Revisiting Single-server Algorithms for Outsourcing Modular Exponentiation

Jothi Rangasamy and Lakshmi Kuppusamy

Society for Electronic Transactions and Security (SETS), Chennai, India
{jothiram,lakshdev}@setsindia.net

Abstract. We investigate the problem of securely outsourcing modular exponentiations to a single, malicious computational resource. We revisit recently proposed schemes using single server and analyse them against two fundamental security properties, namely privacy of inputs and verifiability of outputs. Interestingly, we observe that the chosen schemes do not appear to meet both the security properties. In fact we present a simple polynomial-time attack on each algorithm, allowing the malicious server either to recover a secret input or to convincingly fool the client with wrong outputs. Then we provide a fix to the identified problem in the **ExpSOS** scheme. With our fix and without pre-processing, the improved scheme becomes the best to-date outsourcing scheme for single-server case. Finally we present the *first precomputation-free single-server algorithm*, π**ExpSOS** for *simultaneous* exponentiations, thereby solving an important problem formulated in [6].

1 Introduction

The problem of securely offloading cryptographic computations from a (comparatively) weak device to a more powerful device has been considered since many years [1] but the need for such a solution has been increasing rapidly [10][9][8][11]. A low-cost RFID tag is a natural example as it has limited computing resources but will benefit from running cryptographic protocols [18]. Proliferation of the usage of mobile applications adds one more scenario wherein outsourcing resource-consuming cryptographic tasks to a third-party is desirable.

The growing utilisation of cloud services such as Dropbox, Google and Amazon Cloud Drives has raised concerns about the availability and integrity of the data being handled and stored. By using cryptographic primitives such as provable data possession [2] and proofs of retrievability [3], these service providers could convince their clients that the actual data given by clients has been retrieved entirely. However, during this process, the clients have to engage in performing computationally-intensive operations to verify the claims of their storage providers and this is not practically viable for many devices in use today. Among complex cryptographic operations, modular exponentiation is invariably the predominant and core operation; that is, to compute $u^a \bmod p$ with a variable base u, a variable exponent a and a prime or RSA modulus p. In this paper, our goal is to inspect the recently proposed single-server outsourcing algorithms for modular exponentiation.

1.1 Related Work

The problem of secure delegation of crypto computations to (untrusted) helpers has been considered in various contexts[1,10,9,8,11,17,24]. In particular, the idea of secure delegation of modular exponentiation can be attributed to the work of Schnorr [21,22] as he was the first to propose speeding up modular exponentiations in cryptography. However it has not received formal treatment until Hohenberger and Lysyanskaya [12] developed a formal security framework for secure outsourcing of cryptographic computations to untrusted servers in 2005. In secure outsourcing scenario, preserving the secrecy of the inputs and/or outputs is vital. Hence, in Hohenberger-Lysyanskaya formalism, secrecy is the first notion an outsourcing algorithm should aim to satisfy. The second security notion, namely verifiability addresses the correctness of the output of the powerful helper/server. Hohenberger and Lysyanskaya also presented a scheme for outsourcing modular exponentiation to two non-colluding servers. This approach was improved in [6] and further in [15] with better verifiability results.

Designing an efficient algorithm using single untrusted (cloud) server for securely outsourcing (multi-)modular exponentiation has been a perennial problem. Towards solving this, Wang *et al.* presented (at ESORICS 2014) an efficient protocol to outsource modular exponentiation to a *single* untrusted server [23]. They also presented a generic protocol for outsourcing multi-exponentiations to a single server. However Chevalier*et al.* [7] presented a lattice-based attack on Wang *et al.* scheme recovering the secret exponent. Independently, Kiraz and Uzunkol designed an outsourcing scheme but requires an additional sub algorithm[14]. In 2017, Cai *et al.* [5] proposed a new scheme using redundant inputs but with increased communication complexity undesirably. Recently, Li *et al.*[16] have come up with a novel approach of using logical divisions twice for the given inputs. Then Zhou *et al.*[25] proposed a new scheme, which they call **ExpSOS**, using special ring structure of $\mathbb{Z}_N$ with the goal of eliminating the (de facto) preprocessing and achieving near-full verifiability simultaneously. To the best of our knowledge, **ExpSOS** is the only single-server algorithm which does not require resource-demanding pre-processing techniques such as *Rand* in [4].

1.2 Our Contributions

Contributions of the paper is two-fold. We first present practical attacks on three recent single-server algorithms and then we resolve the issue with the **ExpSOS** scheme, making it the best to-date scheme for this purpose.

First we show that the **CExp** scheme due to Li *et al.*[16] is unfortunately *zero* verifiable, instead of the authors' claim of satisfying the full verifibility. We demonstrate how to manipulate the outputs so that the delegator can be tricked by 100% into accepting false outputs .

Secondly, we note that the **SgExp** scheme proposed by Cai *et al.* [5] does not provide the claimed verifiability guarantee. The idea behind this tricky attack is to classify the queries to the untrusted (cloud) server into two categories because exponents in a set of queries need to be powers of two. This makes the scheme unfortunately *totally* unverifiable, instead of having the verifiability probability $1 - \frac{1}{n^2}$ which is ≈ 0.99996 for $n = 160$. The result is applicable to the **SmExp** scheme proposed by Cai *et al.* for the case of simultaneous exponentiation.

Thirdly, we describe an attack on the **ExpSOS** scheme of Zhou *et al.*[25] on its second invocation with the same secret exponent. The scheme will leak the exponent if it is used again, invalidating the claimed secrecy guarantee. The demonstrated attack extends to the other versions of the **ExpSOS** scheme in [25].

Our last and main contribution are new single-server algorithms for secure outsourcing of single and simultaneous modular exponentiations. For single exponentiation case, our algorithm is obtained by modifying **ExpSOS**. The modified scheme, which we call **MExpSOS** becomes the most efficient and simple scheme available in the related literature in addition to eliminating the *memory requirement* and the substantial *computational cost* of the precomputation step. Finally, we present an elegant algorithm, π**ExpSOS** extending **MExpSOS** for simultaneous multiplications. The π**ExpSOS** algorithm is near error-free and preprocessing-free and hence is the first of its kind in the literature. Our observations are summarised in Table 1.

Algorithm	Secrecy	verifibility Probability	Pre-processing Required
SgExp, SmExp [5]	Yes	0	Yes
CExp [16]	Yes	0	Yes
Kiraz-Uzunkol [14]	Yes	< 1	Yes
ExpSOS[25]	No	≈ 1	No
MExpSOS, π**ExpSOS**	Yes	≈ 1	No

Table 1. Single server based outsourcing algorithms and their properties

Outline. The paper is organised as follows. The Hohenberger-Lysyanskaya security model is recalled in Section 2. In Section 3 we observe weaknesses in recent single-server based outsourcing schemes. In Section 4 we propose a re-designed **ExpSOS** scheme and prove that it satisfies security notions: secrecy and verifiability and Section 5 concludes the paper.

