Efficient and Verifiable Algorithms for Secure Outsourcing of Cryptographic Computations

Mehmet Sabır Kiraz and Osmanbey Uzunkol [*]

Abstract. Being required in many applications, modular exponentiations form the most expensive part of modern cryptographic primitives. It is a significant challenge for resource-constrained mobile devices to perform these heavy computations (e.g., mobile devices that require secure cryptographic computations or RFID tags). Cloud services can significantly enhance the computational capability of these devices. In this way, expensive computations at client side can significantly be reduced by means of secure outsourcing modular exponentiations to a potentially untrusted server $\mathcal{S}$. In this paper, we study this problem which is an active research area of mobile and cloud computing, and mostly known as *secure outsourced computation*. We propose new efficient outsourcing algorithms for modular exponentiations using only one untrusted cloud service provider solving an open problem highlighted in [11]. These algorithms cover each possible case ranging from public-base & private-exponent, private-base & public-exponent, private-base & private-exponent to the most general private-basis & private-exponents simultaneous modular exponentiations. Our algorithms are the most efficient outsourced computation algorithms to date which use single untrusted server and have the best checkability (verifiability) property. Finally, we give two different real-life applications for outsourcing within the realm of Oblivious Transfer protocols and Blind Signatures.

Keywords: Secure outsourcing algorithms, Modular exponentiation, Mobile computing, Secure cloud computing, Privacy.

1 Introduction

Security and privacy of Cloud Computing is getting more and more attention in the scientific community due to their multiple benefits for the real-world applications (e.g., on-demand self-service, ubiquitous network access (internet standards based), location independent resource pooling, pay per use, rapid elasticity, and measure/metered service, and outsourcing). Depending on the need of configurable computing resources, it is possible to efficiently outsource costly calculations to more powerful servers using cloud computing techniques.

[*] Mehmet Sabır Kiraz and Osmanbey Uzunkol are with Mathematical and Computational Sciences Labs, TÜBİTAK BİLGEM UEKAE, Kocaeli, Turkey (e-mail: {mehmet.kiraz, osmanbey.uzunkol}@tubitak.gov.tr.

Today's resource-constrained devices can be incapable of computing expensive cryptographic operations like modular exponentiations. This is the main reason that outsourcing computation plays more and more a predominant role in real-life cryptographic applications. For example, modular exponentiation of the form u^a modulo a prime number p where u, a, and p have minimum length 2048 (in order to have a cryptographically secure algorithm) has computational obstacle for the computationally limited devices. To compute a single modular exponentiation for an 2048-bit exponent a, more than 3000 modular multiplications must be performed in average (using square and multiply method). Therefore, it is vital to outsource the expensive computations to the cloud providers. Nevertheless, the outsourced computations often contain additional sensitive information that should not be revealed to the outsiders (e.g., personal, health or financial data). In order to prevent information leakage, the sensitive data has to be masked before outsourcing. On the one hand, the masking technique should be designed in such a way that the overall computational cost to the client is significantly reduced, i.e., reducing the cost of masking before outsourcing and the cost of removing the mask after obtaining the result from cloud provider are of utmost important. On the other hand, it is also essential to assure the client that it computes the desired output correctly, i.e., malicious server or environmental attacks should not be successful without being detected with a non-negligible probability. Therefore, it is an inevitable requirement not only to have an efficient outsourcing algorithm but also to prevent private information leakage from an untrusted cloud provider during the outsourcing procedure by means of checking/verifying the result correctly.

It is one of the most basic assumption that there exist fully-trusted or semi-trusted cloud providers. Nevertheless, it is not realistic to assume trusted parties and it is not that likely the case in real-life scenarios. For example, due to financial reasons, the cloud providers might contain a software bug that will fail after some particular steps of the algorithms and then return a wrong result which is computationally indistinguishable from the correct output. By the checkability property, the client can easily detect any malicious behavior from the cloud servers side. How then the security and the privacy of the clients' data are ensured without revealing the private inputs and the desired outputs while ensuring that outsourced computation is performed correctly using only a single untrusted server?

(a) Our Contributions

In this paper, we propose new, efficient and secure outsourcing algorithms of modular exponentiations modulo prime or composite numbers utilizing only one untrusted server. Our algorithms consider each case public-base & private-exponent, private-base & public-exponent, private-base & private-exponent and simultaneous modular exponentiations. We highlight that the proposed algorithms borrow computing power from only a single untrusted cloud server which is more realistic compared to the state-of-the-art algorithms in [11, 23, 26]. This approach realizes privacy preserving efficient outsourced cryptographic schemes which are highly desirable and mostly inevitable for real-life applications in resource-constrained secure mobile environments. To the best of our knowledge, our outsourcing algorithms make for the first time no distinction between prime and composite modulus by a unified modular exponentiation approach. Therefore, exponentiations in both DLP based and RSA problem based cryptographic protocols can be outsourced securely to an untrusted server.

We notice that in [31], the authors proposed an algorithm utilizing a single untrusted server (for public-base & private-exponent and private-base & public-exponent cases). However, there is a security issue in their algorithm where an untrusted server can easily learn the private exponent (e.g., using the local notation: For $t = 1$ the client invokes the server $\mathsf{Exp}(a_i, g)$ for $1 \leq i \leq n$ in order to delegate the computation of $g^{x_1} = g^{\sum_{i=1}^{n} a_i}$. The server then simply adds permuted values of a_i's and divides easily the result by 2 for learning the private exponent x_1). We also point out that there is indeed no checkability property of the only existing algorithm [30] for modular exponentiations modulo composite numbers. By using the local notation of [30], the attack can be explained briefly as follows: A malicious server $\mathcal{S}$ uses the proposed values $\ell = \ell_1 = \ell_2 = 5$ in [30] (or any other case for which $\ell = \ell_1 = \ell_2$ holds), adds 1 to the values y_j, and outputs $x_i^{y_j+1}$ instead of $x_i^{y_j}$. This enables the server to always manipulate the result u^a with $u^{a+\ell}$ without being detected by the client.

Furthermore, our algorithms are the only efficient and verifiable result for the case of one single untrusted server except the non-checkable result in [30]. We emphasize that although the existing solutions use two servers where one of them is assumed to be honest, they propose only 1/2, 2/3 or 3/4 probability for the checkability. In contrast to these solutions, our algorithms have the best checkability advantage where any adversarial behavior can be detected by the client with the probability 15/16.

We would like to highlight that our algorithm for simultaneous modular exponentiations is more efficient compared to the only existing algorithm in [11] (except the generalized result in [30], for which an analogous attack explained above makes the checkability step impossible). The algorithm proposed in [11] only considers two simultaneous modular exponentiations. We generalize this by means of introducing the notion of *t-simultaneous modular exponentiation*, i.e., t modular exponentiations can be computed simultaneously in a single round. We show that we gain linear complexity advantage in t for both the number of modular multiplications and inversions.

We also apply proposed algorithms to outsource Oblivious Transfer (OT) and Blind Signatures securely. Note that OT is a powerful cryptographic primitive which is "complete" for secure multiparty computation [22] for any computable function [25]. It is also one of the major computational overhead for Yao's garbled circuit protocols [28, 42]. OTs are also used in many applications like biometric authentication, e-auctions, private information retrieval, private search [9, 16, 24, 29]. Hence, by outsourcing OT securely can be enhance the overall complexity for mobile environment and resource-constrained devices. Furthermore, blind signatures [10] are unforgeable and can be verified against a public key like conventional digital signatures which can be used in many applications like e-cash, e-voting and anonymous credentials [8]. Hence, outsourcing blind signatures can be also solely beneficial for many real-life applications.

(b) Related Work

Outsourced secure computation allows parties to compute a functionality which is in the charge of the cloud, without leaking any information about the inputs except possibly the outputs. It is expected to be no interactions between the parties, and the computational cost and the bandwidth of each user are expected to be independent of the functionality. However, general program obfuscation is impossible utilizing only a single cloud server [41]. This is the reason that we solely focus on expensive modular computations for certain cryptographic primitives.

Many algorithms [1, 4, 5, 7, 11, 15, 23, 26, 27, 32–34, 37, 38] have been proposed aiming either within a better security model for outsourcing or at considering the efficiency. However, these algorithms only consider either outsourcing of a public-base & private-exponent or private-base & public-exponent or satisfy a weaker security notions. For example, in [40], Clarke *et al.* propose protocols for speeding up exponentiation in a cyclic group using untrusted servers for public-base & private-exponent and private-base & public-exponent. They also extend them to compute an exponentiation modulo a composite integer where the modulus is the product of two primes.

Hohenberger and Lysyanskaya [23] presented the first outsource-secure algorithm for modular exponentiations with a security model for outsourcing cryptographic computations. This algorithm considers the case private-base & private-exponent exponentiation modulo a prime number. With this algorithm, modular exponentiations can be computed by the client with $O(log^2(l))$ multiplications with error probability $\frac{1}{2}$, where l denotes the number of bits of the exponent element. The main drawback of this solution is outsourcing to two non-colluding untrusted servers to assist the client in the computations.

At ESORICS 2012, Chen *et al.* [11] propose a more efficient solution than Hohenberger-Lysyanskaya's algorithm and the probability of detection of a malicious behavior is improved to 2/3. However, modular exponentiations can be computed by the client with $O(log^2(l))$ multiplications. Chen *et al.* also presented the first secure outsourcing algorithm for simultaneous modular exponentiations. Simultaneous modular exponentiations $u_1^{a_1}u_2^{a_2}$ are also used in many cryptographic primitives such as commitments [19], zero-knowledge proofs [13] and additive variant of ElGamal encryption [18]. Chen *et al.* use the outsourcing algorithms to compute Cramer-Shoup encryptions and Schnorr signatures securely.

As another area of outsourcing techniques, homomorphic encryption allows parties for processing computations on encrypted data without using any additional information like Yao's garbled circuits [42]. Conventional homomorphic encryption schemes are either additive or multiplicative (e.g., RSA is multiplicative, Paillier and modified version of ElGamal encryption are additive [18,35], or [6] scheme which allows multiple additions up to only one exponentiation). These schemes allow to outsource secure function evaluation to a cloud server. Recent somewhat homomorphic and fully homomorphic schemes give a complete solution to the outsourcing problem [21]. However, these systems are not yet efficient enough to be applied in real-life scenarios.

(c) Roadmap

In Section 2, we give our formal security and privacy model based on the model of [23] by simplifying their *two untrusted server model* to a more realistic and secure *one single untrusted server model.* Section 3 starts with basic mathematical background of outsourcing algorithms of modular exponentiation as well as describes the main proposed algorithm for private-base & private-exponent modular exponentiations. We also prove formally the correctness, security and checkability of the proposed algorithm using security/privacy model presented in Section 2. In Section 4, we propose algorithms for all other relevant situations, i.e. public-base & private-exponent, private-base & public-exponent and private-base & private-exponent simultaneous modular exponentiations for both modulo prime and composite number. Section 5 compares the efficiency of our algorithms with each other and the prior works. In Section 6, we utilize our algorithms for Oblivious Transfer and Blind Signatures protocols. Finally, Section 7 concludes the paper.

