frac{2}{3}$

where the input (a, u) may be honest, secret; or honest, protected; or adversarial, protected.

Proof. The proof is similar to [35]. 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'', $Y_u^i = \emptyset$, $rep^i = 1$ (i.e., the output for ideal process is $(estate^i, \text{''error''}, \emptyset)$). If no error is detected, S_1 checks the remaining two outputs. If all checks pass, S_1 outputs $Y_p^i = \emptyset$, $Y_u^i = \emptyset$, $rep^i = 0$ (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 $Y_p^i = r$, $Y_u^i =$, $rep^i = 1$ (i.e., the output for ideal process is $(estate^i, r, \emptyset)$). 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 $EVIEW_{real}^i \sim EVIEW_{ideal}^i$ (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, $EVIEW_{real}^i \sim EVIEW_{ideal}^i$ 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 $UVIEW_{real}^i \sim UVIEW_{ideal}^i$. By the hybrid argument, we conclude that $UVIEW_{real} \sim UVIEW_{ideal}$.

Theorem 3.2. *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 $u^a \bmod p$ (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 $u^a \bmod p$ 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 $\frac{2}{3}$.

3.3 Comparison

We compare the proposed algorithm with Hohenberger-Lysyanskaya's algorithm in [35]. We denote by MM a modular multiplication, by MInv a modular inverse, and by $Rand^{\text{Invoke}}$ 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**.

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 [20] and Schnorr signature scheme [46].

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),

TABLE 2
Efficiency Comparison for Two Algorithms

	Algorithm [35]	Algorithm **SExp**
MM	9	10
MInv	5	4
Invoke($Rand$)	6	5
Invoke(U_1)	4	4
Invoke(U_2)	4	4
Checkability	$\frac{1}{2}$	$\frac{1}{2}$

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 $(k, r = g^k \bmod p)$.
  2. T firstly runs **Exp** to obtain $\mathbf{Exp}(k, W) \rightarrow s$, $\mathbf{Exp}(k, Z) \rightarrow t$ and then computes $e = mt$, and $h = H(r, s, e)$.
  3. T runs **Exp** to obtain $\mathbf{Exp}(k, X) \rightarrow \alpha$, $\mathbf{Exp}(kh, Y) \rightarrow \beta$ and then computes $\gamma = \alpha\beta$.
  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 $\mathbf{Exp}(w, r) \rightarrow \psi_1$ and $\mathbf{Exp}(x + yh, r) \rightarrow \psi_2$.
  3. If and only if $s = \psi_1$ and $\gamma = \psi_2$, T' runs **Exp** to obtain $\mathbf{Exp}(z, r) \rightarrow t$ and then computes $m = et^{-1}$.
  4. T' outputs m.

Remark 4. We present a secure outsourcing algorithm for Cramer-Shoup encryption scheme CS1b. Compared with [35], we do not use a new subroutine $Rand'$ that produces a triple $(b, g^b \bmod p, g'^b \bmod p)$, 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 $g^q = 1 \bmod p$. Define a cryptographic secure hash function $H : \{0,1\}^* \rightarrow \mathbb{Z}_p$. 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 $y = g^{-x} \bmod p$.
- **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 $(k, r = g^k \bmod p)$.
  2. T computes $e = H(m||r)$ and $s = k + xe \bmod q$.
  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 $\mathbf{Exp}(s, g) \rightarrow \psi_1$ and $\mathbf{Exp}(e, y) \rightarrow \psi_2$.
  2. T' computes $r' = \psi_1\psi_2 \bmod p$ and $e' = H(m||r')$.
  3. T' outputs 1 if and only if $e' = e$.

Remark 5. The proposed outsource-secure Schnorr signature scheme is basically same as that in [35]. 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 $u_1^a u_2^b \bmod p$, which play an important role in many cryptographic primitives such as chameleon hashing [5], [6], [23], [24], [38], [48] and trapdoor commitment [9], [44], [34]. 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 simultaneous multiple exponentiation algorithm from Chapter 14 of [39].

