Efficient and Verifiable Algorithms for Secure Outsourcing of Cryptographic Computations

Mehmet Sabır Kiraz and Osmanbey Uzunkol
Mathematical and Computational Sciences Labs
TÜBİTAK BİLGEM UEKAE
National Research Institute of Electronics and Cryptology P. K. 74
41470 Gebze/Kocaeli, TURKEY
Phone: +90 262 648 1945
Fax: +90 262 648 1100
{mehmet.kiraz, osmanbey.uzunkol}@tubitak.gov.tr

Abstract. Reducing the computational cost of cryptographic computations for resource-constrained devices is an active research area. Outsourcing the computation to an external server securely provides a practical solution. In particular, modular exponentiations are the most expensive computation of many cryptographic protocols. Outsourcing the modular exponentiations to a single, external and potentially untrusted cloud server securely while ensuring the privacy is the most realistic scenario. In this paper, we propose new efficient outsourcing algorithms for modular exponentiations using only one untrusted cloud server. These algorithms cover public-base & private-exponent, private-base & public-exponent, private-base & private-exponent and more generally private-base & private-exponents simultaneous modular exponentiations. Our algorithms are the most efficient outsourced computation algorithms using a single untrusted server. Furthermore, our algorithms provide the best checkability property with predetermined parameters. Finally, we give two different applications for outsourcing within the realms 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, location independent resource pooling, pay per use, rapid elasticity, 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.

Today's resource-constrained devices can be incapable of computing expensive cryptographic operations. This is the main reason that outsourcing computation plays 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 of 2048 bits (in order to have a cryptographically secure algorithm) has a big computational obstacle for the computationally limited devices. To compute a single modular exponentiation for 2048-bits exponent a, more than 3000 modular multiplications must be performed in average (using square and multiply method). Therefore, it is more usable 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. Namely, reducing the cost of masking before outsourcing, and the cost of removing the mask after obtaining the result from the 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. Namely, a 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 correctness of the result.

A trivial solution of the problem is to assume the existence of a fully-trusted or a semi-trusted cloud server. However, 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 are ensured without revealing the inputs/outputs while ensuring that the 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 note that our algorithms borrow computing power from only a single untrusted cloud server. This is a more realistic scenario compared to the state-of-the-art algorithms in [1–3]. 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.

Instead of having an adversary model where distrustful servers are assumed not to collude, our algorithms use only one single untrusted server (which is a more realistic adversary setting). We emphasize that although the existing solutions which 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 [4], the authors propose the first generic algorithm for single untrusted server considering private-base private-exponent. However, their scheme is quite inefficient since it requires ≈ 2000 modular multiplications. Furthermore, it has fixed checkability property of 1/2. In contrast to these solutions, we would like to highlight that our algorithms are the most efficient and verifiable solutions with respect to the existing ones (e.g., we have $\approx 10, 17$ times less MMs than the only existing algorithm [4]). Furthermore, our algorithms have the best, adjustable checkability where any adversarial behavior can be detected by the client with the probability $1 - \frac{1}{c(c-1)}$, where c is an arbitrarily small integer used as a security parameter for checkability (e.g., for $c = 4$ and $c = 8$ we have 11/12 and 55/56, resp.).

Moreover, our algorithm for simultaneous modular exponentiations is more efficient compared to the existing algorithms [2] and [4] (except the generalized result in [5], for which an analogous attack explained above makes the checkability step impossible). The algorithm proposed in [2] 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 [6] for any computable function [7]. It is also one of the major computational overhead for Yao's garbled circuit protocols [8,9]. OTs are also used in many applications like biometric authentication, e-auctions, private information retrieval, private search [10–13]. Hence, by outsourcing OT securely can be enhance the overall complexity for mobile environment and resource-constrained devices. Furthermore, blind signatures [14] 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 [15]. 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 [16]. This is the reason that we solely focus on expensive modular computations for certain cryptographic primitives.

Many algorithms [1–4,17–27] 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 [28], 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 [1] 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.* [2] propose a more efficient solution than Hohenberger-Lysyanskaya's algorithm with 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 [29], zero-knowledge proofs [30] and additive variant of ElGamal encryption [31]. Chen *et al.* use the outsourcing algorithms to compute Cramer-Shoup encryptions and Schnorr signatures securely.

We notice that in [32], the authors proposed an algorithm utilizing a single untrusted server (for public-base & private-exponent and private-base & public-exponent cases). The algorithm is quite interesting since the privacy is achieved based on the difficulty of the subset sum problem. Namely, the client first randomizes the exponents and then puts a private pattern to the exponent values before they are sent to the server. Note that the pattern can only be retrieved by solving the underlying subset sum problem. After the server send the values back to the client, the client can efficiently verify the correctness using the pattern. However, there is a checkability issue in their algorithm where an untrusted server can easily manipulate the result. More concretely, the client invokes the server $\mathsf{Exp}(a, g)$ to outsource the computation of g^a. However, the malicious server can compute $\mathsf{Exp}(a, gh)$ instead of $\mathsf{Exp}(a, g)$ for some bogus value h. The checks will pass successfully and subsequently the result would become incorrect without being detected.

We also point out that there is also no checkability property of the only existing algorithm [5] for modular exponentiations modulo composite numbers. By using the local notation of [5], the attack can be explained briefly as follows: A malicious server $\mathcal{S}$ uses the proposed values $\ell = \ell_1 = \ell_2 = 5$ in [5] (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.

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 [8]. Conventional homomorphic encryption schemes are either additive or multiplicative (e.g., RSA is multiplicative, Paillier and modified version of ElGamal encryption are additive [31, 33], or [34] 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 [35]. 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 [1] 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 and 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 Appendix A, we utilize our algorithms for Oblivious Transfer and Blind Signatures protocols. Finally, Section 6 concludes the paper.

2 Security and Privacy Model

In this work, we follow the security model proposed by Hohenberger and Lysyanskaya [1] like the recent results in [2, 4]. Assume that a client $\mathcal{C}$ would like to securely outsource an expensive cryptographic computation Alg to a cloud server $\mathcal{S}$. Our aim is to split the computation into two main 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, which is based on principally the model of [1].

Definition 1. *[1]* ***(Algorithm with outsource-I/O)*** *An algorithm* Alg *obeys the outsource input/output specification if it takes five inputs, and produces three outputs. The first three inputs are generated by an honest party, and are classified by how much the adversary* $\mathcal{A} = (\mathcal{E}, \mathcal{S}')$ *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 means that if a malicious $\mathcal{S}'$ can obtain some information about the secret of $\mathcal{C}^{\mathcal{S}}$ by playing the role of $\mathcal{C}$ instead of $\mathcal{S}$, then $\mathcal{S}'$ can also obtain it without following this procedure. More concretely, when $\mathcal{C}^{\mathcal{S}}(x)$ is queried, a simulator $\mathsf{Sim}_{\mathcal{S}'}$ is constructed in such a way 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. *[1]* ***(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* $\mathsf{stop}^i = \mathsf{TRUE}$*.).*
  - *The* ideal *process:*
    $\mathsf{EVIEW}^i_{\mathsf{ideal}} = \{(\mathsf{istate}^i, x^i_{hs}, x^i_{hp}, x^i_{hu}) \leftarrow I(1^k, \mathsf{istate}^{i-1});$
    $(\mathsf{estate}, j^i, x^i_{ap}, x^i_{au}, \mathsf{stop}^i) \leftarrow E(1^k, \mathsf{EVIEW}^{i-1}_{\mathsf{ideal}}, x^i_{hp}, x^i_{hu});$
    $(\mathsf{astate}^i, y^i_s, y^i_p, y^i_u) \leftarrow \mathsf{Alg}(\mathsf{astate}^{i-1}, x^{j^i}_{hs}, x^{j^i}_{hp}, x^{j^i}_{hu}, x^i_{ap}, x^i_{au});$
    $(\mathsf{sstate}^i, \mathsf{ustate}^i, Y^i_p, Y^i_u, \mathsf{rep}^{\mathsf{i}}) \leftarrow$