2 Security and Privacy Model

Assume that a client $\mathcal{C}$ would like to securely outsource an expensive cryptographic computation Alg to a cloud server $\mathcal{S}$. Informally speaking, the goal is to split the computation into two procedures (1) $\mathcal{C}$ knows the input value to Alg, (2) $\mathcal{C}$ invokes $\mathcal{S}$ which is an untrusted server that can carry out expensive computation operations. Briefly, $\mathcal{C}$ securely outsource some computation if the following conditions hold:

1. $\mathcal{C}$ and $\mathcal{S}$ implement Alg, i.e., $\mathsf{Alg} = \mathcal{C}^{\mathcal{S}}$
2. Assume that $\mathcal{C}$ has oracle access to an adversary $\mathcal{S}'$ (instead of an honest $\mathcal{S}$) which stores its computational results during each run and behaves maliciously in order to learn extra information. $\mathcal{S}'$ is not able to retrieve any valuable information about the input-output pair of $\mathcal{C}^{\mathcal{S}'}$.

We are now ready to give the formal model for secure outsourced cryptographic algorithms based principally on the model of [23].

Definition 1. *[23]* ***(Algorithm with outsource-I/O)*** *An algorithm* Alg *obeys the outsource input/output specification if it takes five inputs, and produces three outputs. The first three inputs are generated by an honest party, and are classified by how much the adversary* $\mathcal{A} = (\mathcal{E}, \mathcal{S}')$ *knows about them, where* $\mathcal{E}$ *is the adversarial environment that submits maliciously chosen inputs to* Alg*, and* $\mathcal{S}'$ *is the adversarial software operating in place of oracle* $\mathcal{S}$*.*

1. *1st is the honest secret input, which is unknown to both* $\mathcal{E}$ *and* $\mathcal{S}'$*,*
2. *2nd is the honest protected input, which may be known by* $\mathcal{E}$*, but is protected from* $\mathcal{S}'$*,*

3. *3rd is the honest unprotected input, which may be known by both $\mathcal{E}$ and $\mathcal{S}$,*
4. *4th is the the adversarial protected input which is known to $\mathcal{E}$, but protected from $\mathcal{S}'$,*
5. *5th is the the adversarial unprotected input, which may be known by $\mathcal{E}$ and $\mathcal{S}$,*
6. *1st is the secret input which is unknown to both $\mathcal{E}$ and $\mathcal{S}'$,*
7. *2nd is the protected input which may be known to $\mathcal{E}$, but not $\mathcal{S}'$,*
8. *3rd is the unprotected input which may be known by both parties of $\mathcal{A}$.*

Outsource-security roughly means that if a malicious $\mathcal{S}'$ can obtain some information about the secret or protected inputs to $\mathcal{C}^{\mathcal{S}}$ from playing the role of $\mathcal{C}$ instead of $\mathcal{S}$, it is also in a position to obtain useful information without this procedure. More concretely, when $\mathcal{C}^{\mathcal{S}}(x)$ is queried, construct a simulator $\mathsf{Sim}_{\mathcal{S}'}$ such that without the knowledge of the secret or protected inputs of x the view of $\mathcal{S}'$ can be simulated. In the following outsource-security definition, it is guaranteed that the malicious environment $\mathcal{E}$ cannot learn any valuable information about the secret inputs and outputs of $\mathcal{C}^{\mathcal{S}}$ (even in the case that $\mathcal{C}$ runs the malicious software $\mathcal{S}'$ developed by $\mathcal{E}$).

Definition 2. *[23] **(Outsource security)** Let $\mathsf{Alg}(\cdot,\cdot,\cdot,\cdot,\cdot)$ be an algorithm with outsource-I/O. A pair of algorithms $(\mathcal{C},\mathcal{S})$ is said to be an outsource-secure implementation of Alg if:*
***Correctness:** $\mathcal{C}^{\mathcal{S}}$ is a correct implementation of Alg.*
***Security:** For all probabilistic polynomial-time adversaries $\mathcal{A} = (\mathcal{E},\mathcal{S}')$, there exist probabilistic expected polynomial-time simulators $(\mathsf{Sim}_{\mathcal{E}},\mathsf{Sim}_{\mathcal{S}'})$ such that the following pairs of random variables are computationally indistinguishable.*

– *Pair One.* $\mathsf{EVIEW}_{\mathsf{real}} \sim \mathsf{EVIEW}_{\mathsf{ideal}}$
  - *The real process:*
    $\mathsf{EVIEW}^i_{\mathsf{real}} = \{(\mathsf{istate}^i, x^i_{hs}, x^i_{hp}, x^i_{hu}) \leftarrow I(1^k, \mathsf{istate}^{i-1});$
    $(\mathsf{estate}, j^i, x^i_{ap}, x^i_{au}, \mathsf{stop}^i) \leftarrow \mathcal{E}(1^k, \mathsf{EVIEW}^{i-1}_{\mathsf{real}}, x^i_{hp}, x^i_{hu});$ $(\mathsf{tstate}^i, \mathsf{ustate}^i, y^i_s, y^i_p, y^i_u)$
    $\leftarrow \mathcal{C}^{\mathcal{S}'(\mathsf{ustate}^{i-1})}(\mathsf{tstate}^{i-1}, x^{j^i}_{hs}, x^{j^i}_{hp}, x^{j^i}_{hu}, x^i_{ap}, x^i_{au}) :$
    $(\mathsf{estate}^i, y^i_p, y^i_u)\}$
    $\mathsf{EVIEW}_{\mathsf{real}} = \mathsf{EVIEW}^i_{\mathsf{real}}$ *if* $\mathsf{stop}^i = \mathsf{TRUE}$.
    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 $\mathcal{E}$ does not have access. Then $\mathcal{E}$, based on its view from the last round,

1. *chooses 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* $\mathcal{C}^{\mathcal{S}'}$ *(note that* $\mathcal{E}$ *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 $\mathcal{C}^{\mathcal{S}'}$ *is run on the inputs* $(\mathsf{tstate}^{i-1}, x^{j^i}_{hs}, x^{j^i}_{hp}, x^{j^i}_{hu}, x^i_{ap}, x^i_{au})$, *where* tstate^{i-1} *is* $\mathcal{C}$*'s previously saved state, and produces a new state* tstate^i *for* $\mathcal{C}$*, as well as the secret* y^i_s*, protected* y^i_p *and unprotected* y^i_u *outputs. The oracle* $\mathcal{S}'$ *is given its previously saved state,* ustate^{i-1}*, as input, and the current state of* $\mathcal{S}'$ *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* $\mathcal{E}$ *in the real process is just its view in the last round (i.e.,* i *for which* stop^i $= \mathsf{TRUE}$*.).*

- *The* ideal *process:*
  $\mathsf{EVIEW}^i_{\mathsf{ideal}} = \{(\mathsf{istate}^i, x^i_{hs}, x^i_{hp}, x^i_{hu}) \leftarrow I(1^k, \mathsf{istate}^{i-1});$
  $(\mathsf{estate}, j^i, x^i_{ap}, x^i_{au}, \mathsf{stop}^i) \leftarrow E(1^k, \mathsf{EVIEW}^{i-1}_{\mathsf{ideal}}, x^i_{hp}, x^i_{hu});$
  $(\mathsf{astate}^i, y^i_s, y^i_p, y^i_u) \leftarrow \mathsf{Alg}(\mathsf{astate}^{i-1}, x^{j^i}_{hs}, x^{j^i}_{hp}, x^{j^i}_{hu}, x^i_{ap}, x^i_{au});$
  $(\mathsf{sstate}^i, \mathsf{ustate}^i, Y^i_p, Y^i_u, \mathsf{rep^i}) \leftarrow$
  $\mathsf{Sim}^{\mathcal{S}'(\mathsf{ustate}^{i-1})}_{\mathcal{E}}(\mathsf{sstate}^{i-1}, x^{j^i}_{hp}, x^{j^i}_{hu}, x^i_{ap}, x^i_{au}, y^i_p$
  $, y^i_u);\ (z^i_p, z^i_u) = \mathsf{rep^i}(Y^i_p, Y^i_u) + (1 - \mathsf{rep^i})(y^i_p, y^i_u) : (\mathsf{estate}, z^i_p, z^i_u)\}$
  $\mathsf{EVIEW}_{\mathsf{ideal}} = \mathsf{EVIEW}^i_{\mathsf{ideal}}$ *if* $\mathsf{stop}^i = \mathsf{TRUE}$.
  The ideal process also proceeds in rounds. In the ideal process, we have a stateful simulator $\mathsf{Sim}_{\mathcal{E}}$ *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* $\mathcal{S}'$*; moreover,* $\mathcal{S}'$ *saves its state as in the real experiment.*

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

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

  $$\mathsf{EVIEW}_{\mathsf{ideal}} = \mathsf{EVIEW}^i_{\mathsf{ideal}}\ \textit{if}\ \mathsf{stop}^i = \mathsf{TRUE}.$$

  In the ideal process, we have a stateful simulator $\mathsf{Sim}_{\mathcal{S}'}$ *who, equipped with only the unprotected inputs* (x^i_{hu}, x^i_{au})*, queries* $\mathcal{S}'$*. As before,* $\mathcal{S}'$ *may maintain state.*

Definition 3. *[23] (**α-efficient, secure outsourcing**) A pair of algorithms* $(\mathcal{C}, \mathcal{S})$ *is said to be an* α*-efficient implementation of* Alg *if*

1. $\mathcal{C}^{\mathcal{S}}$ *is a correct implementation of* Alg *and*
2. $\forall$ *inputs* x*, the running time of* $\mathcal{C}$ *is no more than an* α*-multiplicative factor of the running time of* Alg.

Definition 4. *[23] (**β-checkable, secure outsourcing**) A pair of algorithms* $(\mathcal{C}, \mathcal{S})$ *is said to be an* β*-checkable implementation of* Alg *if*

1. $\mathcal{C}^{\mathcal{S}}$ *is a correct implementation of* Alg *and*
2. $\forall$ *inputs* x*, if* $\mathcal{S}$ *deviates from its advertised functionality during the execution of* $\mathcal{C}^{\mathcal{S}'}(x)$*,* $\mathcal{C}$ *will detect the error with probability no less than* β.

Definition 5. *[23] (**(α,β)-outsource-security**) A pair of algorithms* $(\mathcal{C}, \mathcal{S})$ *is said to be an* (α,β)*-outsource-secure implementation of* Alg *if it is both* α*-efficient and* β*-checkable.*

3 Main Algorithm for Modular Exponentiation (Private-Base & Private-Exponent)

(a) Preliminaries

For almost all cryptographic applications, there are basically two different settings for which modular exponentiations are the most expensive parts of the cryptographic computation. The first one is the discrete logarithm problem (DLP) based and the second is the RSA problem based systems. In both cases, we summarize the following conditions to obtain mathematical problem instances which are intractable enough to obtain the desired level of security for the corresponding cryptographic schemes.

DLP case Let p and q be prime numbers and $G \subseteq \mathbb{F}_p^*$ be a subgroup generated by a primitive element g of order q. In order to have an intractable discrete logarithm problem on G, we impose the usual conditions on the number of distinct cosets in $\mathbb{F}_p^*/G$ being comparably small, i.e. we have a small cofactor $c = \frac{p-1}{q}$ (since otherwise by Chinese Remainder Theorem (Pohling-Hellman reduction) the complexity of DLP reduces to much smaller groups leading to less secure group based cryptographic systems [12]). This means that we need to hide the exponent of the exponentiation but not necessarily the base for the security of the encryption algorithms. On the other hand, hiding the base element in the modular exponentiation realizes the privacy preserved applications.