5.1 Outsourcing Algorithm

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

1. T firstly runs $Rand$ twice to create two blinding pairs (α, g^α) and (β, g^β). We denote $v = g^\alpha \bmod p$ and $\mu = g^\beta \bmod p$.
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, T runs $Rand$ to obtain three pairs (t_1, g^{t_1}), (t_2, g^{t_2}), and (t_3, g^{t_3}).
4. T queries U_1 in random order as
$$U_1(t_2/t_1, g^{t_1}) \rightarrow g^{t_2};$$
$$U_1(\gamma/t_3, g^{t_3}) \rightarrow g^{\gamma};$$
$$U_1(k, w_1) \rightarrow w_1^k;$$
$$U_1(t, w_2) \rightarrow w_2^t.$$
Similarly, T queries U_2 in random order as
$$U_2(t_2/t_1, g^{t_1}) \rightarrow g^{t_2};$$
$$U_2(\gamma/t_3, g^{t_3}) \rightarrow g^{\gamma};$$

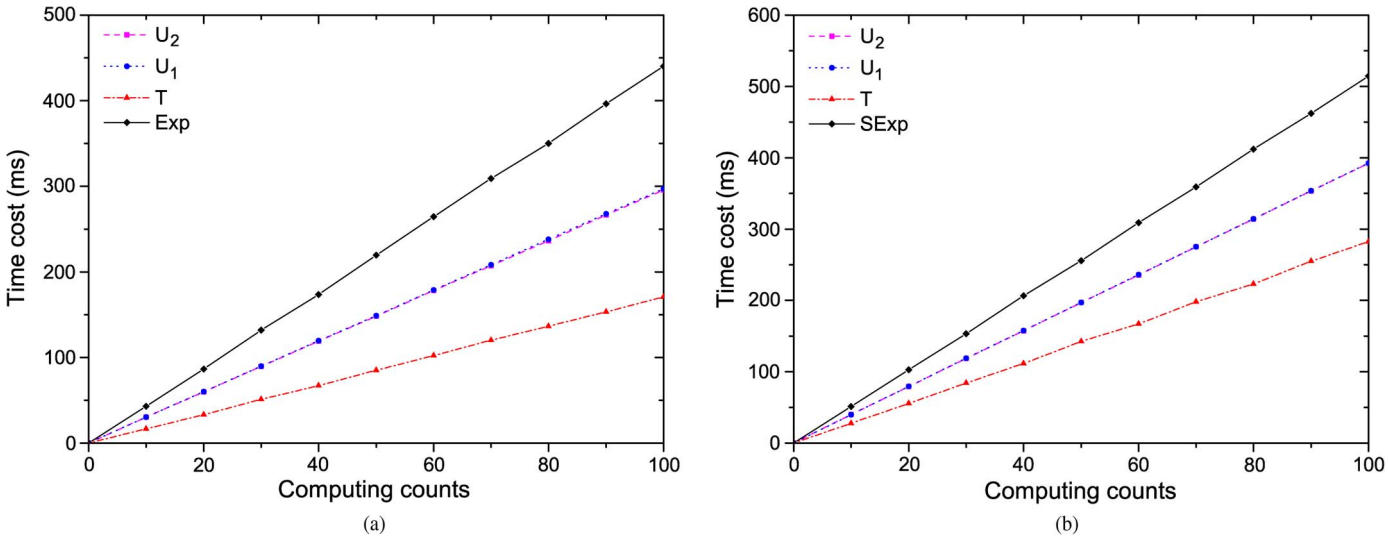


Fig. 2. Simulation for single modular exponentiations. (a) Single modular exponentiation. (b) Simultaneous modular exponentiations.

$U_2(l, w_1) \rightarrow w_1^l$;
$U_2(s, w_2) \rightarrow w_2^s$.

5. 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 $u_1^a u_2^b = \mu g^{\gamma} w_1^k w_1^l w_2^t w_2^s$.

Similar to Theorem 3.1 and 3.2, we can easily prove the following theorem:

Theorem 5.1. *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**.

5.2 Efficiency

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**. Surprisingly, it is even comparable to that of outsourcing *one* modular exponentiation [35]. Table 2 presents the comparison of the efficiency and the checkability between Hohenberger-Lysyanskaya's Exp algorithm and our proposed algorithm **SExp**.