2 Security Definitions

The first formal treatment for the problem of outsourcing cryptographic computations from a weak client to a powerful server was due to Hohenberger and Lysyanskaya [12]. The security model is useful in checking the privacy, efficiency and verifiability probability when outsourcing the task. In this section, we reproduce the security definitions of the Hohenberger-Lysyanskaya framework.

Adversarial behaviour. Assume that an algorithm Alg is run by two parties: a computationally weak and trusted party $\mathcal{C}$ (i.e., a client) who invokes a computationally powerful and untrusted party $\mathcal{U}$ through oracle queries. An outsource-secure implementation of an algorithm $\mathsf{Alg} = \mathcal{C}^{\mathcal{U}}$ is specified by $(\mathcal{C},\mathcal{U})$. where $\mathcal{C}$ carries out the tasks by invoking $\mathcal{U}$.

Hohenberger and Lysyanskaya modelled adversary $\mathcal{A} = (\mathcal{E},\mathcal{U}^{'})$ and its behaviour in two parts: (i) the adversarial environment $\mathcal{E}$ simulated to send/submit inputs to Alg; (ii) a malicious oracle $\mathcal{U}^{'}$ simulated to mimic $\mathcal{U}$. $\mathcal{E}$ and $\mathcal{U}^{'}$ can establish a direct communication channel only before agreeing on a joint initial strategy after which the only way they can communicate is by passing the messages through a channel re-directed/monitored by $\mathcal{C}$.

Input/output specifications. The following are the forms of information the algorithm's input/output may have:

Secret information possessed by $\mathcal{C}$;
Protected information known to both $\mathcal{C}$ and $\mathcal{E}$ but unknown to $\mathcal{U}^{'}$. This protected information is categorized depending upon the honest or adversarial generation of inputs;
Unprotected information known to $\mathcal{C}$, $\mathcal{E}$ and $\mathcal{U}^{'}$.

Definition 1. (Algorithm with IO-outsource) *The outsource algorithm* Alg *obeys the input/output specification if it accepts five inputs and produces three outputs. The honest entity generates the first three inputs and the last two inputs are generated by the environment $\mathcal{E}$. The first three inputs can be further classified based on the information about them available to the adversary $\mathcal{A} = (\mathcal{E},\mathcal{U}^{'})$. The first input is the honest, secret input which is unknown to both $\mathcal{E}$ and $\mathcal{U}^{'}$. The second input is the honest, protected input which may be known by $\mathcal{E}$, but is protected from $\mathcal{U}^{'}$. The third input is the honest, unprotected input which may be known by both $\mathcal{E}$ and $\mathcal{U}^{'}$. The fourth input is the adversarial, protected input which may be known by $\mathcal{E}$, but is protected from $\mathcal{U}^{'}$. The fifth input is the adversarial, protected input which may be known by $\mathcal{E}$, but is protected from $\mathcal{U}^{'}$. Similarly, the first, second and third outputs are called secret, protected and unprotected outputs respectively.*

2.1 Outsource-security definitions

The following are the security requirements an outsource algorithm should satisfy:

– *Secrecy.* It should be ensured that the malicious environment $\mathcal{E}$ should not learn secret inputs and outputs of the algorithm Alg, although there exist a joint initial strategy between $\mathcal{E}$ and the oracle $\mathcal{U}'$. In the formal definition, it is assumed that a simulator $\mathcal{S}_1$ exists without having access to the secret inputs and simulates the view of $\mathcal{E}$.

– *verifiability*. The malicious oracle $\mathcal{U}'$ does not gain any knowledge about the inputs to Alg even when it mimics the behaviour of $\mathcal{U}$ to $\mathcal{C}$. In the formal definition it is assumed that a simulator $\mathcal{S}_2$ exists without having access to the secret/protected inputs and simulates the view of $\mathcal{U}'$.

The following Definitions 2, 3, 4 and 5 are reproduced from [12].

Definition 2. (Outsource-security) *[12] A pair of algorithms ($\mathcal{C}$,$\mathcal{U}$) is said to be an outsource-secure implementation of an algorithm* Alg *with IO-outsource if:*

Correctness $\mathcal{C}^{\mathcal{U}}$ *is a correct implementation of* Alg.

Security *For all probabilistic polynomial-time adversaries $\mathcal{A} = (\mathcal{E},\mathcal{U}')$, there exist probabilistic expected polynomial-time simulators $(\mathcal{S}_1, \mathcal{S}_2)$ such that the following pairs of random variables are computationally indistinguishable.*

Pair One *($\mathcal{E}$ learns nothing):* $\text{EVIEW}_{real} \sim \text{EVIEW}_{ideal}$.

The real process: This process proceeds in rounds. Assume that the honestly generated inputs are chosen by a process I. The view that the adversarial environment obtains by participating in the following process:

$$\begin{aligned}\text{EVIEW}^i_{real} = \{\left(istate^i, x^i_{hs}, x^i_{hp}, x^i_{hu}\right) \leftarrow I\left(1^k, istate^{i-1}\right);\\ \left(estate^i, j^i, x^i_{ap}, x^i_{au}, stop^i\right) \leftarrow \text{E}\left(1^k, \text{EVIEW}^{i-1}_{real}, x^i_{hp}, x^i_{hu}\right);\\ \left(tstate^i, ustate^i, y^i_s, y^i_p, y^i_u\right) \leftarrow \mathcal{C}^{\mathcal{U}'(ustate^{i-1})}\left(tstate^{i-1}, x^{j^i}_{hs}, x^{j^i}_{hp}, x^{j^i}_{hu}, x^i_{ap}, x^i_{au}\right):\\ \left(estate^i, y^i_p, y^i_u\right)\}\end{aligned}$$

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

In round i, The adversarial environment does not have access to the honest inputs $(x^i_{hs}, x^i_{hp}, x^i_{hu})$ that are picked using an honest, stateful process I. The environment based on its view from last round, chooses the value of its $estate_i$ variable that is used to recall what it did next time it is invoked. Then, among the previously generated honest inputs, the environment chooses a input vector $(x^{j^i}_{hs}, x^{j^i}_{hp}, x^{j^i}_{hu})$ to give it to $\mathcal{C}^{\mathcal{U}'}$. Observe that the environment can specify the index j^i of the inputs but not the values. The environment also chooses the adversarial protected and unprotected input x^i_{ap} and x^i_{au} respectively. It also chooses the boolean variable $stop^i$ that determines whether round i is the last round in this process.
Then, $\mathcal{C}^{\mathcal{U}'}$ is run on 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 $\mathcal{C}$'s previously saved state . The algorithm produces a new state $tstate^i$ for $\mathcal{C}$ along with the secret y^i_s, protected y^i_p and unprotected y^i_u outputs. The oracle $\mathcal{U}'$ is given $ustate^{i-1}$ as input and the current state in saved in $ustate^i$. 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 the environment in the real process is just its

view in the last round.