We restrict ourself to the multiplicative subgroup of prime field case $G \leq \mathbb{F}_p^*$, although it is also possible to use prime order multiplicative subgroups of the extension fields of $\mathbb{F}_p$. The main reason of our restriction is that the recent quasi-polynomial attacks on DLP of certain extension fields suggests not to use non-prime finite fields in cryptographic setting [3].

Remark 1. We note that all secure outsourcing algorithms for modular exponentiation (including the algorithms proposed in this paper) can easily be adapted to secure outsourcing algorithms for scalar multiplication of elliptic curve based cryptographic (ECC) schemes by using a prime order subgroup of $E(\mathbb{F}_p)$ instead of the group G. Using these secure outsourcing algorithms for scalar multiplication, one can also obtain hybrid privacy preserving outsourcing algorithms for paring-based encryption algorithms (e.g. by realization of ID-based cryptography [12]) by means of outsourcing possibly hided inputs of pairing functions, bilinearity property, and hided finite field exponentiations.

RSA case In this case, we have the modulus $n = p \cdot q$, where p and q are distinct large prime numbers. Since RSA based systems rely on the arithmetic of $G := (\mathbb{Z}/n\mathbb{Z})^*$, we have an exponent ranging 0 to $(p-1)(q-1)-1$. For public key encryptions the message must be hidden but the public key can be disclosed to the server, but for the signatures the inverse is true. However, similar to the DLP case, hiding the exponent or base element in the modular exponentiation enables the designers to obtain a privacy preserving outsourced cryptographic schemes. In particular, constructing a system which makes impossible for the server to distinguish between encryption and decryption processes might be an important design criteria for privacy preserving infrastructure (e.g., attribute based encryption schemes). To the best of our knowledge, there is only one algorithm proposed for RSA based modular exponentiation [30], which is non-checkable as explained in Section 1.1. Hence, our algorithm is the first which unifies modular exponentiation modulo a prime or a composite modulus.

(b) The Algorithm

We now propose our new main algorithm for modular exponentiation modulo n with the underlying group G which is either the subgroup of $\mathbb{F}_n^*$ or $(\mathbb{Z}/n\mathbb{Z})^*$ as above with order m. Note that n can be either a prime number or an RSA modulus covering the both cases described above. More precisely, the algorithm has the inputs $u \in G$ and $a \in \{0, \cdots, m-1\}$ and n, and it computes $u^a \bmod n$ without explicitly giving the values of u and a to the server. We note that in the case of DLP based system we have the extra condition that m is a prime number.

Let now three blinding factors $(x, g^x), (y, g^y), (t, g^t) \in \mathbb{Z}/m\mathbb{Z} \times G$ be given using a Rand Algorithm as defined in [23]. Note that these blinding factors are computed in order to speed up computations [11, 23]. The values $(x, g^x), (y, g^y)$ can be used several times for different exponents in order to hide the exponent whereas the value (t, g^t) should be used only once in order to hide the base element u.

For a given finite set $M = \{m_1, \cdots, m_n\}$, we denote further with $\mathbb{S}_n(M)$ the group of permutations on M. We identify any permutation on $\mathbb{S}_n(\{1, \cdots, n\})$ with a permutation on $\mathbb{S}_n(M)$ if it is necessary, where M is any finite set with n elements. By abuse of notation we also write $\sigma(m_i) = \sigma(i)$ for any $\sigma \in \mathbb{S}_n(\{1, \cdots, n\})$

Furthermore, we abbreviate by $\mathcal{C}$ the Client and by $\mathcal{S}$ the Server. We have also the assumption that $\mathcal{C}$ can run an algorithm to query u^a to $\mathcal{S}$. We denote the output of such a query with $\mathsf{Exp}(a, u)$. The algorithm is now given as follows:

Algorithm 1 ($\mathsf{Alg}_{\mathsf{pr}}^{\mathsf{pr}}$): Private-Base & Private-Exponent Modular Exponentiations

---

Input: $n, m, \ell, k \in \mathbb{N}$ and $(a, u) \in \mathbb{Z}/m\mathbb{Z} \times G$, where $G \leq \mathbb{F}_n^*$ or $G = (\mathbb{Z}/n\mathbb{Z})^*$.
Output: The value u^a in G, i.e. $u^a \bmod n$.
Precomputation: A Rand algorithm computes for $\mathcal{C}$ a primitive element g for DLP case or a random element g in RSA case and computes (x, g^x), (y, g^y), $(t, g^t) \in \mathbb{Z}/m\mathbb{Z} \times G$ with $v = g^x$ and $\mu = g^y$.

1. $\mathcal{C}$ computes $uv^{-1} \leftarrow w$, $ax - y \leftarrow z$ and runs $\mathsf{Exp}(zt^{-1}, g^t) \leftarrow Z$.
2. $\mathcal{C}$ chooses random elements $r, r_i, r'_i, a_j, a'_j \in \mathbb{Z}/m\mathbb{Z}$, $1 \leq i \leq \ell$, $1 \leq j \leq k$, sets $L_1 := \{r_1, \cdots, r_\ell, a_1, \cdots, a_k\}$ and $L_2 := \{r'_1, \cdots, r'_\ell, a'_1, \cdots, a'_k\}$ such that $r = \sum_{i=1}^{\ell} r_i = \sum_{i=1}^{\ell} r'_i$ and $a = \sum_{j=1}^{k} a_j = \sum_{j=1}^{k} a'_j$.
3. $\mathcal{C}$ chooses two random permutations $\sigma_1 \in \mathbb{S}_{\ell+k}(L_1), \sigma_2 \in \mathbb{S}_{\ell+k}(L_2)$ and two random elements $\alpha = (\alpha_1, \cdots, \alpha_{\ell+k}) \in \mathbb{F}_2^{\ell+k}, \beta = (\beta_1, \cdots, \beta_{\ell+k}) \in \mathbb{F}_2^{\ell+k}$ and stores positions $\Pi_1 = \{\pi_{r_1}, \cdots, \pi_{r_\ell}\}, \Pi_2 = \{\pi_{r'_1}, \cdots, \pi_{r'_\ell}\}$ of the values r_i, r'_i and $\Theta_1 = \{\alpha_j : \ \sigma_1(\alpha_j) = 1\}, \Theta_2 = \{\beta_j : \ \sigma_2(\beta_j) = 1\}$ corresponding to $\sigma_1(r_i), \sigma_2(r'_j)$, and the components of α, β having value 1, respectively.
4. $\mathcal{C}$ sets $1 \leftarrow W_1$, $1 \leftarrow W_2$ and $1 \leftarrow \mathcal{R}_1$, $1 \leftarrow \mathcal{R}_2$. For $i \in L_1$
   (a) If $\sigma_1(i) \in \Pi_1$:
      i. If $\sigma_1(i) \in \Theta_1$: $\mathcal{C}$ runs $\mathsf{Exp}(-\sigma_1(i), w) \leftarrow w_i^{(1)}$
         A. $W_1 \leftarrow W_1 \cdot w_i^{(1)}$
         B. $\mathcal{R}_1 \leftarrow \mathcal{R}_1 \cdot w_i^{(1)}$
      ii. If $\sigma_1(i) \notin \Theta_1$: $\mathcal{C}$ runs $\mathsf{Exp}(\sigma_1(i), w) \leftarrow w_i^{(1)}$
         A. $W_2 \leftarrow W_2 \cdot w_i^{(1)}$
         B. $\mathcal{R}_2 \leftarrow \mathcal{R}_2 \cdot w_i^{(1)}$
   (b) Else
      i. If $\sigma_1(i) \in \Theta_1$: $\mathcal{C}$ runs $\mathsf{Exp}(-\sigma_1(i), w) \leftarrow w_i^{(1)}$
         A. $W_1 \leftarrow W_1 \cdot w_i^{(1)}$
      ii. If $\sigma_1(i) \notin \Theta_1$: $\mathcal{C}$ runs $\mathsf{Exp}(\sigma_1(i), w) \leftarrow w_i^{(1)}$
         A. $W_2 \leftarrow W_2 \cdot w_i^{(1)}$
5. $\mathcal{C}$ sets $1 \leftarrow W'_1$, $1 \leftarrow W'_2$ and $1 \leftarrow \mathcal{R}'_1$, $1 \leftarrow \mathcal{R}'_2$. For $j \in L_2$
   (a) If $\sigma_2(j) \in \Pi_2$:
      i. If $\sigma_2(j) \in \Theta_2$: $\mathcal{C}$ runs $\mathsf{Exp}(-\sigma_2(j), w) \leftarrow w_j^{(2)}$
         A. $W'_1 \leftarrow W'_1 \cdot w_j^{(2)}$
         B. $\mathcal{R}'_1 \leftarrow \mathcal{R}'_1 \cdot w_j^{(2)}$
      ii. If $\sigma_2(j) \notin \Theta_2$: $\mathcal{C}$ runs $\mathsf{Exp}(\sigma_2(j), w) \leftarrow w_j^{(2)}$
         A. $W'_2 \leftarrow W'_2 \cdot w_j^{(2)}$
         B. $\mathcal{R}'_2 \leftarrow \mathcal{R}'_2 \cdot w_j^{(2)}$
   (b) Else
      i. If $\sigma_2(j) \in \Theta_2$: $\mathcal{C}$ runs $\mathsf{Exp}(-\sigma_2(j), w) \leftarrow w_j^{(2)}$
         A. $W'_1 \leftarrow W'_1 \cdot w_j^{(2)}$
      ii. If $\sigma_2(j) \notin \Theta_2$: $\mathcal{C}$ runs $\mathsf{Exp}(\sigma_2(j), w) \leftarrow w_j^{(2)}$
         A. $W'_2 \leftarrow W'_2 \cdot w_i^{(1)}$

6. If

$$W_1^{-1} \cdot W_2 \neq W_1'^{-1} \cdot W_2' \text{ or}$$
$$\mathcal{R}_1 \cdot \mathcal{R}_2^{-1} \neq \mathcal{R}_1' \cdot \mathcal{R}_2'^{-1}$$

   Return *Checkability fails*
7. Else Return $\mu \cdot Z \cdot (W_1^{-1} \cdot W_2) \cdot (\mathcal{R}_1 \cdot \mathcal{R}_2^{-1})$

Correctness and Termination.

Theorem 1. $\mathsf{Alg}_{\mathsf{pr}}^{\mathsf{pr}}$ *terminates and outputs correctly.*

Proof. Precomputation and Step 1 of $\mathsf{Alg}_{\mathsf{pr}}^{\mathsf{pr}}$ says that

$$\begin{aligned} u^a &= (vw)^a \\ &= g^{xa} w^a \\ &= \mu g^z w^a \\ &= \mu Z w^a, \end{aligned}$$

where $w = uv^{-1}$ and $z = ax - y$.
Let A_1, A_1' be the sum of a_i, a_i''s whose corresponding values in the components of α, β are 1, respectively. Let further $A_2 = a - A_1$ and $A_2' = a - A_1'$. Similarly, we set U_1, U_1' be the sum of r_i, r_i''s whose corresponding values in the components of α, β are 1, respectively. Let further $U_2 = r - U_1$ and $U_2' = r - U_1'$.