5.3 Applications

In this section, we present a concrete application of the algorithm **SExp**, e.g., secure outsourcing algorithm for chameleon signatures.

Chameleon signatures, introduced by Krawczyk and Rabin [38], are based on well established hash-and-sign paradigm, where a chameleon hash function is used to compute the cryptographic message digest. A chameleon hash function is a trapdoor one-way hash function, which prevents everyone except the holder of the trapdoor information from computing the collisions for a randomly given input. Chameleon signatures simultaneously provide the properties of non-repudiation and non-transferability for the signed message as undeniable signatures [17] do, but the former allows for simpler and more efficient realization than the latter. In particular, chameleon signatures are non-interactive and less complicated. Besides, since the chameleon signatures are based on well established hash-and-sign paradigm, it provides more generic and flexible constructions.

In order to illustrate in details, we take a concrete chameleon signature scheme as an example, which employed for the Schnorr signature and the chameleon hashing based on the Pedersen trapdoor commitment [9], [44].

- **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 $g^q = 1 \bmod p$. Define a cryptographic secure hash function $H : \{0,1\}^* \rightarrow \mathbb{Z}_p$. 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_A, y_A), here $y_A = g^{-x_A} \bmod p$. Similarly, run the key generation algorithm to obtain the trapdoor/hash key pair is (x_B, y_B), here $y_B = g^{x_B} \bmod p$.

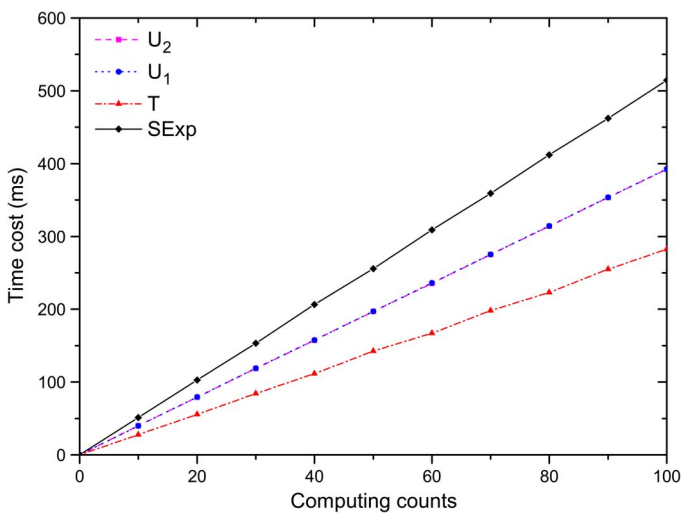


Fig. 3. Simulation for simultaneous modular exponentiations.

TABLE 3
Evaluation Time for Three Outsourcing Algorithms

	Hohenberger and Lysyanskaya's algorithm [35]	The proposed algorithm **Exp**	The proposed algorithm **SExp**
T	3.0386 ms	1.6845 ms	2.7869 ms
U_1	4.0641 ms	3.0479 ms	4.0639 ms
U_2	4.0642 ms	3.0480 ms	4.0645 ms

- **Signature Generation**: On input the singing key x_A and a message m, the outsourcer T generates the chameleon signature σ as follows:
  1. T randomly chooses an integer $a \in_R \mathbb{Z}_q^*$, and runs **SExp** to obtain $\mathbf{SExp}(a, m, g, y_B) \rightarrow \psi$.
  2. T runs $Rand$ to obtain a pair $(k, r = g^k \bmod p)$.
  3. T computes $e = H(\psi||r)$ and $s = k + x_A e \bmod q$.
  4. T outputs the signature $\sigma = (a, e, s)$.
- **Signature Verification**: On input the verification key y_A and the hash key y_B, the message m, and the signature $\sigma = (a, e, s)$, the outsourcer T' can verify the chameleon signature σ as follows:
  1. T' runs **SExp** to obtain $\mathbf{SExp}(a, m, g, y_B) \rightarrow \psi'$ and $\mathbf{SExp}(s, e, g, y_A) \rightarrow r'$.
  2. T' computes $e' = H(\psi'||r')$.
  3. T' outputs 1 if and only if $e' = e$.
- **Denial Protocol**: If and only if T is given a pair $(m', a') \neq (m, a)$, T could compute a new collision (m^*, a^*) for any message m^*, where $a^* = a + (m - m^*)(a - a')(m' - m)^{-1} \bmod q$.