The ideal process:

$$\begin{aligned}\mathrm{EVIEW}^i_{ideal} = \{ &\left(istate^i, x^i_{hs}, x^i_{hp}, x^i_{hu}\right) \leftarrow I\left(1^k, istate^{i-1}\right);\\ &\left(estate^i, j^i, x^i_{ap}, x^i_{au}, stop^i\right) \leftarrow \mathrm{E}\left(1^k, \mathrm{EVIEW}^{i-1}_{ideal}, x^i_{hp}, x^i_{hu}\right);\\ &\left(astate^i, y^i_s, y^i_p, y^i_u\right) \leftarrow \mathsf{Alg}\left(astate^{i-1}, x^{j^i}_{hs}, x^{j^i}_{hp}, x^{j^i}_{hu}, x^i_{ap}, x^i_{au}\right);\\ &\left(sstate^i, ustate^i, Y^i_p, Y^i_u, replace^i\right) \leftarrow \mathcal{S}_1^{\mathcal{U}'(ustate^{i-1})}\left(sstate^{i-1}, x^{j^i}_{hp}, x^{j^i}_{hu}, x^i_{ap}, x^i_{au}, y^i_p, y^i_u\right);\\ &\left(z^i_p, z^i_u\right) = replace^i\left(Y^i_p, Y^i_u\right) + \left(1 - replace^i\right)\left(y^i_p, y^i_u\right):\\ &\left(estate^i, z^i_p, z^i_u\right)\}\end{aligned}$$

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

This process also proceeds in rounds. The secret input x^i_{hs} is hidden from the stateful simulator $\mathcal{S}_1$. But, the non-secret inputs produced by the algorithm that is run on all inputs of round i is given to $\mathcal{S}_1$. Now, $\mathcal{S}_1$ decides whether to output the values (y^i_p, y^i_u) generated by the algorithm Alg *or replace them with some other values (Y^i_p, Y^i_u). This replacement is captured using the indicator variable $replace^i \in \{0,1\}$. The simulator is allowed to query the oracle $\mathcal{U}'$ which saves its state as in the real experiment.*

Pair two *($\mathcal{U}'$ learns nothing):* $\mathrm{UVIEW}_{real} \sim \mathrm{UVIEW}_{ideal}$.

The view that the untrusted entity $\mathcal{U}'$ obtains by participating in the real process is described in pair one. $\mathrm{UVIEW}_{real} = ustate^i if\, stop^i = \mathrm{TRUE}$. *The ideal process:*

$$\begin{aligned}\mathrm{UVIEW}^i_{ideal} = \{ &\left(istate^i, x^i_{hs}, x^i_{hp}, x^i_{hu}\right) \leftarrow I\left(1^k, istate^{i-1}\right);\\ &\left(estate^i, j^i, x^i_{ap}, x^i_{au}, stop^i\right) \leftarrow \mathrm{E}\left(1^k, estate^{i-1}, x^i_{hp}, x^i_{hu}, y^{i-1}_p, y^{i-1}_u\right);\\ &\left(astate^i, y^i_s, y^i_p, y^i_u\right) \leftarrow \mathsf{Alg}\left(astate^{i-1}, x^{j^i}_{hs}, x^{j^i}_{hp}, x^{j^i}_{hu}, x^i_{ap}, x^i_{au}\right);\\ &\left(sstate^i, ustate^i\right) \leftarrow \mathcal{S}_2^{\mathcal{U}'(ustate^{i-1})}\left(sstate^{i-1}, x^{j^i}_{hu}, x^i_{au}\right);\\ &\left(ustate^i\right)\}\end{aligned}$$

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

In the ideal process, the stateful simulator $\mathcal{S}_2$ is given with only the unprotected inputs (x^i_{hu}, x^i_{au}), queries $\mathcal{U}'$. As before, $\mathcal{U}'$ may maintain state.

Definition 3. ($\alpha-$efficient, secure outsourcing)*[12] A pair of algorithms ($\mathcal{C}$,$\mathcal{U}$) is said to be an $\alpha-$efficient implementation of an algorithm* Alg *if ($\mathcal{C}$,$\mathcal{U}$) is an outsource secure implementation of algorithm* Alg *and for all inputs x, the running time of $\mathcal{C}$ is $\leq$ an $\alpha-$ multiplicative factor of the running time of* $\mathsf{Alg}(x)$

Definition 4. ($\beta-$verifiable , secure outsourcing) *[12] A pair of algorithms ($\mathcal{C}$,$\mathcal{U}$) is a $\beta-$verifiable implementation of an algorithm* Alg *if ($\mathcal{C}$,$\mathcal{U}$) is an outsource secure implementation of algorithm* Alg *and for all inputs x, if $\mathcal{U}'$ deviates* *from its advertised functionality during the execution of $\mathcal{C}^{\mathcal{U}'}(x)$, $\mathcal{C}$ will detect the error with probability $\geq \beta$*

Definition 5. **($(\alpha, \beta)-$outsource-security)** *[12]A pair of algorithms ($\mathcal{C}$,$\mathcal{U}$) is said to be an $(\alpha, \beta)-$outsource-secure implementation of an algorithm* Alg *if they are both $\alpha-$efficient and $\beta-$checkable.*

3 On Recent Single-server Outsourcing Schemes

This section presents security issues with three single-server algorithms proposed in 2017; They are 1) Li *et al.* scheme (**CExp**), 2) Cai *et al.* scheme (**SgExp**) and 3) Zhou *et al.* scheme (**ExpSOS**).

3.1 Li *et al.* Scheme (CExp) and Its Weakness

First, we present Li *et al.* scheme briefly and then show its security weakness of not achieving full verifiability as claimed by the authors.

CExp Algorithm We use the same notations followed by Li *et al.* [16] to describe their outsourcing algorithm. Let $N = pq$ for two large primes p and q. Let CExp be an algorithm which outputs $u^d \bmod N$ upon accepting $u \in \mathbb{Z}_N^*$ and $d \in \mathbb{Z}_{\phi(N)}^*$ as inputs. The assumption is that the inputs u and d are secret or (honest/adversarial) protected. Hence the inputs need to be computationally blinded (masked) by the delegator $\mathcal{C}$ before passing them to the untrusted server $\mathcal{U}$.

Masking the Inputs. To mask the inputs, the algorithm CExp uses preprocessing technique RandN for efficient generation of pairs of the form $(x, x^e \bmod N)$ for a fixed e. (For more details about the description, analysis and efficiency of these pre-processing techniques, please refer to [4, Section 2] and [20].)

The client $\mathcal{C}$ runs RandN to generate four pairs $(g_1, g_1^e), (g_2, g_2^e), (g_3, g_3^e), (g_4, g_4^e)$. Let $v_1 = g_1^e \bmod N, w_1 = g_2^e \bmod N, v_2 = g_3^e \bmod N$, and $w_2 = g_4^e \bmod N$. To logically split base u and exponent d into random looking pieces, the first logical divisions are done as follows:

$$u^d = v_1 c_1^{r_1} w_1 w^{\ell_1} (w^{k_1} g_2)^{t_1}, \quad (1)$$

where $c_1 = g_1/g_2, r_1 = d - e, w = u/g_1, t_1 = r_1 - e$ and $d = \ell_1 + k_1 t_1$. For second logical divisions, $\mathcal{C}$ computes an integer a such that $ad \equiv 1 \bmod \phi(N)$. Then $\mathcal{C}$ computes $r^a \bmod N$ for a randomly chosen $r \in \{2, 10\}$ and sets $u' = u \cdot r^a$.

$$(u')^d = v_2 c_2^{r_1} w_2 (w')^{\ell_1} ((w')^{k_1} g_4)^{t_1}, \quad (2)$$

where $c_2 = g_3/g_4, r_1 = d - e, w' = u'/g_3, t_1 = r_1 - e$ and $d = \ell_1 + k_1 t_1$. It is recommended to choose the random blinding factor t_1 such that $t_1 \geq 2^\lambda$, where λ being the security parameter should be at least 64 bits long.