$\mathsf{Sim}_{\mathcal{E}}^{\mathcal{S}'(\mathsf{ustate}^{i-1})}$ $(\mathsf{sstate}^{i-1}, x_{hp}^{j^i}, x_{hu}^{j^i}, x_{ap}^i, x_{au}^i, y_p^i$ $, y_u^i)$; $(z_p^i, z_u^i) = \mathsf{rep}^{\mathsf{i}}(Y_p^i, Y_u^i) + (1 - \mathsf{rep}^{\mathsf{i}})(y_p^i, y_u^i) : (\mathsf{estate}, z_p^i, z_u^i)\}$
$\mathsf{EVIEW}_{\mathsf{ideal}} = \mathsf{EVIEW}_{\mathsf{ideal}}^i$ *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_{hs}^i*, but given the non-secret outputs that* Alg *produces when run all the inputs for round* i*, decides to either output the values* (y_p^i, y_u^i) *generated by* Alg*, or replace them with some other values* (Y_p^i, Y_u^i)*. Note that this is captured by having the indicator variable* rep^i *be a bit that determines whether* y_p^i *will be replaced with* Y_p^i*. In doing so, it is allowed to query oracle* $\mathcal{S}'$*; 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:*
    $\mathsf{UVIEW}_{ideal}^i = \{\ (\mathsf{istate}^i, x_{hs}^i, x_{hp}^i, x_{hu}^i) \leftarrow I(1^k, \mathsf{istate}^{i-1})$;
    $(\mathsf{estate}^i, j^i, x_{ap}^i, x_{au}^i, \mathsf{stop}^i) \leftarrow \mathcal{E}(1^k, \mathsf{estate}^{i-1}, x_{hp}^i, x_{hu}^i, y_p^{i-1}, y_u^{i-1}$;
    $(\mathsf{astate}^i, y_s^i, y_p^i, y_u^i) \leftarrow \mathsf{Alg}(\mathsf{astate}^{i-1}, x_{hs}^{j^i}, x_{hp}^{j^i}, x_{hu}^{j^i}, x_{ap}^i, x_{au}^i)$;
    $(\mathsf{sstate}^i, \mathsf{ustate}^i) \leftarrow \mathsf{Sim}_{\mathcal{S}'}^{\mathcal{S}'(\mathsf{ustate}^{i-1})}$ $(\mathsf{sstate}^{i-1}, x_{hu}^{j^i}, x_{au}^i) : (\mathsf{ustate}^i)\ \}$

    $\mathsf{EVIEW}_{\mathsf{ideal}} = \mathsf{EVIEW}_{\mathsf{ideal}}^i$ *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_{hu}^i, x_{au}^i)*, queries* $\mathcal{S}'$*. As before,* $\mathcal{S}'$ *may maintain state.*

Definition 3. *[1]* ***(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. *[1]* ***(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. *[1] ((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

There are basically two different settings for which modular exponentiations are the most expensive parts of the cryptographic computation: Discrete logarithm problem (DLP) and RSA problem based. 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 DLP 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 [36]). 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 ourselves 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 [37]. 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 algorithms for scalar multiplications of elliptic curves, one can also obtain hybrid privacy preserving outsourcing algorithms for pairing-based cryptosystems by means of outsourcing private inputs of pairing functions, bilinearity property, and private exponentiations in finite fields for the realization of ID-based cryptography [36].

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 private but the public key can be disclosed to the server but for the signatures only the private key are kept private. However, similar to the DLP case, hiding the exponent or the base element in the modular exponentiation enables the cryptographic protocols to obtain a privacy preserving outsourced schemes. Constructing such a system makes impossible for the server to distinguish between encryption/decryption/signature/verification processes which can be an important design criteria for privacy preserving infrastructures (e.g., attribute-based encryption schemes). To the best of our knowledge, there is only one algorithm proposed for RSA based modular exponentiation [5], which is non-checkable as explained in Section (b). Hence, our algorithm is the first which unifies modular exponentiation modulo a prime or a composite number.

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.

(b) The Main Algorithm

In this section, we 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 $(\mathbb{Z}/n\mathbb{Z})^*$ with order m. Note that n can be either a prime number or an RSA modulus covering the both cases as 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, as usual, m must be a prime number for DLP based systems.

Let now the blinding factors $(x, g^x, g^{-x}), (t_1, g^{t_1}, g^{-t_1}), (t_2, g^{t_2}, g^{-t_2}) \in \mathbb{Z}/m\mathbb{Z} \times G^2$ and (y, g^y), $(s, g^s) \in \mathbb{Z}/m\mathbb{Z} \times G$ be given using a Rand Algorithm as defined in [1]. Note that these blinding factors are computed in order to speed up computations [1, 2]. The values x, y, t_1 and t_2 can be used several times for different exponents in order to hide the exponent whereas the value s should be used only once in order to hide the exponent.

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 by $\mathsf{Exp}(a, u)$. Before we explain our main algorithm we propose the following subalgorithm SubAlg for outsourcing g^z where g is a generator of the group where the base g and the exponent z are not necessarily private.

SubAlgorithm (SubAlg): Outsourcing an auxiliary modular exponentiation

Input: (z, g, c) (where $z \in \mathbb{Z}/m\mathbb{Z}$ with $< g >= G \leq \mathbb{F}_n^*$ for DLP or $g \in_R G = (\mathbb{Z}/n\mathbb{Z})^*$ for RSA with $|G| = m$, where $n, m \in \mathbb{N}$, and an arbitrary small $c \in \mathbb{N}$).
Output: The value g^z in G.
Precomputation: A Rand algorithm computes and stores the following values for $\mathcal{C}$:

- $(s, g^s) \in \mathbb{Z}/m\mathbb{Z} \times G$,
- (t_1, t_1^{-1}, g^{t_1}), $(t_2, t_2^{-1}, g^{t_2}) \in_R (\mathbb{Z}/m\mathbb{Z})^2 \times G$,
- I=$\{1, \ldots, c\} \subseteq \mathbb{Z}/m\mathbb{Z}$ with $I^{-1} = \{1^{-1}, \ldots, c^{-1}\} \subseteq \mathbb{Z}/m\mathbb{Z}$.

1. $\mathcal{C}$ picks random elements c_1^{-1}, $c_2^{-1} \in_R I^{-1}$ with $c_1, c_2 \in I,\ c_1 \neq c_2$, and computes $z_1 \leftarrow (z - s) \cdot c_1^{-1}$ and $z_2 \leftarrow (-z + 2s) \cdot c_2^{-1}$.
2. $\mathcal{C}$ runs
   (a) $Z_1 \leftarrow \mathsf{Exp}(z_1 \cdot {t_1}^{-1}, g^{t_1})$.
   (b) $Z_2 \leftarrow \mathsf{Exp}(z_2 \cdot {t_2}^{-1}, g^{t_2})$.
3. $\mathcal{C}$ verifies $Z_1^{c_1} \cdot Z_2^{c_2} \stackrel{?}{=} g^s$ and returns $Z_1^{2c_1} \cdot Z_2^{c_2}$.

Theorem 1. SubAlg *terminates and outputs correctly with probability* $\frac{1}{c(c-1)}$.

Proof. $\mathcal{C}$ first computes $z_1 = (z - s) \cdot c_1^{-1}$ and $z_2 = (-z + 2s) \cdot c_2^{-1}$, where c_1^{-1}, $c_2^{-1} \in_R I^{-1}$ with $c_1, c_2 \in I,\ c_1 \neq c_2$.

$\mathcal{S}$ returns $Z_1 = g^{z_1} = g^{(z-s)\cdot c_1^{-1}}$ and $Z_2 = g^{z_2} = g^{(-z+2s)\cdot c_2^{-1}}$. Finally, $\mathcal{C}$ computes and verifies the following result. If the equality does not hold then algorithm outputs checkability failure.

$$\begin{aligned} Z_1^{c_1} \cdot Z_2^{c_2} &= (g^{(z-s)\cdot c_1^{-1}})^{c_1} \cdot (g^{(-z+2s)\cdot c_2^{-1}})^{c_2} \\ &= g^{(z-s)+(-z+2s)} = g^s \end{aligned}$$

Finally, $\mathcal{C}$ outputs

$$\begin{aligned} Z_1^{2c_1} \cdot Z_2^{c_2} &= (g^{(z-s)\cdot c_1^{-1}})^{2c_1} \cdot (g^{(-z+2s)\cdot c_2^{-1}})^{c_2} \\ &= g^{2(z-s)+(-z+2s)} = g^z \end{aligned}$$

We are now ready to prove that a malicious $\mathcal{S}$ cannot maliciously behave without being detected with probability $\frac{1}{c(c-1)}$. This is actually trivial because $\mathcal{S}$ does not know z, s, c_1 and z_1 (and c_2 and z_2) and it will only be successful if it can guess c_1 and c_2 correctly to have equality $Z_1^{c_1} \cdot Z_2^{c_2} = g^s$. □

We now propose our main algorithm Algorithm 1 ($\mathsf{Alg}_{\mathsf{pr}}^{\mathsf{pr}}$) for private-base and private-exponent. For completeness, we introduce the following notation: Let a finite set $M = \{m_1, \cdots, m_n\}$ be given. We denote by $\mathbb{S}_n(M)$ the group of permutations on M. Note that we can identify any permutation on $\mathbb{S}_n(M)$ with a permutation on $\mathbb{S}_n(\{1, \cdots, n\})$. By abuse of notation, we will write $\sigma(m_i) = \sigma(i)$ for any $\sigma \in \mathbb{S}_n(\{1, \cdots, n\})$.