6 Performance Evaluation

In this section, we provide an experimental evaluation of the proposed outsourcing algorithms and cryptographic schemes. Our experiment is simulated on two LINUX machines with Intel Core 4 processors running at 3.20 GHz and 4G memory (cloud server), and AMD Core 1 processor running at 1.60 GHz and 512 M memory (local user), respectively. The programming language is Python.

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

$$p = 8\text{df2a494 } 492276\text{aa } 3\text{d}25759\text{b } \text{b}06869\text{cb}$$
$$\text{eac}0\text{d}83\text{a } \text{fb}8\text{d}0\text{cf}7 \text{ cbb}8324\text{f } 0\text{d}7882\text{e}5$$
$$\text{d}0762\text{fc}5 \text{ b}7210\text{eaf } \text{c}2\text{e}9\text{adac } 32\text{ab}7\text{aac}$$
$$49693\text{dfb } \text{f}83724\text{c}2 \text{ ec}0736\text{ee } 31\text{c}80291$$
$$\text{q} = \text{c}773218\text{c } 737\text{ec}8\text{ee } 993\text{b}4\text{f}2\text{d } \text{ed}30\text{f}48\text{e } \text{dace}915\text{f}.$$

Firstly, we provide a simulation for both single and simultaneous modular exponentiations (i.e., **Exp** and **SExp**) as shown in Figs. 2 and 3, respectively. Note that a number of random pairs with the form of $(b, g^b \bmod p)$ should be prepared off-line, and we omit this in our simulation. Due to our outsourcing technique, a number of computations have been delegated to both U_1 and U_2 and thus the time cost for T is much smaller than that for directly computing modular exponentiation from scratch.

Next, we provide the evaluation time for the outsourcing algorithms proposed in [35] and this paper. From the result shown in Table 3, we can see that the outsource-secure single exponentiation algorithm **Exp** proposed in this paper is superior to Hohenberger and Lysyanskaya's algorithm [35] in efficiency. Moreover, the efficiency of the outsource-secure algorithm for simultaneous exponentiations **SExp** is comparable to that of outsourcing only one modular exponentiation in [35].

In addition, we present the evaluation time for outsourcing Cramer-Shoup encryptions, Schnorr signatures and chameleon signatures in Table 4. Obviously, compared with the original schemes, the time cost for T in outsourcing schemes is mostly reduced due to the algorithms **Exp** and **SExp**.

Remark 6. For all outsourcing algorithms, there is only one-round communication between the client and the servers (i.e. the client gives the server inputs, and the server returns results). For each instance of an outsourcing algorithm, the communication complexity is only a few kilobytes so that it will not downgrade the overall performance. Nevertheless, we leave it as a future work to have a more detailed study.

7 Conclusion

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 [35], 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 [35]). Therefore, an interesting open problem is whether there is an efficient algorithm for secure outsourcing modular exponentiation using only one untrusted cloud sever.

TABLE 4
Evaluation Time for Outsourcing Cryptographic Schemes

	Cramer-Shoup Encryption		Schnorr Signature	Chameleon Signature	
	Encryption	Decryption	Verification	Signature Generation	Verification
Outsourcing Scheme	0.0743 ms	1.0957 ms	0.0481 ms	0.0358 ms	0.0298 ms
Original Scheme	17.5671 ms	14.2154 ms	8.7945 ms	5.1524 ms	10.2631 ms

ACKNOWLEDGMENT

The authors are grateful to the anonymous referees for their invaluable suggestions. This work is supported by the National Natural Science Foundation of China (Nos. 61272455 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] 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] G. Ateniese and B. de Medeiros, ''Identity-Based Chameleon Hash and Applications,'' in *Proc. FC*, 2004, vol. LNCS 3110, pp. 164-180.

[6] G. Ateniese and B. de Medeiros, ''On the Key-Exposure Problem in Chameleon Hashes,'' in *Proc. SCN*, 2005, vol. LNCS 3352, pp. 165-179, Springer-Verlag: New York, NY, USA.