Queries to $\mathcal{U}$. $\mathcal{C}$ queries $\mathcal{U}$ in random order as follows:

1. $(r_1, c_1) \to c_1^{r_1}$;
2. $(r_1, c_2) \to c_2^{r_1}$;
3. $(\ell_1, w) \to w^{\ell_1}$;
4. $(k_1, w) \to w^{k_1}$;
5. $(\ell_1, w') \to (w')^{\ell_1}$;
6. $(k_1, w') \to (w')^{k_1}$.

Verifying the correctness of $\mathcal{U}$'s outputs. The client $\mathcal{C}$ checks whether

$$r\left(v_1 c_1^{r_1} w_1 w^{\ell_1} (w^{k_1} g_2)^{t_1}\right) \overset{?}{=} v_2 c_2^{r_1} w_2 (w')^{\ell_1} ((w')^{k_1} g_4)^{t_1} \tag{3}$$

Recovering $u^d \bmod N$. If the above check passes, $\mathcal{C}$ computes the result as

$$u^d = v_1 c_1^{r_1} w_1 w^{\ell_1} (w^{k_1} g_2)^{t_1}. \tag{4}$$

Otherwise $\mathcal{C}$ outputs error message.

Attack on CExp Algorithm Li *et al.* claim that their algorithm CExp is 1-verifiable; that is, it allows the client $\mathcal{C}$ to verify the outputs returned by $\mathcal{U}$ with probability 1. We show that with a minimal effort, $\mathcal{U}$ can cheat $\mathcal{C}$ with the malformed outputs and hence CExp offers unfortunately 0-verifiability.

The attacker's strategy is to identify and segregate just 2 out of 6 queries for which the exponent is same. For instance, let $\mathcal{U}$ choose (r_1, c_1) and (r_1, c_2). Note that this separation is easier since each base value c_i is distinct and does not appear twice. Now the malicious $\mathcal{U}$ manipulates the outputs corresponding to these two queries only by multiplying them with a random $\delta \in \{1, N\}$ and proceeds as follows.

1. $(r_1, c_1) \to \delta c_1^{r_1}$;
2. $(r_1, c_2) \to \delta c_2^{r_1}$;
3. $(\ell_1, w) \to w^{\ell_1}$;
4. $(k_1, w) \to w^{k_1}$;
5. $(\ell_1, w') \to (w')^{\ell_1}$;
6. $(k_1, w') \to (w')^{k_1}$.

After receiving the outputs, $\mathcal{C}$ checks if

$$r\left(v_1 \delta c_1^{r_1} w_1 w^{\ell_1} (w^{k_1} g_2)^{t_1}\right) \overset{?}{=} v_2 \delta c_2^{r_1} w_2 (w')^{\ell_1} ((w')^{k_1} g_4)^{t_1}.$$

Or equivalently, $r\delta u^d \overset{?}{=} \delta (u')^d$.
Since the check has been passed, $\mathcal{C}$ finally computes the unintended output:

$$u^d = v_1 \delta c_1^{r_1} w_1 w^{\ell_1} (w^{k_1} g_2)^{t_1}.$$

3.2 Cai *et al.* Scheme (SgExp) and Its Weakness

For the single untrusted server model, Cai *et al.* [5] proposed two algorithms to securely outsource single and simultaneous modular exponentiations with verifiability probability being close to 1. We show in this section that both their variants fail to detect wrong values output by the malicious server.

SgExp Algorithm We use the same notations followed by Cai *et al.* to describe their outsourcing algorithm SgExp.

Masking the Inputs. To mask the inputs, the algorithm SgExp used the preprocessing techniques BPV^+ or SMBL to generate four pairs (α, g^α), (β, g^β), $(\epsilon, g^\epsilon), (\theta, g^\theta)$ denoted by A, B, C and D respectively. Let $w = u/A \mod p$ and $v = u/C \mod p$. Then $\mathcal{C}$ represents u in the following two ways:
- $u^a = (Aw)^a = g^{a\alpha} w^a = g^\beta g^\gamma w^a \mod p$, where $\gamma = (a\alpha - \beta) \mod q$;
- $u^a = (Cv)^a = g^{a\epsilon} v^a = g^\theta g^\tau v^a \mod p$, where $\tau = (a\epsilon - \theta) \mod q$.

To implicitly mask a in w^a and v^a, $\mathcal{C}$ randomly chooses i, j such that $2^i \neq 2^j < a$ and computes
- $a_1 = a - 2^i$
- $a_2 = a - 2^j$.

Queries to $\mathcal{U}$. $\mathcal{C}$ runs BPV^+ or SMBL to generate 8 pairs $(t_1, g^{t_1}), (t_2, g^{t_2})$ and $(s_1, g^{s_1}), (s_2, g^{s_2}) \cdots (s_6, g^{s_6})$. Then $\mathcal{C}$ query $\mathcal{U}$ in random order after choosing $m_1, \cdots m_{i-1}, m_{i+1}, \cdots m_{j-1}, m_{j+1}, \cdots m_n$ as follows:
- $(g^{t_1}, \gamma/t_1, p) \to g^\gamma$;
- $(wg^{s_1}, a_1, p) \to R_{11} = w^{a_1} g^{s_1 a_1}$;
- $(g^{s_3}, \frac{s_1 a_1 - s_2}{s_3}, p) \to R_{12} = g^{s_1 a_1 - s_2}$;
- $(g^{t_2}, \tau/t_2, p) \to g^\tau$;
- $(vg^{s_4}, a_2, p) \to R_{21} = v^{a_2} g^{s_4 a_2}$;
- $(g^{s_6}, \frac{s_4 a_2 - s_5}{s_6}, p) \to R_{22} = g^{s_4 a_2 - s_5}$;
- $(m_1, 2) \to m[1] = m_1^2$;
- $(m_2, 2^2) \to m[2] = m_2^4$;
- $\cdots$
- $(w, 2^i) \to m[i] = w^{2^i}$;
- $\cdots$
- $(v^{-1}, 2^j) \to m[j] = v^{-2^j}$;
- $\cdots$
- $(m_n, 2^n) \to m[n] = m_n^{2^n}$;

Verifying the correctness of $\mathcal{U}$'s outputs. The client $\mathcal{C}$ computes

$$w^{a_1} = R_{11}(R_{12} g^{s_2})^{-1}$$
$$v^{a_2} = R_{21}(R_{22} g^{s_5})^{-1}$$

and checks whether

$$Bg^\gamma w^{a_1} m[i] m[j] \mod p \overset{?}{\equiv} Dg^\tau v^{a_2} \mod p \tag{5}$$

Recovering u^a. If the above check passes, $\mathcal{C}$ computes

$$u^a \equiv Bg^\gamma w^{a_1} m[i] \mod p.$$

Otherwise $\mathcal{C}$ outputs the error symbol $\perp$.