Before we go into details we give a brief summary of $\mathsf{Alg}_{\mathsf{pr}}^{\mathsf{pr}}$ as follows. The client $\mathcal{C}$ first masks the base u and the exponent a respectively, and sends them to the server in a special form (based on the precomputed values). The server applies the algorithm specifications and returns the masked results. The client then removes the masks and verifies the correctness. More precisely, the value u^a is converted into $(vw)^a = g^{xa}\ w^a = \mu g^z w^a = \mu Z w^a$ where x, y, v, w are random values such that $w = uv^{-1}$, $z = ax - y$ and the precomputed values are $v = g^x$ and $\mu = g^y$. Therefore, the algorithm has basically three computations in order to compute u^a, i.e., μ, Z and w^a.

- The first value $\mu = g^y$ is already precomputed and stored.
- The second value $Z = g^z$ is computed via the subalgorithm SubAlg for computing a modular exponentiation for a generator g and an exponent $z = ax-y$. We highlight that this subalgorithm only assures the correctness of the result rather than hiding the base g and the exponent z. Note that z is already masked with x and y therefore does not leak any information to $\mathcal{S}$.
- Finally, w^a is outsourced securely which is the longest and the most complicated part. This value is outsourced by first dividing the private exponent a and a random value r into k and ℓ subcomponents such that $a = \sum_{i=1}^{k} a_i$ and $r = \sum_{i=1}^{\ell} r_i$, respectively. More precisely,
  - $\mathcal{C}$ chooses random sets $R := U_1 \cup U_2 \cup U_3 \cup U_4 := \{r_1, \cdots, r_\ell\}$ and $A := U_5 \cup U_6 \cup U_7 \cup U_8 := \{a_1, \cdots, a_k\}$ as in Figure 1 such that $r =$

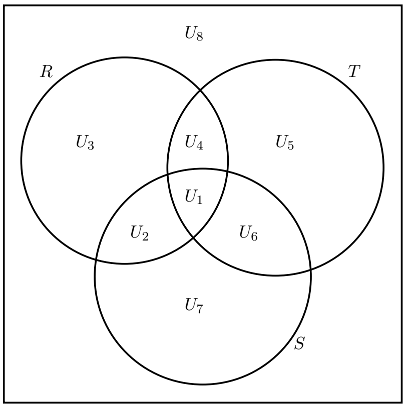

Fig. 1. The Partition of the Set $U = \cup_{i=1}^{8} U_i$

$\sum_{i=1}^{\ell} r_i$, $a = \sum_{i=1}^{k} a_i$. $\mathcal{C}$ first prepares the random subsets U_i (with arbitrary length) such that $U := A \cup R = \cup_{i=1}^{8} U_i$ where $U_i \neq \emptyset$, $\forall$ i and $U_i \cap U_j = \emptyset$, $\forall$ $i \neq j$. For the sign of the values, $\mathcal{C}$ chooses further a random element $\alpha = (\alpha_1, \cdots, \alpha_{\ell+k}) \in_R \{0,1\}^{\ell+k}$. Note that the elements of R is used to randomize the private exponent a.

- Next, $\mathcal{C}$ forms random subsets $S := U_1 \cup U_2 \cup U_6 \cup U_7$ and $T := U_1 \cup U_4 \cup U_5 \cup U_6$ of U such that $s = \sum_{s_i \in S} s_i$ and $t = \sum_{t_i \in T} (-1)^{\alpha_i} \cdot t_i$ satisfying the condition that $s + c_1 t = c_2$ where $c_1, c_2 \in_R \{1, \ldots, c\}$, $c \in \mathbb{N}$. The aim of this condition is to assure the checkability property of $\mathsf{Alg}_{\mathsf{pr}}^{\mathsf{pr}}$.
- Let $U := \{u_1, \ldots, u_{k+\ell}\}$. $\mathcal{C}$ chooses a random permutation $\sigma \in_R \mathbb{S}_{\ell+k}(U)$ and sets the permuted elements $U = \sigma(U) := (\sigma_1, \ldots, \sigma_{k+\ell})$. The permutation σ basically mixes the subcomponents of a and r to ensure the privacy of the exponent a. Moreover, the invocations take place with signed values of the subcomponents using $\alpha = (\alpha_1, \cdots, \alpha_{\ell+k}) \in_R \{0,1\}^{\ell+k}$ (i.e., $\mathcal{S}$ computes $w^{(-1)^{\alpha_i} \cdot \sigma(u_i)}$).
- After $\mathcal{S}$ returns the computed values $\mathcal{C}$ basically computes w^{a+r}, w^r, w^s and w^t and verifies the correctness of the result w^a by checking $s + c_1 t = c_2$ in the exponents.
- If the verification is successful, $\mathcal{C}$ outputs w^a by removing w^r from w^{a+r}.

– $\mathcal{C}$ finally returns the expected outcome u^a by computing $\mu Z w^a$.

The algorithm is now given as follows:

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

---

Input: (a, u, k, ℓ, c) (where $a \in \mathbb{Z}/m\mathbb{Z}$ with $u \in < g >= G \leq \mathbb{F}_n^*$ for DLP or $u \in_R G = (\mathbb{Z}/n\mathbb{Z})^*$ for RSA with $|G| = m$, where $n, m \in \mathbb{N}$, and an arbitrary small $c \in \mathbb{N}$).
Output: The value u^a in G.
Precomputation: A Rand algorithm computes $(y, g^y) \in_R \mathbb{Z}/m\mathbb{Z} \times G$ and $(x, g^x, g^{-x}) \in \mathbb{Z}/m\mathbb{Z} \times G^2$ for $\mathcal{C}$ with $v = g^x$ and $\mu = g^y$.

1. $\mathcal{C}$ computes $w \leftarrow uv^{-1}$, $z \leftarrow ax - y$, runs $Z = \mathsf{SubAlg}(z, g, c)$.
2. Using Figure 1, $\mathcal{C}$ chooses random sets $R := U_1 \cup U_2 \cup U_3 \cup U_4 := \{r_1, \cdots, r_\ell\}$ and $A := U_5 \cup U_6 \cup U_7 \cup U_8 := \{a_1, \cdots, a_k\}$ such that $r = \sum_{i=1}^{\ell} r_i$, $a = \sum_{i=1}^{k} a_i$.
3. $\mathcal{C}$ first forms random subsets U_i with arbitrary length such that $U := A \cup R = \cup_{i=1}^{8} U_i$ where $U_i \neq \emptyset\ \forall\ i$ and $U_i \cap U_j = \emptyset\ \forall\ i \neq j$.
4. For the sign of the values $\mathcal{C}$ chooses further a random $\alpha = (\alpha_1, \cdots, \alpha_{\ell+k}) \in_R \{0,1\}^{\ell+k}$.
5. Next, $\mathcal{C}$ forms random subsets $S := U_1 \cup U_2 \cup U_6 \cup U_7$ and $T := U_1 \cup U_4 \cup U_5 \cup U_6$ of U such that $s = \sum_{s_i \in S} s_i$ and $t = \sum_{t_i \in T} (-1)^{\alpha_i} \cdot t_i$ satisfying the condition that $s + c_1 t = c_2$ where $c_1, c_2 \in_R \{1, \ldots, c\}$.
6. Let $U := \{u_1, \ldots, u_{k+\ell}\}$. $\mathcal{C}$ chooses a random permutation $\sigma \in_R \mathbb{S}_{\ell+k}(U)$ and sets the permuted elements $U = \sigma(U) := (\sigma_1, \ldots, \sigma_{k+\ell})$.
7. $\mathcal{C}$ sets $\mathcal{U}_-, \mathcal{U}_+ \leftarrow 1$ and uses the partitions in Figure 1. Furthermore, $\mathcal{C}$ runs and computes in random order for $j \in \{1, \cdots, k+\ell\}$
   `(`$\mathcal{U}_-$`, `$\mathcal{U}_+$` are negative/positive parts of the exponents of `U`)`
   (a) If $\sigma_j \in U_1$: `(Computation of `w^{u_1}` where `$|U_1| = u_1$`)`
      i. If $\alpha_j = 1$: $w_j \leftarrow \mathsf{Exp}(-\sigma_j, w)$
         A. $\mathcal{U}_- \leftarrow \mathcal{U}_- \cdot w_j$
      ii. If $\alpha_j = 0$: $w_j \leftarrow \mathsf{Exp}(\sigma_j, w)$
         A. $\mathcal{U}_+ \leftarrow \mathcal{U}_+ \cdot w_j$
      iii. $\mathcal{C}$ sets $\mathcal{T}_- \leftarrow \mathcal{U}_-$ and $\mathcal{T}_+ \leftarrow \mathcal{U}_+$
   (b) If $\sigma_j \in U_2$: `(Computation of `w^{u_2}`)`
      i. If $\alpha_j = 1$: $w_j \leftarrow \mathsf{Exp}(-\sigma_j, w)$
         A. $\mathcal{U}_- \leftarrow \mathcal{U}_- \cdot w_j$
      ii. If $\alpha_j = 0$: $w_j \leftarrow \mathsf{Exp}(\sigma_j, w)$
         A. $\mathcal{U}_+ \leftarrow \mathcal{U}_+ \cdot w_j$
      iii. $\mathcal{C}$ sets $\mathcal{S}_- \leftarrow \mathcal{U}_-$ and $\mathcal{S}_+ \leftarrow \mathcal{U}_+$