After the preparation of Step 2 and 3 and using the query results of $\mathcal{S}$, the client $\mathcal{C}$ computes the following for $i \in L_1$ in Step 4:

$$\begin{aligned} W_1 &= \prod_{\sigma_1(i) \in \Theta_1} w_i^{(1)} \\ &= \prod_{\sigma_1(i) \in \Theta_1} w^{-\sigma_1(i)} \\ &= w^{-\sum_{\sigma_1(i) \in \Theta_1} \sigma_1(i)} \\ &= w^{-U_1 - A_1}, \end{aligned}$$

$$\begin{aligned} W_2 &= \prod_{\sigma_1(i)\notin\Theta_1} w_i^{(1)} \\ &= \prod_{\sigma_1(i)\notin\Theta_1} w^{\sigma_1(i)} \\ &= w^{\sum_{\sigma_1(i)\notin\Theta_1}\sigma_1(i)} \\ &= w^{U_2+A_2}. \end{aligned}$$

Moreover, $\mathcal{C}$ computes similarly
$R_1 = w^{-U_1}$, $R_2 = w^{U_2}$.
Analogously, $\mathcal{C}$ computes for $j \in L_2$ in Step 5:

$$\begin{aligned} W_1' &= \prod_{\sigma_2(j)\in\Theta_2} w_j^{(2)} \\ &= \prod_{\sigma_2(j)\in\Theta_2} w^{-\sigma_2(j)} \\ &= w^{-\sum_{\sigma_2(j)\in\Theta_2}\sigma_1(i)} \\ &= w^{-U_1'-A_1'}, \end{aligned}$$

$$\begin{aligned} W_2' &= \prod_{\sigma_2(j)\notin\Theta_2} w_j^{(2)} \\ &= \prod_{\sigma_2(j)\notin\Theta_2} w^{\sigma_2(j)} \\ &= w^{\sum_{\sigma_2(j)\notin\Theta_2}\sigma_2(j)} \\ &= w^{U_2'+A_2'}. \end{aligned}$$

$R_1' = w^{-U_1'}$, $R_2' = w^{U_2'}$.

Step 6 simply verifies that if $\mathcal{S}$ runs the algorithm correctly by checking the following equalities:

$$\begin{aligned}
W_1^{-1} \cdot W_2 &= w^{U_1+A_1} w^{U_2+A_2} \\
&= w^{U_1+A_1+U_2+A_2} \\
&= w^{r+a} \\
&= w^{U_1'+A_1'+U_2'+A_2'} \\
&= w^{U_1'+A_1'} w^{U_2'+A_2'} \\
&= W_1'^{-1} \cdot W_2',
\end{aligned}$$

and

$$\begin{aligned}
R_1 \cdot R_2^{-1} &= w^{-U_1} w^{-U_2} \\
&= w^{-U_1-U_2} \\
&= w^{-r} \\
&= w^{-U_1'-U_2'} \\
&= w^{-U_1'} w^{-U_2'} \\
&= R_1' \cdot R_2'^{-1}.
\end{aligned}$$

If the equalities do not hold then algorithm outputs checkability failure. If $\mathcal{S}$ runs the query algorithm properly then the algorithm ends with Step 7 as follows:

$$\begin{aligned}
\mu \cdot Z \cdot (W_1^{-1} \cdot W_2) \cdot (\mathcal{R}_1 \cdot \mathcal{R}_2^{-1}) &= g^y \cdot g^{ax-y} \cdot w^{r+a} \cdot w^{-r} \\
&= g^{ax} \cdot w^a \\
&= (g^x \cdot w)^a = (v \cdot w)^a \\
&= u^a.
\end{aligned}$$

Security and Checkability. Assume that a client $\mathcal{C}$ would like to outsource $u^a \mod n$ where u and a are private and n is public. In this part, we give the security analysis of $\mathsf{Alg}_{\mathsf{pr}}^{\mathsf{pr}}$ and show that a malicious server cannot be able to get any valuable information about u and a.

The next lemma gives the probability that a malicious server obtains the exponent.

Lemma 1. *A malicious server $\mathcal{S}'$ learns the exponent a with probability at most $\left(\frac{k}{2(\ell+k)}\right)^{2k}$.*

Proof. The output will only be disclosed if $\mathcal{S}'$ obtains exactly the same permutation $\sigma_1, \sigma_2 \in \mathbb{S}_{\ell+k}$, and finds $\alpha, \beta \in \mathbb{F}_2^{k+\ell}$. And, the probability of this event is $\left(1/\left(\binom{\ell+k}{k} \cdot 2^k\right)\right)^2$. To conclude, we only need show that $\binom{\ell+k}{k} \geq (\frac{\ell+k}{k})^k$.

We have[1]

$$\binom{\ell+k}{k} = \frac{\ell+k}{k} \cdot \prod_{i=1}^{k-1} \frac{\ell+k-i}{k-i} \geq \left(\frac{\ell+k}{k}\right)^k.$$

Hence, $\mathcal{S}'$ cannot distinguish the two test queries from all of the $k+\ell$ queries that $\mathcal{C}$ makes, and during any execution of $\mathsf{Alg}_{\mathsf{pr}}^{\mathsf{pr}}$ $\mathcal{S}'$ can successfully cheat without being detected with probability at most $\left(\frac{k}{2(\ell+k)}\right)^{2k}$. For $\ell = k$, the probability becomes at most $(\frac{1}{2})^{4k}$ (e.g., letting $\ell = k = 20$ the probability becomes negligible).

Now, we prove the security of $\mathsf{Alg}_{\mathsf{pr}}^{\mathsf{pr}}$. Note that outsource-secure informally means that there exists a simulator which simulates the view of the adversary in a real algorithm run. This means that the adversary obtains no relevant information from the real run since it could output any result from what it knows by itself.

Theorem 2. *The algorithms $(\mathcal{C}, \mathcal{S})$ are an outsource-secure implementation of $\mathsf{Alg}_{\mathsf{pr}}^{\mathsf{pr}}$, where the input (a, u) may be honest secret; or honest protected; or adversarial protected.*

Proof. We note that this proof is inspired from the proof of the security analysis of [23]. Let $\mathcal{A} = (\mathcal{E}, \mathcal{S}')$ be a probabilistic polynomial-time (PPT) adversary interacting with a PPT-based algorithm $\mathcal{C}$ in the outsource-security model.

Firstly, we prove $\mathsf{EVIEW}_{\mathsf{real}} \sim \mathsf{EVIEW}_{\mathsf{ideal}}$. (Pair One– The external adversary $\mathcal{E}$ learns nothing.)

Let (a, u) be a private input of an honest party. Assume that $\mathsf{Sim}_{\mathcal{E}}$ is a PPT simulator which acts as follows. $\mathsf{Sim}_{\mathcal{E}}$ ignores the the ith round when getting input, and instead first chooses two permutations $\sigma_1, \sigma_2 \in \mathbb{S}_{\ell+k}$ and two bit-strings $\alpha = (\alpha_1, \cdots, \alpha_{k+\ell}), \beta = (\beta_1, \cdots, \beta_{k+\ell}) \in \mathbb{F}_2^{k+\ell}$, and then prepares two signed permuted random queries of the form $((-1)^{\alpha_j}\sigma_1(\lambda_j^{(1)}), \gamma_j)$, $((-1)^{\beta_j}\sigma_2(\lambda_j^{(2)}), \gamma_j) \in \mathbb{Z}/m\mathbb{Z} \times G$ to $\mathcal{S}'$ where $j \in$

[1] Note that for any m, n $m \leq n \iff (n-i) \cdot m \geq (m-i) \cdot n \; \forall \; 1 \leq i \leq m-1$.

$\{1, \ldots, k+\ell\}$. $\mathsf{Sim}_{\mathcal{E}}$ randomly checks $(k+\ell)$ outputs from each procedure using $\sigma_1, \sigma_2, \alpha_j, \beta_j$ where $1 \leq j \leq k+\ell$ (i.e., $\gamma_j^{\lambda_j^{(1)}}$ and $\gamma_j^{\lambda_j^{(2)}}$). If an error occurs, $\mathsf{Sim}_{\mathcal{E}}$ stores its own and $\mathcal{S}'$'s states and outputs $Y_p^i =$ "$error$"$, Y_u^i = \emptyset, \mathsf{rep^i} = 1$. If all checkability steps are valid, $\mathsf{Sim}_{\mathcal{E}}$ outputs $Y_p^i = \emptyset$, $Y_u^i = \emptyset$, $\mathsf{rep^i} = 0$; otherwise, $\mathsf{Sim}_{\mathcal{E}}$ chooses a random group value $r \in_R G$ and outputs $Y_p^i = r$, $Y_u^i = \emptyset$, $\mathsf{rep^i} = 1$. Next, $\mathsf{Sim}_{\mathcal{E}}$ stores the corresponding states. The distributions in the real and ideal executions of the input to $\mathcal{S}'$ are computationally indistinguishable. In the ideal setting, the inputs are uniformly chosen random from $\mathbb{Z}/m\mathbb{Z} \times G$. In the real setting, we follow steps 4 and 5 of $\mathsf{Alg}_{\mathsf{pr}}^{\mathsf{pr}}$ to assure that all parts of Exp $\mathcal{C}$ invokes is randomized independently using σ_1, σ_2 and α, β. Now, we consider all possible cases. If $\mathcal{S}'$ behaves in an honest manner in the ith round, then $\mathsf{EVIEW}_{\mathsf{real}}^i \sim \mathsf{EVIEW}_{\mathsf{ideal}}^i$, because in the real execution $\mathcal{C}^{\mathcal{S}'}$ perfectly runs $\mathsf{Alg}_{\mathsf{pr}}^{\mathsf{pr}}$ and in the ideal execution $\mathsf{Sim}_{\mathcal{E}}$ does not change the output of $\mathsf{Alg}_{\mathsf{pr}}^{\mathsf{pr}}$. If $\mathcal{S}'$ gives a wrong output in the ith round, then the output will be detected by $\mathcal{C}$ and $\mathsf{Sim}_{\mathcal{E}}$ with probability at most $\left(\frac{k}{2(\ell+k)}\right)^{2k}$ due to Lemma 1, resulting in an output of "error"; otherwise, the software will indeed be successful in manipulating the output of $\mathsf{Alg}_{\mathsf{pr}}^{\mathsf{pr}}$.

In the real execution, the $k+\ell$ real outputs of $\mathcal{S}'$ are firstly grouped into two parts corresponding to their signs (positive or negative). The negative part will be multiplied together and inverted at the end. The result will be multiplied together with each element of the positive part. The same procedure is repeated again due to checkability (see step 4,5 and 6 of $\mathsf{Alg}_{\mathsf{pr}}^{\mathsf{pr}}$). At the last step, we multiply the overall result with the masking values of the base element generated at the first step. Hence, a manipulated output of $\mathsf{Alg}_{\mathsf{pr}}^{\mathsf{pr}}$ will seem to be wrong, but random to $\mathcal{E}$.