Attack on SgExp Algorithm Cai *et al.* claim that their algorithm SgExp preserves secrecy and the client $\mathcal{C}$ can verify the outputs returned by $\mathcal{U}$ with probability $1 - 1/n^2$. We show that even a minimal effort from $\mathcal{U}$ could lead to cheating the client $\mathcal{C}$ with the malformed outputs and hence SgExp is unfortunately 0-verifiable.

The attacker's strategy is to identify and segregate n out of $n + 6$ queries for which the first argument is a power of 2. Then the attacker forms two bins; to fill n number of queries of the form 2^i in one bin and the remaining 6 queries in the other bin. Note that this kind of separation of $n + 6$ queries is possible as exponent is a power of 2 in n queries. After segregation of n queries, the adversary manipulates the outputs of remaining 6 queries only by multiplying them with a random $\delta \in \mathbb{G}$ and proceeds as follows.

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

After receiving the outputs, $\mathcal{C}$ computes

$$w^{a_1} = \delta R_{11} (\delta R_{12} g^{s_2})^{-1} = R_{11} (R_{12} g^{s_2})^{-1}$$
$$v^{a_2} = \delta R_{21} (\delta R_{22} g^{s_5})^{-1} = R_{21} (R_{22} g^{s_5})^{-1}$$

and checks

$$B \delta g^{\gamma} w^{a_1} m[i] m[j] \mod p \stackrel{?}{=} D \delta g^{\tau} v^{a_2} \mod p. \tag{6}$$

Since the check has been passed, $\mathcal{C}$ finally computes the undesired output:

$$u^a = B \delta g^{\gamma} \delta w^{a_1} m[i] \mod p \neq B g^{\gamma} w^{a_1} m[i] \mod p.$$

3.3 Zhou *et al.* Scheme (ExpSOS) and Its Weakness

Zhou *et al.*[25] proposed several algorithms for outsourcing variable-exponent variable-base modular exponentiation using only a single untrusted server. In this section, we consider only the most generic algorithm, namely **ExpSOS** under malicious model [25, Section IV]. However our observations here are also applicable to other versions in the paper [25].

ExpSOS Algorithm Let N be either a prime number or an RSA modulus and $u, a \in \mathbb{Z}_N$. The aim of the client $\mathcal{C}$ is to compute $u^a \bmod N$ keeping the variable values u, a and u^a secret. The client runs the oracle $\mathcal{U}$ whose task is to return $i^j \bmod k$ on input (i, j, k). In order to maintain the secrecy of u and a they are computationally masked before being given as input to $\mathcal{U}$.

Masking the Inputs. To mask the inputs, the client $\mathcal{C}$ generates a large prime p and calculates $L = pN$ keeping N and p secret from the server $\mathcal{U}$. By choosing the random integers k_1, k_2, t_1, t_2, r such that $t_1, t_2 \leq b$, (where b is a security parameter) $\mathcal{C}$ calculates the following:

1. $A_1 = a + k_1\phi(N)$
2. $A_2 = t_1 a + t_2 + k_2\phi(N)$
3. $U = u + rN \bmod L$.

Queries to $\mathcal{U}$. $\mathcal{C}$ queries $\mathcal{U}$ in random order as follows:

1. $(U, A_1, L) \rightarrow R_1$
2. $(U, A_2, L) \rightarrow R_2$.

Verifying the correctness of $\mathcal{U}$'s outputs. The client $\mathcal{C}$ checks whether

$$R_1^{t_1} \cdot u^{t_2} \overset{?}{\equiv} R_2 \mod N. \tag{7}$$

Recovering u^a. If the above check passes, $\mathcal{C}$ computes

$$u^a \equiv R_1 \mod N. \tag{8}$$

Otherwise $\mathcal{C}$ outputs error message.

Attack on ExpSOS Algorithm The first generic algorithm for outsourcing variable-exponent variable-base modular exponentiation using only a single untrusted server was due to Wang *et al.* [23]. In [7] Chevalier *et al.* presented a lattice-based attack on Wang *et al.*'s scheme recovering the secret exponent when it appears again in another invocation. In this section, we follow the approach of Chevalier *et al.* and describe a similar attack on **ExpSOS** when the same secret exponent is used in two or more runs. In fact we will show that an exponent in **ExpSOS** can be recovered in polynomial time when two exponentiations having the same exponent are outsourced to the server $\mathcal{U}$. The assumed scenario is evident from the first application proposed in [25, Section VI.A] to securely offload Inner Product Encryption for Biometric Authentication [13].

The considered attack scenario is the following: The client wants to compute $u^a \mod N$ first and $(u')^a \mod N$ later. Let (U, A_1, A_2) and (U', A_1', A_2') be the queries to $\mathcal{U}$ corresponding to two exponentiations such that

$$A_1 = a + k_1\phi(N); A_2 = t_1 a + t_2 + k_2\phi(N)$$

and

$$A_1' = a + k_3\phi(N); A_2' = t_3 a + t_4 + k_4\phi(N).$$

Now, subtracting the first exponents in two exponentiations gives $A_1 - A_1' = (k_1 - k_3)\phi(N)$. Thus, given a multiple of $\phi(N)$, $\mathcal{U}$ can recover the secret exponent a in polynomial time using the well-known Miller's algorithm. (Miller in [19] showed that factoring of n is possible given any multiple of $\phi(n)$).

Remark 1. The above attack breaks the secrecy of other versions of **ExpSOS** in [25]. In fact it is applicable even for **ExpSOS** under honest-but-curious server model in [25, Section III.C]. The malicious server could act benignly in computing the required values but can learn silently any reused secret exponent. In the next section we attempt to thwart this attack under malicious server model since there is no efficiency gain to consider having the semi-honest server.

4 Our Algorithms for Single and Simultaneous Exponentiations

We first present the algorithm for single exponentiation by revising the ExpSOS algorithm and then extend the resulting algorithm for simultaneous exponentiations.

4.1 Improved ExpSOS Scheme for Single Exponentiation

In this section we present an improved ExpSOS algorithm which resists attack described in Section 3.3. We use the same notations from Section 3.3 used to describe the ExpSOS algorithm.

The MExpSOS Algorithm Let N be either a prime number or an RSA modulus and $u, a \in \mathbb{Z}_N$. The aim of the client $\mathcal{C}$ is to compute $u^a \bmod N$ keeping the variable values u, a and u^a secret.