[7] M. Blanton, ''Improved Conditional E-Payments,'' in *Proc. ACNS*, 2008, vol. LNCS 5037, pp. 188-206, Springer-Verlag: New York, NY, USA.

[8] 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.

[9] G. Brassard, D. Chaum, and C. Crepeau, ''Minimum Disclosure Proofs of Knowledge,'' *J. Comput. Syst. Sci.*, vol. 37, no. 2, pp. 156-189, Oct. 1988.

[10] 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.

[11] 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.

[12] S. Benabbas, R. Gennaro, and Y. Vahlis, ''Verifiable Delegation of Computation Over Large Datasets,'' in *Proc. Crypto*, 2011, vol. LNCS 6841, pp. 111-131, Springer-Verlag: New York, NY, USA.

[13] 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, Springer-Verlag: New York, NY, USA.

[14] 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.

[15] 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.

[16] 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, Springer-Verlag: New York, NY, USA.

[17] D. Chaum and H. van Antwerpen, ''Undeniable Signatures,'' in *Proc. Crypto*, 1989, vol. LNCS 435, pp. 212-216, Springer-Verlag: New York, NY, USA.

[18] D. Chaum and T. Pedersen, ''Wallet Databases with Observers,'' in Proc. Crypto 1992, 1993, vol. LNCS 740, pp. 89-105, Springer-Verlag: New York, NY, USA.

[19] 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.

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

[21] B. Carbunar and M. Tripunitara, ''Conditioal Payments for Computing Markets,'' in *Proc. CANS*, 2008, vol. LNCS 5339, pp. 317-331, Springer-Verlag: New York, NY, USA.

[22] B. Carbunar and M. Tripunitara, ''Fair Payments for Outsourced Computations,'' in *Proc. SECON*, 2010, pp. 529-537.

[23] X. Chen, F. Zhang, and K. Kim, ''Chameleon Hashing without Key Exposure,'' in *Proc. ISC*, 2004, vol. LNCS 3225, pp. 87-98, Springer-Verlag: New York, NY, USA.

[24] X. Chen, F. Zhang, W. Susilo, and Y. Mu, ''Efficient Generic On-Line/Off-Line Signatures without Key Exposure,'' in *Proc. ACNS*, 2007, vol. LNCS 4521, pp. 18-30, Springer-Verlag: New York, NY, USA.

[25] X. Chen, J. Li, J. Ma, Q. Tang, and W. Lou, ''New Algorithms for Secure Outsourcing of Modular Exponentiations,'' in *Proc. ESORICS*, 2012, vol. LNCS 7459, pp. 541-556, Springer-Verlag: New York, NY, USA.

[26] 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, Springer-Verlag: New York, NY, USA.

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

[28] M. Green, S. Hohenberger, and B. Waters, ''Outsourcing the Decryption of ABE Ciphertexts,'' in *Proc. 20th USENIX Conf. Secur.*, 2011. [Online]. Available: http://static.usenix.org/events/sec11 /tech/full_papers/Green.pdf

[29] 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, Springer-Verlag: New York, NY, USA.

[30] 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.

[31] M. Girault and D. Lefranc, ''Server-Aided Verification: Theory and Practice,'' in *Proc. ASIACRYPT*, 2005, vol. LNCS 3788, pp. 605-623, Springer-Verlag: New York, NY, USA.

[32] 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.

[33] P. Golle and I. Mironov, ''Uncheatable Distributed Computations,'' in *Proc. CT-RSA*, 2001, vol. LNCS 2020, pp. 425-440, Springer-Verlag: New York, NY, USA.

[34] J. Garay, P. MacKenzie, and K. Yang, ''Strengthening Zero-Knowledge Protocols using Signatures,'' in *Proc. Eurocrypt*, 2003, vol. LNCS 2656, pp. 177-194, Springer-Verlag: New York, NY, USA.

[35] 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.

[36] J. Kilian, ''A Note on Efficient Zero-Knowledge Proofs and Arguments,'' in *Proc. ACM Symp. Theory Comput.*, 1992, pp. 723-732.