We simulate this situation in the ideal execution by replacing the output of $\mathsf{Alg}_{\mathsf{pr}}^{\mathsf{pr}}$ with a random element in G when there is an attempt to behave maliciously by $\mathcal{S}'$ which would not be detected by $\mathcal{C}$ in the real execution. Hence, even if $\mathcal{S}'$ behaves maliciously in the ith round, $\mathsf{EVIEW}_{\mathsf{real}}^i \sim \mathsf{EVIEW}_{\mathsf{ideal}}^i$. By the hybrid argument, we conclude that $\mathsf{EVIEW}_{\mathsf{real}} \sim \mathsf{EVIEW}_{\mathsf{ideal}}$.

Next, we prove $\mathsf{EVIEW}_{\mathsf{real}} \sim \mathsf{EVIEW}_{\mathsf{ideal}}$. (Pair Two– The untrusted server $\mathcal{S}'$ obtains no useful information).

We now consider the cases where (a, u) is honest secret/protected or adversarial protected. Let $\mathsf{Sim}_{\mathcal{S}'}$ be a PPT simulator that acts in the following manner. $\mathsf{Sim}_{\mathcal{S}'}$ ignores the the ith round when getting input, and instead chooses two permutations $\sigma_1, \sigma_2 \in \mathbb{S}_{\ell+k}$ and prepares two signed permuted random queries of the form $((-1)^{\alpha_j}\sigma_1(\lambda_j^{(1)}),\gamma_j) \in \mathbb{Z}/m\mathbb{Z} \times G$ to $\mathcal{S}'$ using $\sigma_1, \sigma_2, \alpha_j, \beta_j$ where $j \in \{1,\ldots,k+\ell\}$. $\mathsf{Sim}_{\mathcal{E}}$ randomly checks $(k+\ell)$ outputs from each procedure using σ_1, σ_2 (i.e., $\gamma_j^{\lambda_j^{(1)}}$ and $\gamma_j^{\lambda_j^{(2)}}$). Then, $\mathsf{Sim}_{\mathcal{S}'}$ stores its own and $\mathcal{S}'$'s states. Note that these real and ideal executions are distinguishable by $\mathcal{E}$ but $\mathcal{E}$ cannot use this information to $\mathcal{S}'$ (e.g., the output of the ideal execution is never manipulated). During the ith round of the real execution, the inputs of $\mathcal{C}$) are always randomized to $2(k+\ell)$ utilizing $\sigma_1, \sigma_2, \alpha, \beta$ (see steps 4 and 5 of $\mathsf{Alg}_{\mathsf{pr}}^{\mathsf{pr}}$). In the ideal execution, $\mathsf{Sim}_{\mathcal{S}'}$ always generates independently random queries for $\mathcal{S}'$. The view is consistent and indistinguishable from the server's view when there is an interaction with honest $\mathcal{C}$. Thus, for each round we have $\mathsf{EVIEW}_{\mathsf{real}} \sim \mathsf{EVIEW}_{\mathsf{ideal}}$, which by the hybrid argument yields $\mathsf{EVIEW}_{\mathsf{real}} \sim \mathsf{EVIEW}_{\mathsf{ideal}}$.

Consequently, we simulate every step of $\mathsf{Alg}_{\mathsf{pr}}^{\mathsf{pr}}$ for the simulator which completes the simulation for both malicious environment and server.

Lemma 2. *The algorithm $(\mathcal{C},\mathcal{S})$ is an $O(\log^2(l)/l)$-efficient implementation of* $\mathsf{Alg}_{\mathsf{pr}}^{\mathsf{pr}}$*, where l denotes the number of bits of the exponent a.*

Proof. We use the same approach of the proof of the algorithm in [23]. The proposed algorithm $\mathsf{Alg}_{\mathsf{pr}}^{\mathsf{pr}}$ makes 3 calls to Rand, $4\ell+2k+10$ modular multiplications (MMs) and 6 modular inverses (MInvs) in order to compute $u^a \mod n$ (other operations like modular additions are omitted). Also, a server aided exponentiation takes $O(\log^2(l))$ MMs using the number theoretic complexity analysis of Nguyen, Shparlinski, and Stern [33], or $O(1)$ MMs if a table-lookup method is used. On the other hand, it takes in avarage $1.5l$ MMs to compute $u^a \mod n$ by the classical square-and-multiply method. Thus, the algorithm $(\mathcal{C},\mathcal{S})$ is an $O(\log^2 l/l)$-efficient implementation of $\mathsf{Alg}_{\mathsf{pr}}^{\mathsf{pr}}$.

Lemma 3. *The algorithm $(\mathcal{C},\mathcal{S})$ is an $\left(1-(k/2(k+\ell))^2\right)$-checkable implementation of* $\mathsf{Alg}_{\mathsf{pr}}^{\mathsf{pr}}$.

Proof. By Alg_{pr}^{pr}, a malicious server $\mathcal{S}$ gives a wrong result without being detected if it can find any position of a_i's and a_i''s and their corresponding signs according to the components of α and β, respectively. Hence, $\mathcal{S}$ finds with probability $k/(k+\ell)$ an a_i and with probability $1/2$ to decide whether it has negative or positive sign. The same holds for an a_i' and its sign. Therefore, the overall probability for a malicious server $\mathcal{S}$ to declare a wrong value without being detected is $k^2/4(k+\ell)^2$.

Now security and checkability of $\mathsf{Alg}_{\mathsf{pr}}^{\mathsf{pr}}$ follow obviously from the following corollary:

Corollary 1. *The algorithm* $(\mathcal{C}, \mathcal{S})$ *is an* $\left(O\left(\log^2(l)/l\right), (1-(k/2(k+\ell))^2\right)$*-outsource-secure implementation of* $\mathsf{Alg}_{\mathsf{pr}}^{\mathsf{pr}}$.

Table 1. Computation Complexity of the Proposed Algorithms Using Single Server

	Exp. ($\mathcal{S}$)	Modular Mult.	Modular Inv.	Rand	Checkability
Alg_{pr}^{pr}	$2(\ell+k)+1$	$4\ell+2k+10$	6	3	$1-(\frac{k}{2(k+\ell)})^2$
Alg_{pb}^{pr}	$2(\ell+k)$	$4\ell+2k+5$	4	0	$\frac{15}{16}$
Alg_{pr}^{pb}	4	12	3	5	$1-\frac{1}{m}$
t-Sim-$\mathsf{Alg}_{\mathsf{pr}}^{\mathsf{pr}}$	$2t(\ell+k)+1$	$4\ell t+2kt+7t+2$	$4t+1$	3	$1-(\frac{k}{2(k+\ell)})^{2t}$

Remark 2. Letting $k = \ell$ gives us the probability 15/16 by Lemma 3 which is the best checkability result compared to previous works [11, 23, 26]. We highlight that this is more than a reasonable result for possible real-world applications. Because each request is independent of each other. Sending approximately 20 wrong results to the client $\mathcal{C}$ makes the probability of not being detected to negligibly small ($\approx 1/2^{80}$).

Table 2. Computation Complexity for Proposed Algorithms for $k = \ell = 20$

	Exp. ($\mathcal{S}$)	Modular Mult.	Modular Inv.	Rand	Checkability
$\mathsf{Alg}_{\mathsf{pr}}^{\mathsf{pr}}$	81	130	6	3	15/16
$\mathsf{Alg}_{\mathsf{pb}}^{\mathsf{pr}}$	80	125	4	0	15/16
$\mathsf{Alg}_{\mathsf{pr}}^{\mathsf{pb}}$	4	12	4	5	$1-1/m$
2-Sim-$\mathsf{Alg}_{\mathsf{pr}}^{\mathsf{pr}}$	161	256	9	3	255/256

Note that in outsourced computation model the malicious server $\mathcal{S}$ can be seen as a covert adversary [2], which may arbitrarily behave to cheat depending on whether being detected with reasonable probability (not necessarily with very high probability) by an honest party. In [2], covert adversaries are described for many real-life scenarios where they are always eager to cheat but only if they are not detected. Therefore, cloud servers can be seen as covert adversaries in outsourced computation setting because their financial and reputational interests deter them from cheating.

4 Other Relevant Algorithms

In this section, we simplify $\mathsf{Alg}_{\mathsf{pr}}^{\mathsf{pr}}$ for Public-Base & Private-Exponent and Private-Base & Public-Exponent cases, and modify it to obtain a more efficient simultaneous modular exponentiations algorithm.

(a) Public-Base & Private-Exponent

In this part, we modify $\mathsf{Alg}_{\mathsf{pr}}^{\mathsf{pr}}$ for the case of public-base & private-exponent. The modified method is especially designed to outsource the cryptographic outsourced computation for the cases in which there is no need to hide the base element if there exists waived privacy needs in the cryptographic setting (e.g., signatures). The first precomputation of $\mathsf{Alg}_{\mathsf{pr}}^{\mathsf{pr}}$ is unnecessary in this case since we are not forced to hide our base element u. We denote this algorithm by $\mathsf{Alg}_{\mathsf{pb}}^{\mathsf{pr}}$, and it is given as follows:

Algorithm 2 ($\mathsf{Alg}_{\mathsf{pb}}^{\mathsf{pr}}$): Public-Base & Private-Exponent Modular Exponentiations

Input: $n, m, \ell, k \in \mathbb{N}$ and $(a, u) \in \mathbb{Z}/m\mathbb{Z} \times G$, where $G \leq \mathbb{F}_n^*$ or $G = (\mathbb{Z}/n\mathbb{Z})^*$.
Output: The value u^a in G, i.e. $u^a \bmod n$.