Masking the Inputs. To mask the inputs, the client $\mathcal{C}$ generates a large prime p and calculates $L = pN$ to keep N and p secret from the server $\mathcal{U}$. Select a random r such that $N' = rN$ is fixed for all invocations. By choosing the random integers k_1, k_2, t_1, t_2 such that $t_1, t_2 \leq b$ (where b is a security parameter), $\mathcal{C}$ calculates the following:
1. $a_1 = a - t_1$
2. $A_1 = a_1 + k_1\phi(N)$
3. $A_2 = t_2 a + k_2\phi(N)$
4. $U = u + N' \mod L$

Queries to $\mathcal{U}$. $\mathcal{C}$ queries $\mathcal{U}$ in random order as follows:
1. $(U, A_1, L) \rightarrow R_1$
2. $(U, A_2, L) \rightarrow R_2$

Verifying the correctness of $\mathcal{U}$'s outputs. The client $\mathcal{C}$ checks whether

$$(R_1 u^{t_1})^{t_2} \mod N \overset{?}{=} R_2 \mod N \tag{9}$$

Recovering u^a. If the above check passes, $\mathcal{C}$ computes

$$u^a = R_1 u^{t_1} \mod N \tag{10}$$

Otherwise $\mathcal{C}$ outputs error message.

Remark 2. The performance gain of the above algorithm is that instead of a *full modular exponentiation*(on the size of an RSA private key, for example), the client device needs to do 2 *smaller* exponentiations with size comparable to the security parameter. Moreover the communication cost and the overhead for the third-party server are not prohibitive compared to the previously-known algorithms. Hence the proposed solution shall directly produce speed-ups in practice.

Lemma 1. *(Correctness). In the malicious model, the algorithms* $(\mathcal{C},\mathcal{U})$ *are correct implementation of* $\mathsf{MExpSOS}$.

Proof. Whenever $\mathcal{U}$ returns R_1 and R_2, $\mathcal{C}$ computes u^{t_1} on its own and then raising the value $R_1u^{t_1} \mod N$ to the power t_2. Then the resultant $(R_1u^{t_1})^{t_2} \mod N$ is compared with R_2. If the equality holds, then $\mathcal{C}$ computes the desired result $u^a = R_1u^{t_1} \mod N$. □

In the following theorem, we show that $(\mathcal{C},\mathcal{U})$ is an outsource-secure implementation of $\mathsf{MExpSOS}$ using Hohenberger-Lysyanskaya security model for a single malicious server. [12].

Theorem 1. *(Privacy) In the one malicious program model, the algorithms* $(\mathcal{C},\mathcal{U})$ *are an outsource-secure implementation of* $\mathsf{MExpSOS}$.

Proof. Assume that $\mathcal{A} = (\mathcal{E},\mathcal{U}')$ be a probabilistic polynomial time (PPT) adversary which interacts with the PPT algorithm $\mathcal{C}$ in the one malicious program model.

Pair One: ($\mathcal{E}$ learns nothing) $\text{EVIEW}_{real} \sim \text{EVIEW}_{ideal}$

If the input (u, a, N) is honest, protected or adversarial protected, the simulator $\mathcal{S}_1$ behaves the same way as in the real experiment. If the input is honest and secret, then $\mathcal{S}_1$ ignores the received input in the ith round. The goal of $\mathcal{S}_1$ in this ith round is to query $\mathcal{U}'$ with the inputs (U^*, A_1^*, A_2^*, L^*) such that the inputs U^*, A_1^*, A_2^* and L^* are chosen at random by $\mathcal{S}_1$. After receiving the outputs, $\mathcal{S}_1$ saves both the states of $\mathcal{S}_1$ and $\mathcal{U}'$.

In real process, all the inputs that occur in the queries are re-randomized to give computational indistinguishability. Whereas $\mathcal{S}_1$ always set the queires to $\mathcal{U}'$ with the random input values. Hence the input distributions to $\mathcal{U}'$ are computationally indistinguishable both in the real and ideal process.

Pair Two: ($\mathcal{U}'$ learns nothing): $\text{UVIEW}_{real} \sim \text{UVIEW}_{ideal}$:
Let $\mathcal{S}_2$ be a PPT simulator that behaves in the same manner regardless of whether the input is honest, secret or honest, protected or adversarial protected. That is, $\mathcal{S}_2$ ignores the actual input in the ith round and set the queries to $\mathcal{U}'$ with the random value. Then $\mathcal{S}_2$ saves not only its state but also $\mathcal{U}'$'s state.

Whenever the inputs to the experiment are honest, protected and adversarial protected, $\mathcal{E}$ can easily distinguish *ith* round of two experiments. But it is of no help as $\mathcal{E}$ cannot communicate to $\mathcal{U}'$ and the inputs are computationally blinded by $\mathcal{C}$ before being given as input to $\mathcal{U}'$ in the ideal experiment. In the ideal experiment, the simulator $\mathcal{S}_2$ always query the values selected uniform at random from the same distribution. Hence $\text{UVIEW}^i_{real} \sim \text{UVIEW}^i_{ideal}$ for each round i. By the hybrid argument, it is easy to see that $\text{UVIEW}_{real} \sim \text{UVIEW}_{ideal}$. □

Theorem 2. *(verifiability) In the one malicious program model, the above algorithms* $(\mathcal{C},\mathcal{U})$ *are an* $(3+1.5(\log t_1 + \log t_2), 1-1/2b)$*-outsource-secure implementation of* MExpSOS.

Proof. The computation of modular exponentiation $u^a \bmod N$ without outsourcing requires roughly $1.5 \log a$ modular multiplications (MM) using square and multiply method. As discussed in [25], the computational overhead to calculate $\phi(N), L$ and N' becomes negligible when the client $\mathcal{C}$ runs MExpSOS multiple times. The following paragraph shows that with outsourcing, the modular exponentiation computation is reduced to $3+1.5(\log t_1 + \log t_2)$ modular multiplications: the computation of A_1 and A_2 during the masking step requires two modular multiplication altogether. Then the verification step requires one modular exponentiation (u^{t_1}), one modular multiplication $(R_1 u^{t_1})$ and one modular exponentiation $((R_1 u^{t_1})^{t_2})$. Thus our algorithm MExpSOS requires 3 modular multiplications and two $b-$bit modular exponentiations. Therefore our algorithm $(\mathcal{C},\mathcal{U})$ is an $(\frac{1}{2}\log_b a)-$efficient implementation of MExpSOS.

On the other hand the two outputs sent by $\mathcal{U}$ are verified as in Equation 11. The server $\mathcal{U}$ can trick the client $\mathcal{C}$ if it correctly guesses t_2 as in the following:

- Assume that the malicious $\mathcal{U}$ sets A_1 as $A_1+\theta$ and A_2 as $A_2+\theta$
- Then the Equation 11 becomes

$$\begin{aligned}(U^{A_1+\theta}u^{t_1})^{t_2} &\equiv U^{A_2+\theta} \mod N\\ u^{a_1 t_2+\theta t_2}u^{t_1 t_2} &\equiv u^{t_2 a+\theta} \mod N\\ u^{a t_2}u^{\theta t_2} &\equiv u^{t_2 a+\theta} \mod N\end{aligned}$$

If the value t_2 is correctly guessed then the adversary can compute $u^{\theta t_2}$ and set A_1 as $A_1+\theta$ and A_2 as $A_2+\theta t_2$ to pass the verification. If t_2 is guessed with probability $1/b$ and θt_2 is inserted accordingly in one out of two queries sent in random order, the malicious server can pass the verification step with false outputs with probability $\frac{1}{2b}$. Hence our algorithm is a $(1-\frac{1}{2b})$-verifiable implementation of MExpSOS.

