Practical and Secure Outsourcing of Discrete Log Group Exponentiation to a Single Malicious Server

Giovanni Di Crescenzo
Vencore Labs
Basking Ridge, NJ, 07920, USA
gdicrescenzo@vencorelabs.com

Matluba Khodjaeva
CUNY Graduate Center
New York, NY, 10016, USA
mkhodjaeva@gradcenter.cuny.edu

Delaram Kahrobaei
PhD Program in Computer Science
CUNY Graduate Center
New York, NY, 10016, USA
dkahrobaei@gc.cuny.edu

Vladimir Shpilrain
CUNY Graduate Center
New York, NY, 10016, USA
shpil@groups.sci.ccny.cuny.edu

ABSTRACT

Group exponentiation is an important operation used in many public-key cryptosystems and, more generally, cryptographic protocols. To expand the applicability of these solutions to computationally weaker devices, it has been advocated that this operation is outsourced from a computationally weaker client to a computationally stronger server, possibly implemented in a cloud-based architecture. While preliminary solutions to this problem considered mostly honest servers, or multiple separated servers, some of which honest, solving this problem in the case of a single (logical), possibly malicious, server, has remained open since a formal cryptographic model was introduced in [20]. Several later attempts either failed to achieve privacy or only bounded by a constant the (security) probability that a cheating server convinces a client of an incorrect result.

In this paper we solve this problem for a large class of cyclic groups, thus making our solutions applicable to many cryptosystems in the literature that are based on the hardness of the discrete logarithm problem or on related assumptions. Our main protocol satisfies natural correctness, security, privacy and efficiency requirements, where the security probability is exponentially small. In our main protocol, with very limited offline computation and server computation, the client can delegate an exponentiation to an exponent of the same length as a group element by performing an exponentiation to an exponent of short length (i.e., the length of a statistical parameter). We also show an extension protocol that further reduces client computation by a constant factor, while increasing offline computation and server computation by about the same factor.

CCSW'17, November 3, 2017, Dallas, TX, USA
© 2017 Association for Computing Machinery.
ACM ISBN 978-1-4503-5204-8/17/11…$15.00
https://doi.org/10.1145/3140649.3140657

CCS CONCEPTS

• **Security and privacy** → *Public key encryption*; *Mathematical foundations of cryptography*; *Privacy-preserving protocols*;

KEYWORDS

Secure Outsourcing, Secure Delegation, Modular Exponentiations, Discrete Logarithms, Cryptography, Group Theory

1 INTRODUCTION

Server-aided cryptography is an active research direction addressing the problem of clients delegating or outsourcing cryptographic computations to servers. Ideas related to this area have circulated in the literature already many years ago (see, e.g., [9], which introduced 'wallets with observers' where a third party, such as a bank, installs hardware on a user's computer to facilitate its future computations). Recently, this area is seeing an increased interest because of application scenario like cloud computing (where a client interacts with a much more powerful server), computations with low-power devices, such as wireless, RFIDs, etc (where a resource-constrained client interacts with a more powerful server).

The first formal model for outsourcing of cryptographic operations was introduced in [20], where the authors especially studied outsourcing of modular exponentiation, as this operation is a cornerstone of so many cryptographic protocols. In this model, we have a client, with an input x, who outsources to one or more servers the computation of a function F on the client's input, and the main challenges are:

(1) *privacy*: only minimal or no information about x should be revealed to the servers;
(2) *security*: the servers should not be able, except possibly with very small probability, to convince the client to accept a result different than $F(x)$; and
(3) *efficiency*: the client's computation time should be much smaller than computing $F(x)$ without outsourcing the computation.

In [20], the authors studied secure outsourcing of exponentiation to 2 servers of which at most one was malicious, and to 1 server, who was honest on almost all inputs. Since then, the problem of outsourcing exponentiation to a single, arbitrarily malicious server, has remained unsolved. This open problem was first posed in the original paper [20] and then reiterated in [25]. This is even the case in a model, as used in all previous work in the area, where relatively expensive offline computation can be performed and stored on the client's device. In many proposed protocols, the offline computation even involves the computation of modular exponentiations with random exponents, which are envisioned to be computed by an entity different than the client, or pre-computed by the client itself when additional time or computational power is available. The model also allows a client to perform in the online phase (a not large number of) less expensive operations like multiplications, an assumption that might be reasonable even in computationally challenged devices, in light of recent advances (see, for instance, [1], showing how to practically implement group multiplication, for a specific group, and a related public-key cryptosystem, using RFID tags).