1. $\mathcal{C}$ chooses random elements $r, r_i, r'_i, a_j, a'_j \in \mathbb{Z}/m\mathbb{Z}$, $1 \leq i \leq \ell,\ 1 \leq j \leq k$, sets $L_1 := \{r_1, \cdots, r_\ell, a_1, \cdots, a_k\}$ and $L_2 := \{r'_1, \cdots, r'_\ell, a'_1, \cdots, a'_k\}$ such that $r = \sum_{i=1}^{\ell} r_i = \sum_{i=1}^{\ell} r'_i$ and $a = \sum_{j=1}^{k} a_j = \sum_{j=1}^{k} a'_j$.
2. $\mathcal{C}$ chooses two random permutations $\sigma_1 \in \mathbb{S}_{\ell+k}(L_1), \sigma_2 \in \mathbb{S}_{\ell+k}(L_2)$ and two random elements $\alpha = (\alpha_1, \cdots, \alpha_{\ell+k}) \in \mathbb{F}_2^{\ell+k}, \beta = (\beta_1, \cdots, \beta_{\ell+k}) \in \mathbb{F}_2^{\ell+k}$ and stores positions $\Pi_1 = \{\pi_{r_1}, \cdots, \pi_{r_\ell}\}, \Pi_2 = \{\pi_{r'_1}, \cdots, \pi_{r'_\ell}\}$ of the values r_i, r'_i and $\Theta_1 = \{\alpha_j :\ \sigma_1(\alpha_j) = 1\}, \Theta_2 = \{\beta_j :\ \sigma_2(\beta_j) = 1\}$ corresponding to $\sigma_1(r_i), \sigma_2(r'_j)$, and the components of $\alpha,\ \beta$, respectively.

```
3. $\mathcal{C}$ sets $1 \leftarrow U_1$, $1 \leftarrow U_2$ and $1 \leftarrow \mathcal{R}_1$, $1 \leftarrow \mathcal{R}_2$. For $i \in L_1$
   (a) If $\sigma_1(i) \in \Pi_1$:
       i. If $\sigma_1(i) \in \Theta_1$: $\mathcal{C}$ runs $\mathsf{Exp}(-\sigma_1(i), u) \leftarrow u_i^{(1)}$
          A. $U_1 \leftarrow U_1 \cdot u_i^{(1)}$
          B. $\mathcal{R}_1 \leftarrow \mathcal{R}_1 \cdot u_i^{(1)}$
       ii. If $\sigma_1(i) \notin \Theta_1$: $\mathcal{C}$ runs $\mathsf{Exp}(\sigma_1(i), u) \leftarrow u_i^{(1)}$
          A. $U_2 \leftarrow U_2 \cdot u_i^{(1)}$
          B. $\mathcal{R}_2 \leftarrow \mathcal{R}_2 \cdot u_i^{(1)}$
   (b) Else
       i. If $\sigma_1(i) \in \Theta_1$: $\mathcal{C}$ runs $\mathsf{Exp}(-\sigma_1(i), u) \leftarrow u_i^{(1)}$
          A. $U_1 \leftarrow U_1 \cdot u_i^{(1)}$
       ii. If $\sigma_1(i) \notin \Theta_1$: $\mathcal{C}$ runs $\mathsf{Exp}(\sigma_1(i), u) \leftarrow u_i^{(1)}$
          A. $U_2 \leftarrow U_2 \cdot u_i^{(1)}$
4. $\mathcal{C}$ sets $1 \leftarrow U_1'$, $1 \leftarrow U_2'$ and $1 \leftarrow \mathcal{R}_1'$, $1 \leftarrow \mathcal{R}_2'$. For $j \in L_2$
   (a) If $\sigma_2(j) \in \Pi_2$:
       i. If $\sigma_2(j) \in \Theta_2$: $\mathcal{C}$ runs $\mathsf{Exp}(-\sigma_2(j), u) \leftarrow u_j^{(2)}$
          A. $U_1' \leftarrow U_1' \cdot u_j^{(2)}$
          B. $\mathcal{R}_1' \leftarrow \mathcal{R}_1' \cdot u_j^{(2)}$
       ii. If $\sigma_2(j) \notin \Theta_2$: $\mathcal{C}$ runs $\mathsf{Exp}(\sigma_2(j), u) \leftarrow u_j^{(2)}$
          A. $U_2' \leftarrow U_2' \cdot u_j^{(2)}$
          B. $\mathcal{R}_2' \leftarrow \mathcal{R}_2' \cdot u_j^{(2)}$
   (b) Else
       i. If $\sigma_2(j) \in \Theta_2$: $\mathcal{C}$ runs $\mathsf{Exp}(-\sigma_2(j), u) \leftarrow u_j^{(2)}$
          A. $U_1' \leftarrow U_1' \cdot u_j^{(2)}$
       ii. If $\sigma_2(j) \notin \Theta_2$: $\mathcal{C}$ runs $\mathsf{Exp}(\sigma_2(j), u) \leftarrow u_j^{(2)}$
          A. $U_2' \leftarrow U_2' \cdot w_i^{(1)}$
5. If

   $$U_1^{-1} \cdot U_2 \neq U_1'^{-1} \cdot U_2' \text{ or}$$
   $$\mathcal{R}_1 \cdot \mathcal{R}_2^{-1} \neq \mathcal{R}_1' \cdot \mathcal{R}_2'^{-1}$$

   Return *Checkability fails*
6. Else Return $(U_1^{-1} \cdot U_2) \cdot (\mathcal{R}_1 \cdot \mathcal{R}_2^{-1})$
```

Theorem 3. $\mathsf{Alg}_{\mathsf{pb}}^{\mathsf{pr}}$ *terminates and outputs correctly. Furthermore, there exists an algorithm which is an* $\Big(O(\log^2(l)/l), (1-(\frac{k}{2(k+\ell)})^2)\Big)$*-outsource-secure implementation of* $\mathsf{Alg}_{\mathsf{pb}}^{\mathsf{pr}}$.

Proof. Correctness, termination, security and checkability of the algorithm follow easily as a corollary of the results for $\mathsf{Alg}_{\mathsf{pr}}^{\mathsf{pr}}$.

(b) Private-Base & Public-Exponent

In this part, we give another algorithm for private-base & public-exponent cryptographic computation by modifying $\mathsf{Alg}_{\mathsf{pr}}^{\mathsf{pr}}$. Note that especially for public-key encryption or signature verification based systems it could be desirable to have private-base & public-exponent. This algorithm is denoted by $\mathsf{Alg}_{\mathsf{pr}}^{\mathsf{pb}}$ which works in detail as follows:

Algorithm 3 ($\mathsf{Alg}_{\mathsf{pr}}^{\mathsf{pb}}$): Private-Base & Public-Exponent Modular Exponentiations

Input: $n, m \in \mathbb{N}$ and $(a, u) \in \mathbb{Z}/m\mathbb{Z} \times G$, where $G \leq \mathbb{F}_p^*$ or $G = (\mathbb{Z}/n\mathbb{Z})^*$.
Output: The value u^a in G, i.e. $u^a \bmod n$.
Precomputation: A Rand algorithm computes for $\mathcal{C}$ a primitive element g for DLP case or a random element g for RSA case and computes $(x_1, g^{x_1}), (x_1, g^{x_1}), (y_1, g^{y_1}), (y_2, g^{y_2}), (t, g^t) \in \mathbb{Z}/m\mathbb{Z} \times G$ with $v_i = g^{x_i}$ and $\mu_i = g^{y_i}$ for $1 \leq i \leq 2$.

```
1. $\mathcal{C}$ computes $uv_1^{-1} \leftarrow w_1$, $ax_1 - y_1 \leftarrow z_1$ and runs $\mathsf{Exp}(z_1t^{-1}, g^t) \leftarrow Z_1$.
2. $\mathcal{C}$ computes $uv_2^{-1} \leftarrow w_2$, $ax_2 - y_2 \leftarrow z_2$ and runs $\mathsf{Exp}(z_2t^{-1}, g^t) \leftarrow Z_2$.
3. $\mathcal{C}$ runs $\mathsf{Exp}(a, w_1) \leftarrow W_1$
4. $\mathcal{C}$ runs $\mathsf{Exp}(a, w_2) \leftarrow W_2$
5. If $\mu_1 \cdot g^{z_1} \cdot W_1 \neq \mu_2 \cdot g^{z_2} \cdot W_2$ Return *Checkability fails*
6. Else Return $\mu_1 \cdot g^{z_1} \cdot W_1$
```

Theorem 4. $\mathsf{Alg}_{\mathsf{pr}}^{\mathsf{pb}}$ *terminates and outputs correctly. Furthermore, there exists an algorithm which is an* $(O(\log^2(l)/l), 1-\frac{1}{m})$*-outsource-secure implementation of* $\mathsf{Alg}_{\mathsf{pr}}^{\mathsf{pb}}$.

Proof. Correctness, termination, security and checkability of the algorithm follow analogously as in the proofs for $\mathsf{Alg}_{\mathsf{pr}}^{\mathsf{pr}}$.

Remark 3. For real-life applications, m is typically chosen as a 2048-bit number for RSA or DLP based systems and as a 384-bit number for ECC based systems.

Table 3. Comparison with Previous Results

	Modular Mult.	Modular Inv.	Single Server	Checkability	Total cost (MMs) after equating the checkability to 15/16
[23]	9	5	✗	1/2	$4 \cdot (9 + 5 \cdot 100) \approx 2036$
[11]	7	3	✗	2/3	$2,52 \cdot (7 + 3 \cdot 100) \approx 767$
Ours	130	6	✓	15/16	$130 + 6 \cdot 100 \approx 730$

(c) t-Simultaneous Modular Exponentiations

In this part, we generalize the notion of simultaneous modular exponentiation method of [11] to the notion of *t-simultaneous modular exponentiations* $u_1^{a_1} \cdots u_t^{a_t}$ in the group G for $t \in \mathbb{N}$. t-simultaneous modular exponentiations are extensively used in many real-life cryptographic schemes including [8, 13, 14, 17, 20, 36]. As described in [11], computing 2-simultaneous modular exponentiations is trivial by simply invoking $\mathsf{Alg}_{\mathsf{pr}}^{\mathsf{pr}}$ twice. Here, we show that it is possible to reduce the computation cost significantly for a generalized t-simultaneous setting by improving the method of [11] and utilizing only one untrusted server (instead of two servers one of which is assumed to be honest). We denote by t-Sim-$\mathsf{Alg}_{\mathsf{pr}}^{\mathsf{pr}}$ for t-simultaneous modular exponentiation algorithm.

The scheme of Chen *et al.* [11] has probability of 2/3 for checkability in modular exponentiation utilizing and has probability 1/2 for 2-Sim-$\mathsf{Alg}_{\mathsf{pr}}^{\mathsf{pr}}$ using two non-colluding servers. They simply add a one more variable on the exponentiation at the expense of reducing the probability from 2/3 to 1/2. Our solution has the probability $1 - \frac{1}{256}$ for checkability and utilizes only one single untrusted server.

We further emphasize that the natural generalization for 2-simultaneous modular exponentiation method in [11] reduces the checkability probability from $\frac{1}{2}$ of single exponentiation case to $\frac{2}{t+2}$ for t-simultaneous modular exponentiations. However, the use of t-simultaneous modular exponentiation in real-life protocols, like anonymous credentials [8], causes significant complexity overhead. Hence, this reduction hinders the use of this generalization from 2-simultaneous to t-simultaneous modular exponentiation. Unlike the scheme in [11], our scheme enhances the probability from $1-(\frac{k}{2(k+\ell)})^2$ to $1-(\frac{k}{2(k+\ell)})^{2t}$. More concretely, the algorithm works as follows:

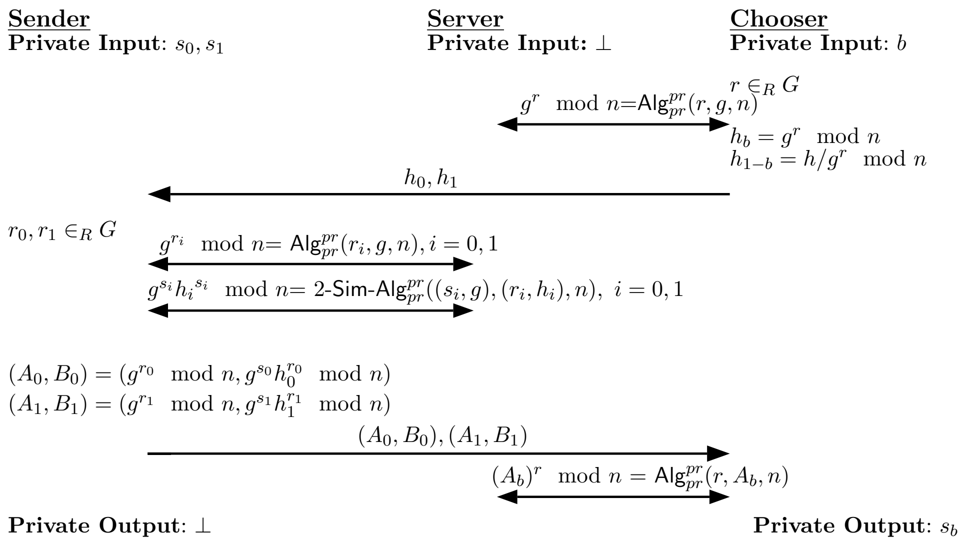
Sender
Private Input: s_0, s_1
Server
Private Input: $\perp$
Chooser
Private Input: b
$r \in_R G$
$g^r \mod n = \mathsf{Alg}_{pr}^{pr}(r, g, n)$
$h_b = g^r \mod n$
$h_{1-b} = h/g^r \mod n$
h_0, h_1
$r_0, r_1 \in_R G$
$g^{r_i} \mod n = \mathsf{Alg}_{pr}^{pr}(r_i, g, n), i = 0, 1$
$g^{s_i} {h_i}^{s_i} \mod n = \text{2-}\mathsf{Sim}\text{-}\mathsf{Alg}_{pr}^{pr}((s_i, g), (r_i, h_i), n),\ i = 0, 1$
$(A_0, B_0) = (g^{r_0} \mod n, g^{s_0} h_0^{r_0} \mod n)$
$(A_1, B_1) = (g^{r_1} \mod n, g^{s_1} h_1^{r_1} \mod n)$
$(A_0, B_0), (A_1, B_1)$
$(A_b)^r \mod n = \mathsf{Alg}_{pr}^{pr}(r, A_b, n)$
Private Output: $\perp$
Private Output: s_b

Fig. 1. Outsourcing Oblivious Transfer

$\mathsf{Alg}_{\mathsf{pr}}^{\mathsf{pr}}$ first runs Rand to compute the blinding pairs (x, g^x), (y, g^y) and (k, g^k). Denote $v = g^x$ and $\mu = g^y$. Now, we have

$$u_1^{a_1} \cdots u_t^{a_t} = (vw_1)^{a_1} \cdots (vw_t)^{a_t} = g^y g^z w_1^{a_1} \cdots w_t^{a_t}.$$

where $w_i = u_i v^{-1}$ and $z = x \sum_{i=1}^{t} a_i - y$ for $1 \leq i \leq t$. First, g^z is computed by invoking $\mathsf{Exp}(zk^{-1}, g^k)$ to $\mathcal{S}$.

Note that w_i's are completely random and therefore, can be revealed to $\mathcal{S}$. Hence, instead of invoking $\mathsf{Alg}_{\mathsf{pr}}^{\mathsf{pr}}$ t times it is now possible to invoke more efficient algorithm $\mathsf{Alg}_{\mathsf{pb}}^{\mathsf{pr}}$ t times. In particular, we gain a linear factor for the number of total multiplication in the number t. More precisely, a t-simultaneous modular exponentiation requires $t(4\ell+2k+5)+2(t+1)$ modular multiplications and $4t+1$ modular inversions instead of invoking $\mathsf{Alg}_{\mathsf{pr}}^{\mathsf{pr}}$ t-times which requires $t(4\ell+2k+10)+(t-1)$ modular multiplications and $6t$ modular inversions. Hence, we save $4t-3$ modular multiplications and $2t-1$ modular inversions by using our t-simultaneous modular exponentiation technique.

For instance, the complexity of 2-simultaneous modular exponentiations running 2-Sim-$\mathsf{Alg}_{\mathsf{pr}}^{\mathsf{pr}}$ is in total $8\ell+4k+16$ modular multiplications and 9 modular inversions instead of $8\ell+4k+21$ modular multiplications 12 modular inversions by running $\mathsf{Alg}_{\mathsf{pr}}^{\mathsf{pr}}$ two times only a single untrusted server.

By utilizing t calls of $\mathsf{Alg}_{\mathsf{pb}}^{\mathsf{pr}}$ and Theorem 3 the following holds.

Theorem 5. *There exists an algorithm $(\mathcal{C},\mathcal{S})$ which is an $(O(t\log^2(l)/l), 1-(\frac{k}{2(k+\ell)})^{2t})$-outsource-secure implementation of t-Sim-$\mathsf{Alg}_{\mathsf{pr}}^{\mathsf{pr}}$.*

We also note that even the less secure version $\ell=k=1$ of our proposed algorithm t-Sim-$\mathsf{Alg}_{\mathsf{pr}}^{\mathsf{pr}}$ has checkability $\frac{15}{16}$ using only one untrusted server instead of the probability $\frac{1}{2}$ using two servers of the algorithm in [11].

5 Complexity Analysis of the Proposed Algorithms

In this section, we first illustrate the complexity of our proposed algorithms using Table 1. In this table, we give the complexity results by counting the number of modular exponentiations for the server side, the number of modular multiplications (MMs), the number of modular inversions (MInvs), the number of Rands and checkability probabilities.

In Table 2, we give the complexity of the proposed algorithms by setting $\ell=k=20$, as we showed in the proof of Lemma 3. We note that letting $\ell=k=20$ reduces the probability of privacy leakage to negligible levels for a malicious server.

In order to compare our algorithm $\mathsf{Alg}_{\mathsf{pr}}^{\mathsf{pr}}$ with the previous results properly, we need to equate the checkability probabilities of all algorithms and count the number of all operations in terms of modular multiplications. For this purpose, we use the fact that in a real-life hardware setting a modular inversion is about 100 times slower than a modular multiplication [39]. In order to have the same checkability probability 15/16, we have to run the algorithm [23] 4 times, and the algorithm [11] $\log_3 2^4 \approx 2,52$ times. The comparison will now be as follows:

In [23], we have 9 MMs and 5 MInvs in one round. Hence, in 4 rounds we obtain 36 MMs and 20 MInvs. Since 20 inversions correspond to approximately 2000 MMs, we have approximately a total number of 2036 MMs for [23].

In [11], we have 7 MMs and 5 MInvs in one round. Hence, in ≈ 2.52 rounds we obtain ≈ 17 MMs and ≈ 7.5 MInvs. Since 7.5 inversions correspond to approximately 750 MMs, we have approximately a total number of 767 MMs for [11].

Our proposed algorithm Alg_{pr}^{pr} has 130 MMs and 6 MInvs. Since 6 inversions correspond to approximately 600 MMs, we have approximately a total number of 730 MMs.

In Table 3, we compare our algorithm Alg_{pb}^{pr} with the results of [23] and [11]. In the last column of Table 3, we give the total number of MMs which shows that our algorithm Alg_{pb}^{pr} is not only the most efficient algorithm but also is the first efficient algorithm using only one single untrusted server $\mathcal{S}$.

Remark 4. Although the number of MMs of Alg_{pr}^{pr} is slightly better than the number of MMs in the algorithm of [11] for only one outsourced modular exponentiation, using t-Sim-$\mathsf{Alg}_{\mathsf{pr}}^{\mathsf{pr}}$ we gain a linear factor in t which gives significantly better complexity results for the number of MMs.

Furthermore, Alg_{pr}^{pr} has better checkability probability (15/16 versus 2/3). We highlight that our checkability probability increases with t-simultaneous modular exponentiations whereas a possible generalization of the algorithm [11] decreases dramatically when t increases.

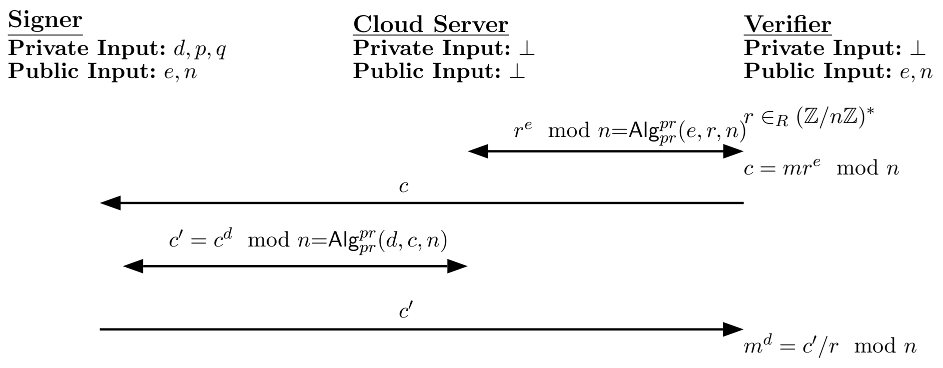

Fig. 2. Outsourcing Blind Signatures

6 Outsourced Oblivious Transfer and Blind Signatures

(a) Oblivious Transfer

Oblivious transfer is a powerful cryptographic primitive which is "complete" for secure multiparty computation [22] for any computable function [25]. In an OT protocol, the sender has two private input bits (s_0, s_1) and the receiver has one private input bit b. At the end of the protocol, the receiver learns only the bit s_b, whereas the sender does not know any information which bit was selected by the receiver.

With the help of cloud providers it is possible to independently compute any outsourced functionality even if the private input has not been revealed. Namely, clients only need to randomize/encrypt their data and de-randomize/decrypt the returned messages to get the desired results. It is one of the major computational overhead for Yao's garbled circuit protocol [28, 42], and used in several applications like biometric authentication, e-auctions, private information retrieval, private search [9,16,24,29]. Hence, running OT protocols for resource-constrained mobile environment may have substantial benefits.

In this section, we provide an example of outsourcing an OT protocol in a discrete log setting (see Figure 1). Assume that G is a group generated by g (i.e. $G =< g >$) and $h \in G$ where $\log_g h$ is unknown to any party. At the first step, the receiver chooses random $r \in_R G$ and invokes the cloud server $\mathcal{S}$ to compute $\mathsf{Alg}_{pr}^{pr}(r, g, n)$ and computes $h_b = g^r \mod n$. Note that at this stage, cloud server and the environment do not learn any valuable information about the inputs or the outputs. The receiver then computes $h_{1-b} = h/g^r$. Next, the receiver sends (h_0, h_1) to the sender. The sender now invokes $\mathcal{S}$ to run $\mathsf{Alg}_{pr}^{pr}(r_0, g, n)$ and $2-\mathsf{Sim}$-$\mathsf{Alg}_{pr}^{pr}((s_i, g), (r_i, h_i), n),\ i = 0, 1$

to compute and receive g^{r_i} and $h_i^{r_i} g^{s_i}$ for $i = 0, 1$, respectively. The sender then returns homomorphic ElGamal encryptions of s_0 and s_1 denoted as $(A_0, B_0) = (g^{r_0}, g^{s_0} h_0^{r_0})$ and $(A_1, B_1) = (g^{r_1}, g^{s_1} h_1^{r_1})$, respectively. Depending on his bit b, the receiver is able to decrypt one of these encryptions to learn either s_0 or s_1. Hence, if both parties follow the protocol specification, the receiver learns exactly one of the bits s_0 and s_1, and the sender does not know any information about what the receiver learns. The OT protocol used for outsourcing is secure in the semi-honest model but malicious versions of OT can be used analogously.