(c) If $\sigma_j \in U_3$: `(Computation of` w^{u_3}`)`
  i. If $\alpha_j = 1$: $w_j \leftarrow \mathsf{Exp}(-\sigma_j, w)$
    A. $\mathcal{U}_- \leftarrow \mathcal{U}_- \cdot w_j$
  ii. If $\alpha_j = 0$: $w_j \leftarrow \mathsf{Exp}(\sigma_j, w)$
    A. $\mathcal{U}_+ \leftarrow \mathcal{U}_+ \cdot w_j$
  iii. $\mathcal{C}$ sets $\mathcal{R}_- \leftarrow \mathcal{U}_-$ and $\mathcal{R}_+ \leftarrow \mathcal{U}_+$

(d) If $\sigma_j \in U_4$, $\mathcal{C}$ sets $temp_-$, $temp_+ \leftarrow 1$: `(Computation of` w^{u_4}`)`
  i. If $\alpha_j = 1$: $w_j \leftarrow \mathsf{Exp}(-\sigma_j, w)$
    A. $temp_- \leftarrow temp_- \cdot w_j$ `(`$temp$ `is used to minimize MMs)`
  ii. If $\alpha_j = 0$: $w_j \leftarrow \mathsf{Exp}(\sigma_j, w)$
    A. $temp_+ \leftarrow temp_+ \cdot w_j$
  iii. $\mathcal{C}$ sets $\mathcal{U}_- \leftarrow \mathcal{U}_- \cdot temp_-$, $\mathcal{R}_- \leftarrow \mathcal{R}_- \cdot temp_-$, $\mathcal{U}_+ \leftarrow \mathcal{U}_+ \cdot temp_+$ and $\mathcal{R}_+ \leftarrow \mathcal{R}_+ \cdot temp_+$

(e) If $\sigma_j \in U_5$, $\mathcal{C}$ sets $temp_-$, $temp_+ \leftarrow 1$: `(Computation of` w^{u_5}`)`
  i. If $\alpha_j = 1$: $w_j \leftarrow \mathsf{Exp}(-\sigma_j, w)$
    A. $temp_- \leftarrow temp_- \cdot w_j$
  ii. If $\alpha_j = 0$: $w_j \leftarrow \mathsf{Exp}(\sigma_j, w)$
    A. $temp_+ \leftarrow temp_+ \cdot w_j$
  iii. $\mathcal{C}$ sets $\mathcal{U}_- \leftarrow \mathcal{U}_- \cdot temp_-$, $\mathcal{T}_- \leftarrow \mathcal{T}_- \cdot temp_-$, $\mathcal{U}_+ \leftarrow \mathcal{U}_+ \cdot temp_+$ and $\mathcal{T}_+ \leftarrow \mathcal{T}_+ \cdot temp_+$

(f) If $\sigma_j \in U_6$, $\mathcal{C}$ sets $temp_-$, $temp_+ \leftarrow 1$: `(Computation of` w^{u_6}`)`
  i. If $\alpha_j = 1$: $w_j \leftarrow \mathsf{Exp}(-\sigma_j, w)$
    A. $temp_- \leftarrow temp_- \cdot w_j$
  ii. If $\alpha_j = 0$: $w_j \leftarrow \mathsf{Exp}(\sigma_j, w)$
    A. $temp_+ \leftarrow temp_+ \cdot w_j$
  iii. $\mathcal{C}$ sets $\mathcal{U}_- \leftarrow \mathcal{U}_- \cdot temp_-$, $\mathcal{T}_- \leftarrow \mathcal{T}_- \cdot temp_-$, $\mathcal{S}_- \leftarrow \mathcal{S}_- \cdot temp_-$, $\mathcal{U}_+ \leftarrow \mathcal{U}_+ \cdot temp_+$, $\mathcal{T}_+ \leftarrow \mathcal{T}_+ \cdot temp_+$ and $\mathcal{S}_+ \leftarrow \mathcal{S}_+ \cdot temp_+$.

(g) If $\sigma_j \in U_7$, $\mathcal{C}$ sets $temp_-$, $temp_+ \leftarrow 1$: `(Computation of` w^{u_7}`)`
  i. If $\alpha_j = 1$: $w_j \leftarrow \mathsf{Exp}(-\sigma_j, w)$
    A. $temp_- \leftarrow temp_- \cdot w_j$
  ii. If $\alpha_j = 0$: $w_j \leftarrow \mathsf{Exp}(\sigma_j, w)$
    A. $temp_+ \leftarrow temp_+ \cdot w_j$
  iii. $\mathcal{C}$ sets $\mathcal{U}_- \leftarrow \mathcal{U}_- \cdot temp_-$, $\mathcal{S}_- \leftarrow \mathcal{S}_- \cdot temp_-$, $\mathcal{U}_+ \leftarrow \mathcal{U}_+ \cdot temp_+$ and $\mathcal{S}_+ \leftarrow \mathcal{S}_+ \cdot temp_+$

(h) If $\sigma_j \in U_8$: `(Computation of` w^{u_8}`)`
  i. If $\alpha_j = 1$: $w_j \leftarrow \mathsf{Exp}(-\sigma_j, w)$
    A. $\mathcal{U}_- \leftarrow \mathcal{U}_- \cdot w_j$
  ii. If $\alpha_j = 0$: $w_j \leftarrow \mathsf{Exp}(\sigma_j, w)$
    A. $\mathcal{U}_+ \leftarrow \mathcal{U}_+ \cdot w_j$

8. $\mathcal{C}$ verifies $\mathcal{S}_+ \cdot (\mathcal{T}_- \cdot \mathcal{T}_+)^{c_1} \stackrel{?}{=} w^{c_2} \cdot \mathcal{S}_-$
   `(Verification step by checking` $s + c_1 t = c_2$ `in the exponents)`

9. $\mathcal{C}$ returns $\mu \cdot Z \cdot (\mathcal{U}_- \cdot \mathcal{R}_+)^{-1} \cdot (\mathcal{R}_- \cdot \mathcal{U}_+)$
   `(This is the expected outcome` u^a`)`

Correctness and Termination.

Theorem 2. $\mathsf{Alg}_{\mathsf{pr}}^{\mathsf{pr}}$ *terminates and outputs correctly.*

Proof. Precomputation and Step 1 of $\mathsf{Alg}_{\mathsf{pr}}^{\mathsf{pr}}$ says that $u^a = (vw)^a = g^{xa}w^a = \mu g^z w^a = \mu Z w^a$ where $w = uv^{-1}$ and $z = ax - y$.

We set $\Theta = \{\alpha_j : \ \alpha_j = 1, j = 1, \dots, k + \ell\}$ and $u_i := |U_i|$ for all $i \in \{1, \dots, 8\}$. In step 7 part (a) with using the query results of $\mathcal{S}$, $\mathcal{C}$ computes the negative part $w_-^{u_1}$ and the positive part $w_+^{u_1}$ as follows:

$$w_-^{u_1} = \prod_{\substack{\alpha_i \in \Theta, \\ \sigma_i \in U_1}} w^{-\sigma_i} \text{ and } w_+^{u_1} = \prod_{\substack{\alpha_i \notin \Theta, \\ \sigma_i \in U_1}} w^{\sigma_i}.$$

The output will be assigned to the negative part $\mathcal{T}_-$ and the positive part $\mathcal{T}_+$ in the exponent the elements of T. Analogously, $\mathcal{C}$ computes in part (b) and (c) the corresponding negative parts and the positive parts, and assigns the output to the exponent elements of the contributed sets.