The proof of the theorem completes by combining the above arguments. □

4.2 New Algorithm for Simultaneous Exponentiation

In this section, we present a generic algorithm πExpSOS for simultaneous exponentiation whose complexity grows linearly in size of the number of exponentiations. Simultaneous modular exponentiations appear predominantly in cryptographic primitives such as provable data possession [2] and proofs of retrievability [3].

The πExpSOS Algorithm Let us follow the notations used to describe the MExpSOS algorithm. Let N be either a prime number or an RSA modulus and $u_i, a_i \in \mathbb{Z}_N$ for $i = 1, \dots, n$. In order to maintain the secrecy of u_i and $a_i, i = 1, \dots, n$ they are computationally masked before being given as input to $\mathcal{U}$.

Masking the Inputs. To mask the inputs, the client $\mathcal{C}$ generates a large prime p and calculates $L = pN$ to keep N and p secret from the server $\mathcal{U}$. Select a random r such that $N' = rN$ is fixed for all invocations. By choosing the random integers $k_{1i}, k_{2i} (i = 1, \dots, n)$ and t_1, t_2 such that $t_1, t_2 \leq b$, $\mathcal{C}$ calculates the following for $i = 1, \dots, n$:

1. $a_{1i} = a_i - t_1$
2. $A_{1i} = a_{1i} + k_{1i}\phi(N)$
3. $A_{2i} = t_2 a_i + k_{2i}\phi(N)$
4. $U_i = u_i + N' \mod L$

Queries to $\mathcal{U}$. $\mathcal{C}$ issues $2n$ queries to $\mathcal{U}$ in random order as follows:

1. $(U_i, A_{1i}, L) \to R_{1i}$
2. $(U_i, A_{2i}, L) \to R_{2i}$

Verifying the correctness of $\mathcal{U}$'s outputs. The client $\mathcal{C}$ checks whether

$$\left[\prod_{i=1}^{n} R_{1i} (\prod_{i=1}^{n} u_i)^{t_1}\right]^{t_2} \mod N \overset{?}{=} \prod_{i=1}^{n} R_{2i} \mod N \tag{11}$$

Recovering u^a. If the above check passes, $\mathcal{C}$ computes

$$\prod_{i=1}^{n} u_i^{a_i} \equiv \prod_{i=1}^{n} R_{1i} (\prod_{i=1}^{n} u_i)^{t_1} \mod N \tag{12}$$

Otherwise $\mathcal{C}$ outputs error message.

Lemma 2. *(Correctness). In the malicious model, the algorithms* $(\mathcal{C}, \mathcal{U})$ *are correct implementation of* πExpSOS.

Proof. Whenever $\mathcal{U}$ returns R_{1i} and R_{2i} for $i = 1 \dots n$, $\mathcal{C}$ computes $(\prod_{i=1}^{n} u_i)^{t_1}$ on its own and then raising the value $\prod_{i=1}^{n} R_{1i} (\prod_{i=1}^{n} u_i)^{t_1} \mod N$ to the power t_2. Then the resultant $(\prod_{i=1}^{n} R_{1i} (\prod_{i=1}^{n} u_i)^{t_1})^{t_2} \mod N$ is compared with $\prod_{i=1}^{n} R_{2i}$. If the equality holds, then $\mathcal{C}$ computes the desired result

$$\prod_{i=1}^{n} u_i^{a_i} \equiv \prod_{i=1}^{n} R_{1i} (\prod_{i=1}^{n} u_i)^{t_1} \mod N. \qquad \square$$

In the following theorem, we show that $(\mathcal{C}, \mathcal{U})$ is an outsource-secure implementation of πExpSOS using Hohenberger-Lysyanskaya security model for a single malicious server. [12].

Theorem 3. *(Privacy) In the one malicious program model, the algorithms* $(\mathcal{C}, \mathcal{U})$ *are an outsource-secure implementation of* πExpSOS.

We omit the proof to this theorem as this can be easily written using Theorem 1

Theorem 4. *(verifiability) In the one malicious program model, the above algorithms* $(\mathcal{C},\mathcal{U})$ *are an* $(5n-2+1.5(\log t_1+\log t_2), 1-1/2b)$*-outsource-secure implementation of* $\pi\mathsf{ExpSOS}$.

Proof. The computation of modular exponentiation $\prod_{i=1}^{n} u_i^{a_i} \bmod N$ without outsourcing requires roughly $1.5n\log a$ modular multiplications (MM) using square and multiply method. Outsourcing the modular exponentiation computation reduces the cost to $2n+3(n-1)+1.5(\log t_1+\log t_2)+1$ modular multiplications as detailed below: the computation of A_{1i} and A_{2i} during the masking step requires $2n$ modular multipications altogether. Then the verification step requires $n-1$ modular multiplications to compute $\prod_{i=1}^{n} u_i$, one modular exponentiation to compute $\prod_{i=1}^{n} u_i^{t_1}$, $n-1$ modular multiplications to compute $\prod_{i=1}^{n} R_{1i}$, one modular multiplication to compute $\prod_{i=1}^{n} R_{1i}\prod_{i=1}^{n} u_i^{t_1}$, one modular exponentiation to compute $(\prod_{i=1}^{n} R_{1i}\prod_{i=1}^{n} u_i^{t_1})^{t_2}$ and $n-1$ modular multiplications to compute $\prod_{i=1}^{n} R_{2i}$. Thus our algorithm $\pi\mathsf{ExpSOS}$ requires $2n+1+3(n-1)=5n-2$ modular multiplications and 2 b-bit modular exponentiations. Therefore our algorithm $(\mathcal{C},\mathcal{U})$ is an $(\frac{1}{2}\log_b a)-$efficient implementation of $\pi\mathsf{ExpSOS}$.

On the other hand the $2n$ outputs sent by $\mathcal{U}$ are verified as in Equation 11. The server $\mathcal{U}$ can trick the client $\mathcal{C}$ if it correctly guess t_2 as in explained in Theorem 2 with probability $1/b$. Thus an adversary can pass the verification step with false outputs with probability $\frac{1}{2b}$. Hence our algorithm is a $(1-\frac{1}{2b})$-verifiable implementation of $\pi\mathsf{ExpSOS}$. The proof of the theorem completes by combining the above arguments. □

5 Conclusion

The need for reducing cost of cryptographic computations is growing especially in the case of devices having resource scarcity. We reviewed several algorithms for offloading single and simultaneous modular exponentiations to a single untrusted helper. In **CExp** and **SgExp** algorithms, we demonstrated that the falsified values of a malicious server could go undetected by the client in the verification and hence the client outputs the unintended value. For **ExpSOS**, we presented a practical attack revealing the secret exponent challenging the claimed security guarantees. We then proposed modifications to the **ExpSOS** algorithm and proved that the resulting algorithm **MExpSOS** meets the fundamental security requirements of the Hohenberger-Lysyanskaya security model. We finally solved an intriguing problem underlined in [6] by proposing π**ExpSOS**, the most efficient to-date algorithm using single untrusted (cloud) server for securely outsourcing (multi-)modular exponentiation. Our proposal being near error-free and preprocessing-free is of both theoretical and practical interest.