Previous results. As also mentioned in [20], a number of solutions had been proposed, even before their paper introduced a security model, and then broken in follow-up papers. The single-server solution from [20] assumes that the server is honest on almost all inputs.

Other solutions were proposed in more recent papers, but these solution either only consider a semi-honest server [11], or two non-colluding servers [10], or do not target input privacy [14], or only achieve constant security probability (of detecting a cheating server) [8, 23, 25]. The schemes proposed in [30, 31] do not satisfy our privacy requirement. Finally, the scheme proposed in [32] does not satisfy our security requirement.

All mentioned solutions for the outsourcing of exponentiation are based on pre-computed exponentiations of random exponents, which are somehow stored on the client machine in an offline phase. These exponentiations might be precomputed by another party and stored on the client's device, or might be computed by the client itself using a pseudo-random power generator. One such generator was proposed by combining the results in [6, 24], based on a hidden-subset-sum hardness assumption, although this assumption needs to be re-evaluated in light of the most recent attacks.

The literature of course contains several elegant general-purpose solutions, applicable to any polynomial-time computable function, and starting with [17], which uses garbled circuits [26] and fully-homomorphic encryption [18] to efficiently, privately and securely outsource any polynomial-time circuit to a single (semi-honest) server. Due to their generality, these solutions are only asymptotically efficient, but not so in practical settings, and are thus out of scope in this paper.

Our Contributions. In this paper we first of all provide rigorous definitions, mainly based on [17] as well as [20], for the requirements of correctness, privacy, security and efficiency for outsourced computation protocols in the single, malicious, server model, with an allowed off-line phase.

In this model, we construct protocols that provably satisfy these requirements while delegating, to a *single* malicious server, group exponentiation, for a large class of cyclic groups, including groups often used for cryptosystems with provable security under the Discrete Logarithm or related assumptions. All protocols satisfy security with probability exponentially small in a statistical security parameter λ (which can be set equal to, for instance, 128). Our protocols' privacy and security properties do not rely on any additional complexity assumptions, as the adversary corrupting the server is *not limited to run in polynomial time.* Our protocols delegate function $F_{exp,g,q}(x) = g^x$ (i.e., variable-exponent, fixed-base exponentiation over an arbitrary cyclic group G).

Our main protocol (in Section 3) delegates exponentiation in cyclic groups to a single, possibly malicious, server. The client delegates an exponentiation with an exponent as long as a group element (e.g., of $\sigma = 2048$ bits), while only performing a single exponentiation with a much smaller exponent (e.g., of $\lambda = 128$ bits). Both the offline phase and the server in the online phase only require 2 exponentiations with σ-bit exponents. It is based on a 1-round probabilistic verification of the server's computation, and satisfies information-theoretic privacy, and security with exponentially small probability $2^{-\lambda}$ of the malicious server fooling the client. The main technical idea is simultaneously enforcing a probabilistic verification equation (verifiable using less than 2λ group multiplications), a deterministic inequality check and a group membership protocol (verifiable using no more than 1 group multiplication), given 2 correct exponentiations available to the client in an offline phase.

Our second protocol, presented in Section 4, extends the protocol in Section 3 to achieve the following (tunable) efficiency tradeoff: it further reduces the number of the client's group multiplications, while increasing the number of group exponentiations of random exponents performed during the offline phase and the number of the server's group exponentiations. Here, the main technical idea consists of running multiple probabilistic verification equations with smaller exponents at a performance cost for the client smaller than independently running each of the equations.

As in all previous work in the area, we consider a model with an offline phase, where a client can precompute exponentiations to random exponents, or another party can precompute them and store them on the client's device. Alternatively, if their hidden-subset-sum assumption holds when run with a low number of multiplications, the pseudo-random power generator from [6, 24] could be used.

2 NOTATIONS AND DEFINITIONS

In this section we formally define outsourcing protocols, and their correctness, security, privacy and efficiency requirements, mainly building on the definitional approach from [17], as well as the one from [20]. We also introduce group notations and definitions that will be used in the rest of the paper. We start with some basic notations.