Different from the first 3 parts, using part (d), $\mathcal{C}$ computes the negative part $w_-^{u_4}$ and the positive part $w_+^{u_4}$:

$$w_-^{u_4} = \prod_{\substack{\alpha_i \in \Theta, \\ \sigma_i \in U_4}} w^{-\sigma_i} \text{ and } w_+^{u_4} = \prod_{\substack{\alpha_i \notin \Theta, \\ \sigma_i \in U_4}} w^{\sigma_i}.$$

The output will be multiplied in this case with the negative parts $\mathcal{U}_-$, $\mathcal{R}_-$ and the positive parts $\mathcal{U}_+$, $\mathcal{R}_+$ in the exponent elements of U and R. Analogously, $\mathcal{C}$ computes $w_-^{u_j}$ and $w_+^{u_j}$ for $j = 5, 6, 7, 8$ and multiply with the corresponding positive and negative parts in the exponent elements of the contributed sets. As a result, one obtains:

$$w^r = w^{u_1+u_2+u_3+u_4} = \left(\prod_{\substack{\alpha_i \in \Theta, \\ \sigma_i \in R}} w^{-\sigma_i} \right)^{-1} \cdot \prod_{\substack{\alpha_i \notin \Theta, \\ \sigma_i \in R}} w^{\sigma_i} = \mathcal{R}_-^{-1} \cdot \mathcal{R}_+,$$

$$w^s = w^{u_1+u_2+u_6+u_7} = \left(\prod_{\substack{\alpha_i \in \Theta, \\ \sigma_i \in S}} w^{-\sigma_i} \right)^{-1} \cdot \prod_{\substack{\alpha_i \notin \Theta, \\ \sigma_i \in S}} w^{\sigma_i} = \mathcal{S}_-^{-1} \cdot \mathcal{S}_+.$$

Using the definition of t, we obtain also

$$w^t = \prod_{\sigma_i \in T} w^{(-1)^{\alpha_i} \sigma_i} = \prod_{\substack{\alpha_i \in \Theta, \\ \sigma_i \in T}} w^{-\sigma_i} \cdot \prod_{\substack{\alpha_i \notin \Theta, \\ \sigma_i \in S}} w^{\sigma_i} = \mathcal{T}_- \cdot \mathcal{T}_+,$$

Together with steps (a) to (g) with step (h), we obtain

$$w^{a+r} == \left(\prod_{\substack{\alpha_i \in \Theta, \\ \sigma_i \in A \cup R}} w^{-\sigma_i} \right)^{-1} \cdot \prod_{\substack{\alpha_i \notin \Theta, \\ \sigma_i \in A \cup R}} w^{\sigma_i} = \mathcal{U}_-^{-1} \cdot \mathcal{U}_+,$$

$$w^{c_2} = w^{s+c_1 t} = w^s \cdot (w^t)^{c_1} = \mathcal{S}_-^{-1} \cdot \mathcal{S}_+ \cdot (\mathcal{T}_- \cdot \mathcal{T}_+)^{c_1}, \text{ hence,}$$

$$\mathcal{S}_+ \cdot (\mathcal{T}_- \cdot \mathcal{T}_+)^{c_1} = w^{c_2} \cdot \mathcal{S}_-.$$

If the equality does not hold then the checkability fails. If $\mathcal{S}$ runs the query algorithm properly then the algorithm ends with Step 7 as follows:

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

□

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 $\frac{\sqrt{\pi k}}{2^{3k}}$ where $k = \ell$.*

Proof. The output will only be disclosed if $\mathcal{S}'$ obtains exactly the same position of a_i's with their signs. Hence, the probability of this event is $1/\left(\binom{2k}{k} \cdot 2^k\right)$.

Hence, $\mathcal{S}'$ cannot distinguish the two test queries from all of the $2k$ 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 $\frac{\sqrt{\pi k}}{2^{3k}}$ by using the Stirling's approximation $\binom{2k}{k} \approx \frac{4^k}{\sqrt{\pi k}}$ [38]. Note that letting $k = \ell = 29$ the probability becomes negligible ($\approx 2^{-80}$). □

We are now ready to prove the security of $\mathsf{Alg}_{\mathsf{pr}}^{\mathsf{pr}}$. As explained above, outsource-security 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 3. *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 [1]. 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, like using Figure 1 it chooses random sets $R := U_1 \cup U_2 \cup U_3 \cup U_4 := \{r_1, \cdots, r_\ell\}$ and $A := U_5 \cup U_6 \cup U_7 \cup U_8 := \{a_1, \cdots, a_k\}$ such that $r = \sum_{i=1}^{\ell} r_i$, $a = \sum_{i=1}^{k} a_i$. $\mathsf{Sim}_{\mathcal{E}}$ first forms random subsets U_i

with arbitrary length such that $U := A \cup R = \cup_{i=1}^{8} U_i$ where $U_i \neq \emptyset$, $\forall\ i$ and $U_i \cap U_j = \emptyset$, $\forall\ i \neq j$. For the sign of the values $\mathsf{Sim}_{\mathcal{E}}$ chooses further a random $\alpha = (\alpha_1, \cdots, \alpha_{\ell+k}) \in_R \{0,1\}^{\ell+k}$. Next, $\mathsf{Sim}_{\mathcal{E}}$ forms random subsets $S := U_1 \cup U_2 \cup U_6 \cup U_7$ and $T := U_1 \cup U_4 \cup U_5 \cup U_6$ of U such that $s = \sum_{s_i \in S} s_i$ and $t = \sum_{t_i \in T} (-1)^{\alpha_i} \cdot t_i$ satisfying the condition that $s + c_1 t = c_2$ where $c_1, c_2 \in_R \{1, \ldots, c\}$. Let $U := \{u_1, \ldots, u_{k+\ell}\}$. $\mathcal{C}$ chooses a random permutation $\sigma \in_R \mathbb{S}_{\ell+k}(U)$ and sets the permuted elements $U = \sigma(U) := (\sigma_1, \ldots, \sigma_{k+\ell})$. $\mathsf{Sim}_{\mathcal{E}}$ sets $\mathcal{U}_-, \mathcal{U}_+ \leftarrow 1$ and uses the partitions in Figure 1.

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 $h \in_R G$ and outputs $Y_p^i = h$, $Y_u^i = \emptyset$, $\mathsf{rep^i} = 1$. 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 step 7 of $\mathsf{Alg}_{\mathsf{pr}}^{\mathsf{pr}}$ to assure that all parts of Exp $\mathcal{C}$ invokes is randomized independently using σ and α. 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 $\frac{\sqrt{\pi k}}{2^{3k}}$ due to Lemma 1, resulting in an output of "$error''$; otherwise, the software will indeed be successful in manipulating the output of $\mathsf{Alg}_{\mathsf{pr}}^{\mathsf{pr}}$ (e.g., because each request is independent of each other, sending approximately 29 wrong results with their signs to the client $\mathcal{C}$ makes the probability of not being detected to negligibly small ($\approx 1/2^{80}$).).

In the real execution, the $k + \ell$ real outputs of $\mathcal{S}'$ are firstly grouped into two different parts corresponding to their signs (positive or negative). The negative and positive parts will be independently computed due to the checkability condition $s + c_1 t = c_2$. The result will be multiplied corresponding to their signs (7 and 8 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 according to their signs. 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,

$\mathsf{EVIEW}^i_{\mathsf{real}} \sim \mathsf{EVIEW}^i_{\mathsf{ideal}}$. By the hybrid argument, we can easily 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 ith round when getting input, and instead chooses a permutation $\sigma \in \mathbb{S}_{\ell+k}$ and prepares a signed permuted random query of the form $((-1)^{\alpha_j}\sigma_j) \in \mathbb{Z}/m\mathbb{Z} \times G$ to $\mathcal{S}'$ using σ_j, α_j where $j \in \{1, \ldots, k+\ell\}$. $\mathsf{Sim}_{\mathcal{E}}$ randomly checks $(k+\ell)$ outputs from each procedure using σ. Then, $\mathsf{Sim}_{\mathcal{S}'}$ stores its own and states of $\mathcal{S}'$. 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 σ, α (see steps 6 and 7 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. $\square$

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 [1]. The algorithm SubAlg makes 3 calls to Rand and $4\log c + 8$ modular multiplications. The proposed algorithm $\mathsf{Alg}^{\mathsf{pr}}_{\mathsf{pr}}$ makes 2 further calls to Rand and together with SubAlg, $k + \ell + 4\log c + 30$ modular multiplications (MMs) and only 1 modular inversion (MInv) in order to compute $u^a \mod n$ (other operations like modular additions, doubling or multiplication with very small numbers like c are omitted). Also, a server aided exponentiation takes $O(\log^2(l))$ MMs using the number theoretic complexity analysis of Nguyen, Shparlinski, and Stern [25], or $O(1)$ MMs if a table-lookup method is used. On the other hand, it takes in average $1.5l$ 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 $\mathsf{Alg}^{\mathsf{pr}}_{\mathsf{pr}}$. $\square$

Lemma 3. *The algorithm $(\mathcal{C}, \mathcal{S})$ is an $(1 - \frac{1}{c(c-1)})$-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 either

1. the correct values c_1 and c_2 in SubAlg, or
2. the correct value a or r, or
3. the correct value s or t, or
4. the position of a value s_i where S=$\{s_1, \ldots, s_{k'}\}$ with $s = \sum_{j=1}^{k'} s_i$, or
5. the position of a value t_i where T=$\{t_1, \ldots, t_{k''}\}$ with $t = \sum_{j=1}^{k''} t_i$

For the first case, $\mathcal{S}$ finds the correct values c_1 and c_2 in SubAlg with probability $\frac{1}{c(c-1)}$ (see Theorem 1 for details).