(b) Blind Signatures

Blind signatures have been suggested by Chaum [10]. Roughly speaking, it allows a signer interactively issue signatures and allows users to obtain them such that the signer does not see the resulting message and the signature pair during the signing session. Like any conventional electronic signatures they are unforgeable and can be verified using a public key.

Blind signatures can be applied to privacy preserving protocols like e-cash, e-voting and anonymous credentials. For a e-cash scenario, a bank blindly signs coins withdrawn by the users. For an e-voting scenario, an authority blindly signs a vote for later to cast the signed votes. As for anonymous credentials, the issuing authority blindly signs a key [8] for later to authenticate services anonymously. Hence, for mobile environment and constrained-devices, outsourcing blind signatures can be beneficial for real-life applications (see Figure 2).

7 Conclusion

In this paper, we propose new secure and efficient algorithms for outsourcing modular exponentiations (i.e., public-base & private-exponent, private-base & public-exponent, private-base & private-exponent and simultaneous modular exponentiations). Our algorithms are more efficient compared to the previous algorithms and they are modeled with only single untrusted cloud server solving the open problem formulated in [11]. Our algorithms also enjoy checkability property which is a significant improvement compared to prior works. The security of our algorithms are proven formally based on the model of [23]. We finally utilize our algorithms for outsourcing oblivious transfer protocols and blind signatures, which may be beneficial for resource-constrained mobile secure environments running on a client.

References

1. M. Abadi, J. Feigenbaum, and J. Kilian. On hiding information from an oracle. In *Proceedings of the Nineteenth Annual ACM Symposium on Theory of Computing*, STOC '87, pages 195–203. ACM, 1987.
2. Yonatan Aumann and Yehuda Lindell. Security against covert adversaries: Efficient protocols for realistic adversaries. *J. Cryptol.*, 23(2):281–343, 2010.
3. Razvan Barbulescu, Pierrick Gaudry, Antoine Joux, and Emmanuel Thomé. A quasi-polynomial algorithm for discrete logarithm in finite fields of small characteristic. volume abs/1306.4244, 2013.
4. D. Beaver, J. Feigenbaum, J. Kilian, and P. Rogaway. Locally random reductions: Improvements and applications. *Journal of Cryptology*, 10(1):17–36, 1997.

5. Donald Beaver and Joan Feigenbaum. Hiding instances in multioracle queries. In *STACS 90*, volume 415 of *Lecture Notes in Computer Science*, pages 37–48. Springer Berlin Heidelberg, 1990.
6. Dan Boneh, Eu-Jin Goh, and Kobbi Nissim. Evaluating 2-dnf formulas on ciphertexts. In *Theory of Cryptography*, volume 3378 of *Lecture Notes in Computer Science*, pages 325–341. Springer Berlin Heidelberg, 2005.
7. Victor Boyko, Marcus Peinado, and Ramarathnam Venkatesan. Speeding up discrete log and factoring based schemes via precomputations. In *Advances in Cryptology EUROCRYPT'98*, volume 1403 of *Lecture Notes in Computer Science*, pages 221–235. Springer Berlin Heidelberg, 1998.
8. Stefan A. Brands. *Rethinking Public Key Infrastructures and Digital Certificates: Building in Privacy.* MIT Press, Cambridge-London, 2000.
9. Julien Bringer, Herv Chabanne, and Alain Patey. Shade: Secure hamming distance computation from oblivious transfer. In *Financial Cryptography and Data Security*, volume 7862 of *Lecture Notes in Computer Science*, pages 164–176. Springer Berlin Heidelberg, 2013.
10. David Chaum. Blind signatures for untraceable payments. In *Advances in Cryptology, , Proceedings of CRYPTO '82*, pages 199–203. Springer US, 1983.
11. Xiaofeng Chen, Jin Li, Jianfeng Ma, Qiang Tang, and Wenjing Lou. New algorithms for secure outsourcing of modular exponentiations. In *Computer Security ESORICS 2012*, volume 7459 of *Lecture Notes in Computer Science*, pages 541–556. Springer Berlin Heidelberg, 2012.
12. H. Cohen, G. Frey, R. Avanzi, C. Doche, T. Lange, K. Nguyen, and F. Vercauteren. *Handbook of elliptic and hyperelliptic curve cryptography.* Chapman & Hall, Boca Raton, FL, 1st edition, 2006.
13. Ronald Cramer, Ivan Damgard, and Berry Schoenmakers. Proofs of partial knowledge and simplified design of witness hiding protocols. In *Advances in Cryptology CRYPTO 94*, volume 839 of *Lecture Notes in Computer Science*, pages 174–187. Springer Berlin Heidelberg, 1994.
14. Ronald Cramer, Rosario Gennaro, and Berry Schoenmakers. A secure and optimally efficient multi-authority election scheme. In *Proceedings of the 16th Annual International Conference on Theory and Application of Cryptographic Techniques*, EUROCRYPT'97, pages 103–118, Berlin, Heidelberg, 1997. Springer-Verlag.
15. Peter de Rooij. On schnorr's preprocessing for digital signature schemes. *J. Cryptology*, 10:1–16, 1997.
16. Giovanni Di Crescenzo, Tal Malkin, and Rafail Ostrovsky. Single database private information retrieval implies oblivious transfer. In *Advances in Cryptology EUROCRYPT 2000*, volume 1807 of *Lecture Notes in Computer Science*, pages 122–138. Springer Berlin Heidelberg, 2000.
17. Giovanni Di Crescenzo and Rafail Ostrovsky. On concurrent zero-knowledge with pre-processing. In *Advances in Cryptology CRYPTO 99*, volume 1666 of *Lecture Notes in Computer Science*, pages 485–502. Springer Berlin Heidelberg, 1999.
18. Taher El Gamal. A public key cryptosystem and a signature scheme based on discrete logarithms. In *Proceedings of CRYPTO 84 on Advances in Cryptology*, pages 10–18. Springer-Verlag New York, Inc., 1985.
19. Marc Fischlin and Roger Fischlin. Efficient non-malleable commitment schemes. volume 22, pages 530–571. Springer-Verlag New York, Inc., 2009.
20. Rosario Gennaro. Multi-trapdoor commitments and their applications to proofs of knowledge secure under concurrent man-in-the-middle attacks. In *Advances in Cryptology CRYPTO 2004*, volume 3152 of *Lecture Notes in Computer Science*, pages 220–236. Springer Berlin Heidelberg, 2004.

21. Craig Gentry. A fully homomorphic encryption scheme. In *Phd Thesis inproceedingsinproceedingsinproceedingsStanford University*, 2009.
22. Oded Goldreich. *Foundations of Cryptography: Volume 2, Basic Applications*. Cambridge University Press, New York, NY, USA, 2004.
23. Susan Hohenberger and Anna Lysyanskaya. How to securely outsource cryptographic computations. In *Theory of Cryptography*, volume 3378 of *Lecture Notes in Computer Science*, pages 264–282. Springer Berlin Heidelberg, 2005.
24. Ari Juels and Michael Szydlo. A two-server, sealed-bid auction protocol. In *In Sixth Annual Proceedings of Financial Cryptography*, pages 72–86. Springer-Verlag, 2002.
25. Joe Kilian. Founding crytpography on oblivious transfer. In *Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing*, STOC '88, pages 20–31, New York, NY, USA, 1988. ACM.
26. Praveen Gauravaram Lakshmi Kuppusamy, Jothi Rangasamy. On secure outsourcing of cryptographic computations to cloud. In *ACM Symposium on Information, Computer and Communications Security ASIACCS*. ACM, 2014.
27. Jingwei Li, Duncan Wong, Jin Li, Xinyi Huang, and Yang Xiang. Secure outsourced attribute-based signatures. volume 99, 2014.
28. Yehuda Lindell and Benny Pinkas. A proof of security of yao's protocol for two-party computation. volume 22, pages 161–188. Springer-Verlag New York, Inc., 2009.
29. Helger Lipmaa. Verifiable homomorphic oblivious transfer and private equality test. In *Advances in Cryptology - ASIACRYPT 2003*, volume 2894 of *Lecture Notes in Computer Science*, pages 416–433. Springer Berlin Heidelberg, 2003.
30. Jie Liu, Bo Yang, and Zhiguo Du. Outsourcing of verifiable composite modular exponentiations. In *INCoS*, pages 546–551, 2013.
31. Xu Ma, Jin Li, and Fangguo Zhang. Outsourcing computation of modular exponentiations in cloud computing. volume 16, pages 787–796. Springer US, 2013.
32. Tsutomu Matsumoto, Koki Kato, and Hideki Imai. Speeding up secret computations with insecure auxiliary devices. In *Advances in Cryptology CRYPTO 88*, volume 403 of *Lecture Notes in Computer Science*, pages 497–506. Springer New York, 1990.
33. Phong Q. Nguyen, Igor E. Shparlinski, and Jacques Stern. Distribution of modular sums and the security of the server aided exponentiation. In *Cryptography and Computational Number Theory*, volume 20 of *Progress in Computer Science and Applied Logic*, pages 331–342. Birkhauser Basel, 2001.
34. Haixin Nie, Xiaofeng Chen, Jin Li, Josolph Liu, and Wenjing Lou. Efficient and verifiable algorithm for secure outsourcing of large-scale linear programming. In *Advanced Information Networking and Applications (AINA), 2014 IEEE 28th International Conference on*, pages 591–596, May 2014.
35. Pascal Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *Advances in Cryptology EUROCRYPT 99*, volume 1592 of *Lecture Notes in Computer Science*, pages 223–238. Springer Berlin Heidelberg, 1999.
36. TorbenPryds Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In *Advances in Cryptology CRYPTO 91*, volume 576 of *Lecture Notes in Computer Science*, pages 129–140. Springer Berlin Heidelberg, 1992.
37. Claus-Peter Schnorr. Efficient signature generation by smart cards. volume 4, pages 161–174, 1991.
38. C.P. Schnorr. Efficient identification and signatures for smart cards. In Jean-Jacques Quisquater and Joos Vandewalle, editors, *Advances in Cryptology EUROCRYPT 89*, volume 434 of *Lecture Notes in Computer Science*, pages 688–689. Springer Berlin Heidelberg, 1990.

39. Martin Seysen. Using an rsa accelerator for modular inversion. In *Cryptographic Hardware and Embedded Systems - CHES 2005, 7th International Workshop, Edinburgh, UK, August 29 - September 1, 2005, Proceedings*, volume 3659 of *Lecture Notes in Computer Science*, pages 226–236. Springer, 2005.
40. Marten Van Dijk, Dwaine Clarke, Blaise Gassend, G.Edward Suh, and Srinivas Devadas. Speeding up exponentiation using an untrusted computational resource. volume 39, pages 253–273. Kluwer Academic Publishers, 2006.
41. Marten Van Dijk and Ari Juels. On the impossibility of cryptography alone for privacy-preserving cloud computing. In *Proceedings of the 5th USENIX Conference on Hot Topics in Security*, HotSec'10, pages 1–8. USENIX Association, 2010.
42. Andrew C. Yao. Protocols for secure computations. In *Proceedings of the 23rd Annual Symposium on Foundations of Computer Science*, SFCS '82, pages 160–164. IEEE Computer Society, 1982.