[37] J. Kilian, ''Improved Efficient Arguments (Preliminary Version),'' in *Proc. Crypto*, 1995, pp. 311-324, Springer-Verlag: New York, NY, USA.

[38] H. Krawczyk and T. Rabin, ''Chameleon Hashing and Signatures,'' in *Proc. NDSS*, 2000, pp. 143-154.

[39] A. Menezes, P. van Oorschot, and S. Vanstone, *Handbook of Applied Cryptography*. Boca Raton, FL, USA: CRC Press, 1996.

[40] S. Micali, ''CS Proofs,'' in *Proc. 35th Annu. Symp. Foundations Comput. Sci.*, 1994, pp. 436-453.

[41] 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, Springer-Verlag: New York, NY, USA.

[42] 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.

[43] 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, Springer-Verlag: New York, NY, USA.

[44] T. Pedersen, ''Non-Interactive and Information-Theoretical Secure Verifiable Secret Sharing,'' in *Proc. Crypto*, 1992, vol. LNCS 576, pp. 129-140, Springer-Verlag: New York, NY, USA.

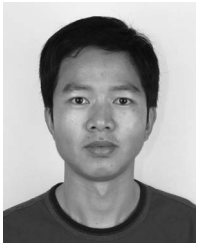
[45] K. Ren, C. Wang, and Q. Wang, ''Security Challenges for the Public Cloud,'' *IEEE Internet Comput.*, vol. 16, no. 1, pp. 69-73, Jan./Feb. 2012.

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

[47] L. Shi, B. Carbunar, and R. Sion, ''Conditional E-Cash,'' in *Proc. FC*, 2007, vol. LNCS 4886, pp. 15-28, Springer-Verlag: New York, NY, USA.

[48] A. Shamir and Y. Tauman, ''Improved Online/Offline Signature Schemes,'' in *Proc. Crypto*, 2001, vol. LNCS 2139, pp. 355-367, Springer-Verlag: New York, NY, USA.
[49] 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.
[50] 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.
[51] 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, Springer-Verlag: New York, NY, USA.

Xiaofeng Chen received the BS and MS degrees on mathematics from Northwest University, Xi'an, China, in 1998 and 2000, respectively, and the PhD degree in cryptography in Xidian University, Xi'an, China, in 2003. Currently, he is a Professor at the School of Telecommunications Engineering, Xidian University. His research interests include public key cryptography, financial cryptography, and cloud computing security.



Jin Li received the BS degree in mathematics from Southwest University, Chongqing, China, in 2002, the MS degree in mathematics from Sun Yat-sen University, Guangzhou, China, in 2004, and the PhD degree in information security from Sun Yat-sen University in 2007. Currently, he works at Guangzhou University, Guangzhou, China. His research interests include applied cryptography and security in cloud computing (secure outsourcing computation and cloud storage).

Jianfeng Ma received the BS degree in mathematics from Shaanxi Normal University, Xi'an, China, in 1985, and the ME and PhD degrees in computer software and communications engineering from Xidian University, Xi'an, China, in 1988 and 1995, respectively. From 1999 to 2001, he was with Nanyang Technological University of Singapore as a Research Fellow. Now, he is a Professor in the School of Computer Science at Xidian University. His current research interests include distributed systems, computer networks, and information and network security.

Qiang Tang received the BS degree from Yantai University, Yantai, China, in 1999, the MS degree from Peking University, Beijing, China, in 2002, and the PhD degree in information security and cryptography from Royal Holloway, University of London, London, UK, in 2007. Currently, he is a Research Scientist in the SnT research center at the University of Luxembourg, Walferdange, Luxembourg. His research interests include applied cryptography and privacy-preserving data engineering.

Wenjing Lou received the BS and MS degrees in computer science and engineering at Xi'an Jiaotong University, Xi'an, China, the MASc degree in computer communications at the Nanyang Technological University, Singapore, and the PhD in electrical and computer engineering at the University of Florida, Gainesville, USA. She is now an Associate Professor in the Computer Science Department at Virginia Polytechnic Institute and State University, Blacksburg, USA. She is a Senior Member of the IEEE.