For the second case, finding either the exact value of a or r has negligibly probability (see Lemma 1).

For the third case, to be able to find the correct values of s, the server $\mathcal{S}$ first needs to find out the subset S from the power set $\mathcal{P}(U)$ such that s= $\sum_{i=1}^{k'} s_i$. The value t can subsequently be obtained by solving the subset sum problem for $s + c_1 t = c_2$, where c_1 and c_2 are small integers. Similarly, one can start with t to find s. The complexity of finding such (s, t) pairs from the power set $\mathcal{P}(U)$ is $2^{k+\ell} \cdot 2^{(k+\ell)/2} = 2^{3/2(k+\ell)}$ (note that $|\mathcal{P}(U)| = 2^{k+\ell}$). The reason is that the best generic algorithms to solve the subset sum problem are lattice-based methods which require $2^{n/2}$ for any set of cardinality n [22, 39–42].

For the last two cases, $\mathcal{S}$ can attack the checkability of the system if she can find a value s_i (or t_i) with its sign. Namely, the checkability follows from $\sum_{i=1}^{k'} s_i + c_1 \sum_{i=1}^{k''} t_i = c_2$ and with the knowledge of s_i (or t_i) and its sign and the knowledge of c_1 and c_2. Finding a value s_i has probability at least $1/2$ and with probability at least $1/2$ to decide whether it has negative or positive sign. Therefore, the overall probability of this event is $\frac{1}{4c^2}$.

Hence, the overall probability for a malicious server $\mathcal{S}$ to declare a wrong value without being detected is

$$1 - \frac{1}{c(c-1)} = \min\{1 - \frac{1}{4c^2}, 1 - \frac{1}{c(c-1)}\}.$$

□

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* $(O(\log^2(l)/l)$, $(1-\frac{1}{c(c-1)})$*-outsource-secure implementation of* $\mathsf{Alg}_{\mathsf{pr}}^{\mathsf{pr}}$.

Remark 1. Letting $c = 4$ gives us the probability $11/12$ by Lemma 3 which is the best checkability result compared to previous works [1–4].

Note that in outsourced computation model the malicious server $\mathcal{S}$ can be seen as a covert adversary [43], 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 [43], 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 interests and their reputation 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. The new algorithm $\mathsf{Alg}_{\mathsf{pb}}^{\mathsf{pr}}$ for public-base & private-exponent is a special case of $\mathsf{Alg}_{\mathsf{pr}}^{\mathsf{pr}}$ by setting the values $x = y = 0$ in the precomputation step.

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

Proof. Correctness, termination and security of the algorithm follow easily as a corollary of the results for $\mathsf{Alg}_{\mathsf{pr}}^{\mathsf{pr}}$ by excluding the subalgorithm SubAlg. Because SubAlg is not used for $\mathsf{Alg}_{\mathsf{pb}}^{\mathsf{pr}}$, the checkability property becomes $1 - \frac{1}{4c^2}$. □

(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: (u, a, c) (where $a \in \mathbb{Z}/m\mathbb{Z}$, $u \in G$ with $< g >= G \leq \mathbb{F}_n^*$ for DLP or $g \in_R G = (\mathbb{Z}/n\mathbb{Z})^*$ for RSA with $|G| = m$, where $n, m \in \mathbb{N}$, and an arbitrary $i \in \mathbb{N}$).
Output: The value u^a in G.
Precomputation: A Rand algorithm computes and stores the following values for $\mathcal{C}$:

- $(s_i, g^{s_i}, g^{-s_i}), (t_i, g^{t_i}, g^{-t_i}) \in_R \mathbb{Z}/m\mathbb{Z} \times G^2$ for $i = 1, 2$.
- I=$\{1, \ldots, c\} \subseteq \mathbb{Z}/m\mathbb{Z}$ with $I^{-1} = \{1^{-1}, \ldots, c^{-1}\} \subseteq \mathbb{Z}/m\mathbb{Z}$

1. $\mathcal{C}$ picks random elements c_1, $c_2 \in_R I$ where $\gcd(c_1, c_2) = 1$, and computes $u_1 \leftarrow u^{c_1} \cdot g^{b_1 s_1}$ and $u_2 \leftarrow u^{c_2} \cdot g^{b_2 s_2}$ where $b_1, b_2 \in_R \{1, -1\}$.
2. $\mathcal{C}$ runs
   (a) $U_1 \leftarrow \mathsf{Exp}(a, u_1)$.
   (b) $U_2 \leftarrow \mathsf{Exp}(a, u_2)$.
   (c) $T_1 \leftarrow \mathsf{Exp}((b_1' a \cdot s_1 + b_3 t_1) \cdot c_3^{-1}, g)$ where $b_1', b_3 \in_R I$ and $c_3 \in I^{-1}$.
   (d) $T_2 \leftarrow \mathsf{Exp}((b_2' a \cdot s_2 + b_4 t_2) \cdot c_4^{-1}, g)$ where $b_2', b_4 \in_R I$ and $c_4^{-1} \in I^{-1}$.
3. For verification, $\mathcal{C}$ does the following computations:
   (a) $T_1' = T_1^{c_3} \cdot g^{-b_3 t_1}$ and $T_2' = T_2^{c_4} \cdot g^{-b_4 t_2}$ `(masking removal)`
   (b) if $b_1 \neq b_1'$ computes $U_1' \leftarrow U_1 \cdot T_1'$ else computes $U_1' \leftarrow U_1 \cdot (T_1')^{-1}$
   (c) if $b_2 \neq b_2'$ computes $U_2' \leftarrow U_2 \cdot T_2'$ else computes $U_2' \leftarrow U_2 \cdot (T_2')^{-1}$
   (d) verifies $(U_1')^{c_2} \stackrel{?}{=} (U_2')^{c_1}$ where $k = c_1 \cdot c_2$ and returns u^a
   `(easily computable since` $\gcd(c_1, c_2) = 1$`)`

Theorem 5. $\mathsf{Alg}_{\mathsf{pr}}^{\mathsf{pb}}$ *terminates and outputs correctly. Furthermore, there exists an algorithm which is an* $(O(\log^2(l)/l), 1 - \frac{1}{c(c-1)})$*-outsource-secure implementation of* $\mathsf{Alg}_{\mathsf{pr}}^{\mathsf{pb}}$.

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

$$u_1 = u^{c_1} \cdot g^{b_1 s_1}$$
$$u_2 = u^{c_2} \cdot g^{b_2 s_2}$$

where $b_1, b_2 \in_R \{1, -1\}$, c_1, $c_2 \in_R I$ and $\gcd(c_1, c_2) = 1$.

At Step 2 $\mathcal{S}$ returns the query results

$$U_1 = u_1^a = u^{ac_1} \cdot g^{b_1 a s_1}$$
$$U_2 = u_2^a = u^{ac_2} \cdot g^{b_2 a s_2}$$

and

$$T_1 = g^{(b_1' a s_1 + b_3 t_1) \cdot c_3^{-1}} \text{ and } T_2 = g^{(b_2' a s_2 + b_4 t_2) \cdot c_4^{-1}}$$

and then $\mathcal{C}$ computes the following to verify the result.

$\mathcal{C}$ first removes the masking values t_1 and t_2 using c_3 and c_4 as

$$T_1' = T_1^{c_3} \cdot g^{-b_3 t_1} = g^{b_1' a s_1}$$
$$T_2' = T_2^{c_4} \cdot g^{-b_4 t_2} = g^{b_2' a s_2}$$

Next, the masking values from U_1 and U_2 will be removed, i.e. $U_1' = u^{ac_1}$ and $U_2' = u^{ac_2}$. In order to avoid inversion we basically compare b_1 with b_1' and b_2 with b_2' ($U_1' = U_1 \cdot T_1'$ or $U_2' = U_2 \cdot (T_2')^{-1}$, respectively).