References

1. Abadi, M., Feigenbaum, J., Kilian, J.: On hiding information from an oracle. In: Proceedings of the Second Annual Conference on Structure in Complexity Theory. pp. 195–203. IEEE Computer Society (1987), https://doi.org/10.1016/0022-0000(89)90018-4

2. Ateniese, G., Burns, R., Curtmola, R., Herring, J., Kissner, L., Peterson, Z., Song, D.: Provable data possession at untrusted stores. In: Proceedings of the 14th ACM Conference on Computer and Communications Security. pp. 598–609. CCS '07, ACM, New York, NY, USA (2007), http://doi.acm.org/10.1145/1315245.1315318
3. Bowers, K.D., Juels, A., Oprea, A.: Proofs of retrievability: Theory and implementation. In: Proceedings of the 2009 ACM Workshop on Cloud Computing Security. pp. 43–54. CCSW '09, ACM, New York, NY, USA (2009), http://doi.acm.org/10.1145/1655008.1655015
4. Boyko, V., Peinado, M., Venkatesan, R.: Speeding up discrete log and factoring based schemes via precomputations. In: Nyberg, K. (ed.) Advances in Cryptology – Proc. EUROCRYPT '98. LNCS, vol. 1403, pp. 221–235. Springer (1998), https://doi.org/10.1007/BFb0054129
5. Cai, J., Ren, Y., Huang, C.: Verifiable outsourcing computation of modular exponentiations with single server. International Journal of Network Security 19(3), 449–457 (May 2017), http://ijns.jalaxy.com.tw/download_paper.jsp?PaperID=IJNS-2015-12-05-1&PaperName=ijns-v19-n3/ijns-2017-v19-n3-p449-457.pdf
6. Chen, X., Li, J., Ma, J., Tang, Q., Lou, W.: New algorithms for secure outsourcing of modular exponentiations. In: Foresti, S., Yung, M., Martinelli, F. (eds.) Proc. ESORICS 2012, LNCS, vol. 7459, pp. 541–556. Springer (2012)
7. Chevalier, C., Laguillaumie, F., Vergnaud, D.: Privately outsourcing exponentiation to a single server: Cryptanalysis and optimal constructions. In: Askoxylakis, I.G., Ioannidis, S., Katsikas, S.K., Meadows, C.A. (eds.) Proc. ESORICS 2016, Part I. LNCS, vol. 9878, pp. 261–278. Springer (2016), https://doi.org/10.1007/978-3-319-45744-4
8. Gennaro, R., Gentry, C., Parno, B.: Non-interactive verifiable computing: Outsourcing computation to untrusted workers. In: Rabin, T. (ed.) IACR CRYPTO 2010. LNCS, vol. 6223, pp. 465–482. Springer (2010)
9. Girault, M., Lefranc, D.: Server-aided verification: Theory and practice. In: Roy, B.K. (ed.) IACR ASIACRYPT 2005. LNCS, vol. 3788, pp. 605–623. Springer (2005)
10. Golle, P., Mironov, I.: Uncheatable distributed computations. In: Naccache, D. (ed.) Topics in Cryptology - CT-RSA 2001, The Cryptographer's Track at RSA Conference 2001. LNCS, vol. 2020, pp. 425–440. Springer (2001)
11. Green, M., Hohenberger, S., Waters, B.: Outsourcing the decryption of ABE ciphertexts. In: USENIX Security Symposium 2011. USENIX Association (2011), https://www.usenix.org/publications/proceedings/?f[0]=im_group_audience%3A277
12. Hohenberger, S., Lysyanskaya, A.: How to securely outsource cryptographic computations. In: Kilian, J. (ed.) IACR TCC 2005. LNCS, vol. 3378, pp. 264–282. Springer (2005)
13. Kim, S., Lewi, K., Mandal, A., Montgomery, H., Roy, A., Wu, D.J.: Function-Hiding Inner Product Encryption is Practical. Cryptology ePrint Archive, Report 2016/440 (2016), accepted at SCN 2018. https://eprint.iacr.org/2016/440
14. Kiraz, M.S., Uzunkol, O.: Efficient and verifiable algorithms for secure outsourcing of cryptographic computations. International Journal of Information Security 15(5), 519–537 (Oct 2016), https://doi.org/10.1007/s10207-015-0308-7
15. Kuppusamy, L., Rangasamy, J.: Crt-based outsourcing algorithms for modular exponentiations. In: Dunkelman, O., Sanadhya, S.K. (eds.) Proc. INDOCRYPT 2016. LNCS, vol. 10095, pp. 81–98. Springer (2016)

16. Li, S., Huang, L., Fu, A., Yearwood, J.: CExp: secure and verifiable outsourcing of composite modular exponentiation with single untrusted server. Digital Communications and Networks 3(4), 236–241 (2017)
17. Matsumoto, T., Kato, K., Imai, H.: Speeding up secret computations with insecure auxiliary devices. In: Goldwasser, S. (ed.) IACR CRYPTO 1988. LNCS, vol. 403, pp. 497–506. Springer (1988)
18. McLoone, M., Robshaw, M.J.B.: Public key cryptography and rfid tags. In: Proceedings of the 7th Cryptographers' Track at the RSA Conference on Topics in Cryptology. pp. 372–384. CT-RSA'07, Springer-Verlag, Berlin, Heidelberg (2006), `http://dx.doi.org/10.1007/11967668_24`
19. Miller, G.L.: Riemann's hypothesis and tests for primality. In: Rounds, W.C., Martin, N., Carlyle, J.W., Harrison, M.A. (eds.) Proc. ACM Symposium on Theory of Computing (STOC) 1975. pp. 234–239. ACM (1975)
20. Nguyen, P.Q., Shparlinski, I.E., Stern, J.: Distribution of modular sums and the security of the server aided exponentiation. In: Proc. Cryptography and Computational Number Theory Workshop. pp. 257–268. Birkh"auser (2001)
21. Schnorr, C.P.: Efficient identification and signatures for smart cards. In: Brassard, G. (ed.) IACR CRYPTO '89. LNCS, vol. 435, pp. 239–252. Springer (1990)
22. Schnorr, C.P.: Efficient signature generation by smart cards. J. Cryptology 4(3), 161–174 (1991)
23. Wang, Y., Wu, Q., Wong, D., Qin, B., Chow, S., Liu, Z., Tan, X.: Securely outsourcing exponentiations with single untrusted program for cloud storage. In: Kutyłowski, M., Vaidya, J. (eds.) Proc. ESORICS 2014. LNCS, vol. 8712, pp. 326–343. Springer (2014), `http://dx.doi.org/10.1007/978-3-319-11203-9_19`
24. 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 (2008)
25. Zhou, K., Afifi, M.H., Ren, J.: ExpSOS: Secure and Verifiable Outsourcing of Exponentiation Operations for Mobile Cloud Computing. IEEE Transactions on Information Forensics and Security 12(11), 2518–2531 (Nov 2017)