$\mathcal{C}$ verifies $(U_1')^{c_2} \stackrel{?}{=} (U_2')^{c_1}$ where $c_1, c_2 \in I$ and $k = c_1 \cdot c_2$. If the equality does not hold then algorithm outputs checkability failure. Finally, because $\gcd(c_1, c_2) = 1$ and c_1, c_2 are very small $\mathcal{C}$ efficiently computes u^a.

A malicious server cannot learn the private base u because it is randomized with g^{s_1} and g^{s_2}. Furthemore, a malicious server cannot also change the outcome unless she finds either c_1, c_2 or c_3, c_4 and the probability of this event is $1 - \frac{1}{c(c-1)}$. □

(c) *t*-Simultaneous Modular Exponentiations

We now generalize the notion of simultaneous modular exponentiation method of [2] 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 [15,30,44–47]. As described in [2], 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 [2] and utilizing only one untrusted server (instead of two servers one of which is assumed to be honest). We denote by $t\text{-}\mathsf{Sim}\text{-}\mathsf{Alg}_{\mathsf{pr}}^{\mathsf{pr}}$ for t-simultaneous modular exponentiation algorithm.

The scheme of Chen *et al.* [2] has probability of 2/3 for checkability in modular exponentiation utilizing and has probability 1/2 for $2\text{-}\mathsf{Sim}\text{-}\mathsf{Alg}_{\mathsf{pr}}^{\mathsf{pr}}$ 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 a scalable probability $1 - \frac{1}{c(c-1)}$ for checkability and utilizes only one single untrusted server.

We further emphasize that the natural generalization for 2-simultaneous modular exponentiation method in [2] 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 [15], 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 [2], our scheme has an adjustable probability of $1 - \frac{1}{c(c-1)}$ which is independent of t. More concretely, the algorithm works as follows:

$\mathsf{Alg}_{\mathsf{pr}}^{\mathsf{pr}}$ 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

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

where $w_i = u_i v^{-1}$ and $Z = g^z$ with $z = x \sum_{i=1}^{t} a_i - y$ for $1 \leq i \leq t$. First, Z is computed by invoking $Z = \mathsf{SubAlg}(z, g, c)$ 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 $t(\ell + k + 4\log c + 28) + 10 + 4\log c$ modular multiplications and t modular inversions instead of invoking $\mathsf{Alg}_{\mathsf{pr}}^{\mathsf{pr}}$ t-times which requires $t(\ell + k + 8\log c + 38)$ modular multiplications and t modular inversions. Hence, we save $10t + 4\log ct$ modular multiplications by using our t-simultaneous modular exponentiation technique.

For instance, the complexity of 2-simultaneous modular exponentiations running $2\text{-}\mathsf{Sim}\text{-}\mathsf{Alg}_{\mathsf{pr}}^{\mathsf{pr}}$ for $c = 4$ requires 184 MMs and 2 MInvs (instead of 200 MMs and 2 MInvs by running $\mathsf{Alg}_{\mathsf{pr}}^{\mathsf{pr}}$ twice).

By utilizing t calls of $\mathsf{Alg}_{\mathsf{pb}}^{\mathsf{pr}}$ and Theorem 4 the following holds.

Theorem 6. *There exists an algorithm* $(\mathcal{C}, \mathcal{S})$ *which is an* $(O(t\log^2(l)/l), 1-(\frac{1}{c(c-1)}))$*-outsource-secure implementation of* t-Sim-$\mathsf{Alg}_{\mathsf{pr}}^{\mathsf{pr}}$.

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

	Exp ($\mathcal{S}$)	MMs	MInvs	Rand	Checkability
SubAlg	2	$8 + 4\log c$	0	3	$1 - \frac{1}{c(c-1)}$
Alg_{pr}^{pr}	$\ell + k + 2$	$\ell + k + 8\log c + 38$	1	5	$1 - \frac{1}{c(c-1)}$
Alg_{pb}^{pr}	$\ell + k$	$\ell + k + 4\log c + 28$	1	0	$1 - \frac{1}{4c^2}$
Alg_{pr}^{pb}	4	$16\log c + 16$	2	8	$1 - \frac{1}{c(c-1)}$
t-Sim-$\mathsf{Alg}_{\mathsf{pr}}^{\mathsf{pr}}$	$t(\ell + k) + 1$	$t(\ell + k + 4\log c + 28)$ $+ 10 + 4\log c$	t	5	$1 - \frac{1}{c(c-1)}$

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; and for the client side the number of modular multiplications (MMs), the number of modular inversions (MInvs), the number of Rands and checkability probabilities. Note that we count the number of multiplication in the worst case by using classical double and and algorithm, i.e. for an $l-$bit exponent we require $2l + 1$.

In Table 2, we give the complexity of the proposed algorithms by setting $\ell = k = 29$ and $c = 4$. We note that by Lemma 3 letting $\ell = k = 29$ reduces the probability of privacy leakage to negligible levels for a malicious server.

In order to compare $\mathsf{Alg}_{\mathsf{pr}}^{\mathsf{pr}}$ 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 [48]. In order to have the same checkability probability 11/12, we have to run the

Table 2. Computation Complexity for Proposed Algorithms for $k = \ell = 29$, $c = 4$

	Exp ($\mathcal{S}$)	MMs	MInvs	Rand	Checkability
SubAlg	2	10	0	3	11/12
$\mathsf{Alg}_{\mathsf{pr}}^{\mathsf{pr}}$	60	100	1	5	11/12
$\mathsf{Alg}_{\mathsf{pb}}^{\mathsf{pr}}$	58	86	1	0	63/64
$\mathsf{Alg}_{\mathsf{pr}}^{\mathsf{pb}}$	4	48	2	8	11/12
2-Sim-$\mathsf{Alg}_{\mathsf{pr}}^{\mathsf{pr}}$	117	184	2	5	11/12

algorithm [1] $\log_2 12 \approx 3,58$ times, and the algorithm [2] $\log_3 12 \approx 2,26$ times. The comparison will now be as follows:

In [1], we have 9 MMs and 5 MInvs in one round. Hence, in $\log_2 12$ rounds we obtain $9 \cdot \log_2 12$ MMs and $5 \cdot \log_2 12$ MInvs. Hence, we have a total number of $9 \cdot \log_2 12 + 100 \cdot 5 \cdot \log_2 12 \approx 1825$ MMs for [1].

In [2], we have 7 MMs and 3 MInvs in one round. Hence, in $\log_3 12$ rounds we obtain $7 \cdot \log_3 12$ MMs and$3 \cdot \log_3 12$ MInvs. Hence, we have a total number of $7 \cdot \log_3 12 + 100 \cdot 3 \cdot \log_3 12 \approx 694$ MMs for [2].

In [4], the goal is to outsource u^a where $c = a - b\xi$ with b and c are known by the server with probability 1/6. Therefore, ξ must be large enough to prevent the brute-force attack. Hence, to have a negligible level, one has to choose $\xi \approx 2^{77}$. There are 167 MMs and 4 MInvs for the checkability of 1/2. Hence, in $\log_2 12$ rounds we obtain $167 \cdot \log_2 12$ MMs and $4 \cdot \log_2 12$ MInvs. Hence, we have a total number of $167 \cdot \log_2 12 + 100 \cdot 4 \cdot \log_2 12$ MMs for [4].

The algorithm Alg_{pr}^{pr} has 100 MMs and only 1 MInv. Hence, we have a total number of approximately $100 + 1 \cdot 100 = 200$ MMs.

In Table 3, we compare our algorithm Alg_{pb}^{pr} with the results of [1], [2] and [4]. In the last column of Table 3, we give the total number of MMs which shows that our algorithm Alg_{pb}^{pr} is the most efficient algorithm using only one single untrusted server $\mathcal{S}$ with the best checkability.

Remark 2. Although the number of MMs of Alg_{pr}^{pr} is slightly better than the number of MMs in the algorithm of [2] 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 (11/12 versus 2/3). We highlight that our checkability probability increases with the value of c at the expense of increasing the number of modular multiplication logarithmically. In particular, our approach enables the designer to

Table 3. Comparison with the Previous Results

	MMs	MInvs	Single Server	Checkability	Total MMs after equating the checkability to 11/12
[1] (TC '05)	9	5	✗	1/2	$\log_2 12 \cdot 509 \approx 1825$
[2] (ESORICS '12)	7	3	✗	2/3	$\log_3 12 \cdot 307 \approx 694$
[4] (ESORICS '14)	167	4	✓	1/2	$\log_2 12 \cdot 567 \approx 2033$
Ours	100	1	✓	11/12	$100 + 100 \approx 200$

obtain privacy preserving outsourcing algorithms with scalable checkability.

6 Conclusion

In this paper, we propose new, scalable, 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 significantly more efficient compared to the previous algorithms. Moreover, the proposed algorithms are modeled where only one single untrusted cloud server exists. Our algorithms also enjoy the predetermined checkability property which is a significant improvement compared to prior works. The security of our algorithms are proven formally based on the model of [1]. 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.

The algorithm for single server in [4] requires extremely large number of MMs whereas our algorithm needs comparably very small number of MMs ($\approx 10, 17$ times less MMs). On the other hand, although the communication round of our algorithm is constant, the overhead of information exchange is still large. Therefore, it is an interesting open problem to find better constructions achieving smaller (possibly constant) communication overhead together with smaller number of modular multiplications without any modular inversions.

Acknowledgements

This work is partly supported by a joint research project funded by Bundesministerium für Bildung und Forschung (BMBF), Germany (01DL12038) and TÜBİTAK, Turkey (TBAG-112T011). It has also been partially supported by the COST Action CRYPTACUS (IC1403).

References

1. 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.
2. 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.
3. 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.
4. Yujue Wang, Qianhong Wu, DuncanS. Wong, Bo Qin, ShermanS.M. Chow, Zhen Liu, and Xiao Tan. Securely outsourcing exponentiations with single untrusted program for cloud storage. In *Computer Security - ESORICS 2014*, volume 8712 of *Lecture Notes in Computer Science*, pages 326–343. Springer International Publishing, 2014.
5. Jie Liu, Bo Yang, and Zhiguo Du. Outsourcing of verifiable composite modular exponentiations. In *INCoS*, pages 546–551, 2013.
6. Oded Goldreich. *Foundations of Cryptography: Volume 2, Basic Applications*. Cambridge University Press, New York, NY, USA, 2004.
7. 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.
8. 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.
9. 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.
10. 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.
11. 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.
12. 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.
13. 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.

14. David Chaum. Blind signatures for untraceable payments. In *Advances in Cryptology, , Proceedings of CRYPTO '82*, pages 199–203. Springer US, 1983.
15. Stefan A. Brands. *Rethinking Public Key Infrastructures and Digital Certificates: Building in Privacy.* MIT Press, Cambridge-London, 2000.
16. 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.
17. Jingwei Li, Duncan Wong, Jin Li, Xinyi Huang, and Yang Xiang. Secure outsourced attribute-based signatures. volume 99, 2014.
18. 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.
19. 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.
20. D. Beaver, J. Feigenbaum, J. Kilian, and P. Rogaway. Locally random reductions: Improvements and applications. *Journal of Cryptology*, 10(1):17–36, 1997.
21. 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.
22. 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.
23. Peter de Rooij. On schnorr's preprocessing for digital signature schemes. *J. Cryptology*, 10:1–16, 1997.
24. 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.
25. 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.
26. 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.
27. Claus-Peter Schnorr. Efficient signature generation by smart cards. volume 4, pages 161–174, 1991.
28. 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.
29. Marc Fischlin and Roger Fischlin. Efficient non-malleable commitment schemes. volume 22, pages 530–571. Springer-Verlag New York, Inc., 2009.
30. 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.

31. 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.
32. Xu Ma, Jin Li, and Fangguo Zhang. Outsourcing computation of modular exponentiations in cloud computing. volume 16, pages 787–796. Springer US, 2013.
33. 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.
34. 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.
35. Craig Gentry. A fully homomorphic encryption scheme. In *Phd Thesis inproceedingsinproceedingsStanford University*, 2009.
36. 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.
37. 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.
38. Keith Conrad. Stirlings formula, http://www.math.uconn.edu/ kconrad/blurbs/analysis/stirling.pdf.
39. Ellis Horowitz and Sartaj Sahni. Computing partitions with applications to the knapsack problem. volume 21, pages 277–292. ACM, 1974.
40. A.K. Lenstra, Jr. Lenstra, H.W., and L. Lovsz. Factoring polynomials with rational coefficients. volume 261, pages 515–534. Springer-Verlag, 1982.
41. MatthijsJ. Coster, Antoine Joux, BrianA. LaMacchia, AndrewM. Odlyzko, Claus-Peter Schnorr, and Jacques Stern. Improved low-density subset sum algorithms. volume 2, pages 111–128. Birkhuser-Verlag, 1992.
42. C.P. Schnorr and M. Euchner. Lattice basis reduction: Improved practical algorithms and solving subset sum problems. volume 66, pages 181–199. Springer-Verlag, 1994.
43. Yonatan Aumann and Yehuda Lindell. Security against covert adversaries: Efficient protocols for realistic adversaries. volume 23, pages 281–343. Springer-Verlag New York, Inc., 2010.
44. 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.
45. 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.
46. 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.
47. 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.
48. Martin Seysen. Using an rsa accelerator for modular inversion. In *Cryptographic Hardware and Embedded Systems - CHES 2005, 7th International Workshop, Ed-*

inburgh, UK, August 29 - September 1, 2005, Proceedings, volume 3659 of *Lecture Notes in Computer Science*, pages 226–236. Springer, 2005.

A Applications: Outsourced Oblivious Transfer and Blind Signatures

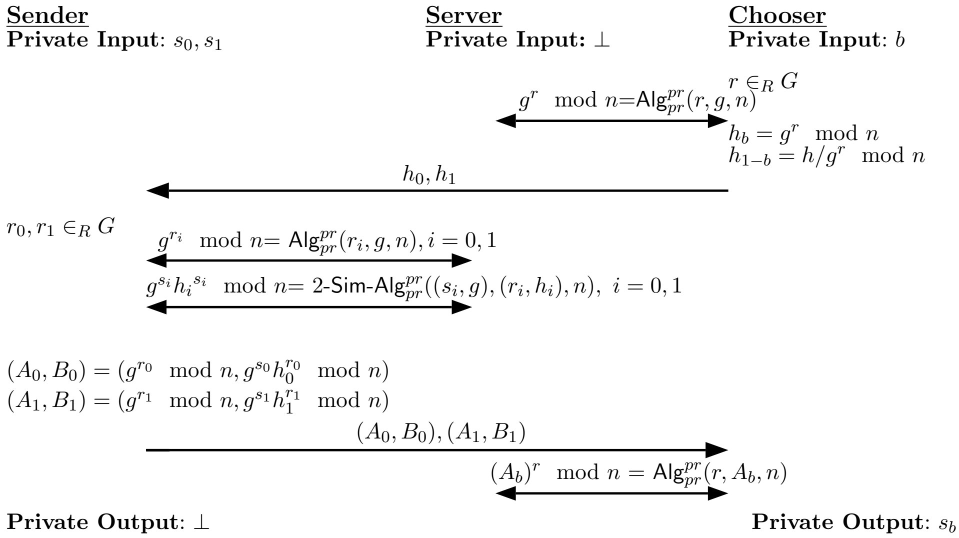

Fig. 2. Outsourcing Oblivious Transfer

(a) Oblivious Transfer

Oblivious transfer is a powerful cryptographic primitive which is "complete" for secure multiparty computation [6] for any computable function [7]. 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 [8,9], and used in several applications like biometric authentication, e-auctions, private information retrieval, private search [10–13]. 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 2). 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.

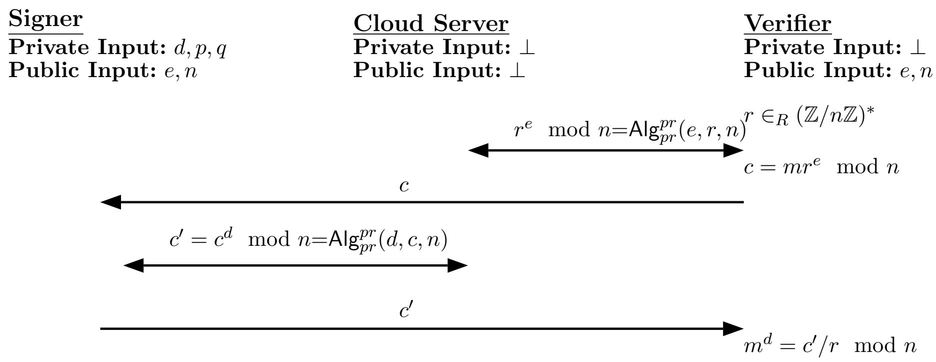

Fig. 3. Outsourcing Blind Signatures

(b) Blind Signatures

Blind signatures have been suggested by Chaum [14]. 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 [15] 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 3).