Basic notations. The expression $y \leftarrow T$ denotes the probabilistic process of randomly and independently choosing y from set T. The expression $y \leftarrow A(x_1, x_2, \ldots)$ denotes the (possibly probabilistic) process of running algorithm A on input $x_1, x_2, \ldots$ and any necessary random coins, and obtaining y as output. The expression $(z_A, z_B) \leftarrow (A(x_1, x_2, \ldots), B(y_1, y_2, \ldots))$ denotes the (possibly probabilistic) process of running an interactive protocol between A, taking as input $x_1, x_2, \ldots$ and any necessary random coins, and B, taking as input $y_1, y_2, \ldots$ and any necessary random coins, where z_A, z_B are A and B's final outputs, respectively, at the end of this protocol's execution.

System scenario, entities, and protocol. We consider a system with two types of parties: clients and servers, where a client's computational resources are expected to be more limited than a server's ones, and therefore clients are interested in delegating (or outsourcing) the computation of specific functions to servers. In all our solutions, we consider a single *client*, denoted as C, and a single *server*, denoted as S. We assume that the communication link between each client and S is private or not subject to confidentiality, integrity, or replay attacks, and note that such attacks can be separately addressed using known techniques in cryptography and security. As in all previous work in the area, we consider a model with an offline phase, where exponentiations to random exponents can be precomputed and made somehow available to the client. This model has been justified in several ways, all appealing to different application settings. In the presence of a trusted party (say, setting up the client's device), the trusted party can simply perform the precomputed exponentiations and store them on the client's device. If no trusted party is available, in the presence of a pre-processing phase where the client's device does not have significant computation constraints, the client can itself perform the precomputed exponentiations and store them on its own device. Clients that always have significant computation constraints could spend more preprocessing time or use the pseudo-random power generator from [6, 24], which, based on a hidden-subset-sum assumption, only performs a relatively smaller number of group multiplications, to generate a new exponentiation for a pseudo-random exponent. Here, we note that this latter assumption needs to be re-evaluated in light of recent attacks. For simplicity of description, we will consider an Offline algorithm executed by either a trusted party or a client with no significant computation constraints, and remark any changes in our results when this algorithm is executed using a pseudo-random power generator.

Let σ denote the computational security parameter (i.e., the parameter derived from hardness considerations on the underlying computational problem), and let λ denote the statistical security parameter (i.e., a parameter such that evens with probability $2^{-\lambda}$ are extremely rare). Both parameters are expressed in unary notation (i.e., $1^\sigma, 1^\lambda$). When performing numerical performance analysis, we use $\sigma = 2048$ and $\lambda = 128$, as these are currently the most often recommended parameter settings in cryptographic protocols and applications.

Let $F : Dom(F) \rightarrow CoDom(F)$ be a function, where $Dom(F)$ denotes F's domain, $CoDom(F)$ denotes F's co-domain, and $desc(F)$ denotes F's description. Assuming $desc(F)$ is known to both C and S, and input x is known only to C, we define a *client-server protocol for the outsourced computation of* F in the presence of an offline phase as a 2-party, 2-phase, communication protocol between C and S, denoted as $(C(1^\sigma, 1^\lambda, desc(F), x), S(1^\sigma, 1^\lambda, desc(F)))$, and consisting of the following steps:

(1) $pp \leftarrow \text{Offline}(1^\sigma, 1^\lambda, desc(F))$,
(2) $(y_C, y_S) \leftarrow (C(1^\sigma, 1^\lambda, desc(F), pp, x), S(1^\sigma, 1^\lambda, desc(F))$.

As discussed above, Step 1 is executed in an *offline phase*, when the input x to the function F is not yet available. Step 2 is executed in the *online phase*, when the input x to the function F is available to C. At the end of both phases, C learns y_C (intended to be $= y$) and S learns y_S (usually an empty string in this paper). We will often omit $desc(F), 1^\sigma, 1^\lambda$ for brevity of description. Executions of outsourced computation protocols can happen sequentially (each execution starting after the previous one is finished), or concurrently (S runs at the same time one execution with each one of many clients).

Correctness Requirement. Informally, the (natural) correctness requirement states that if both parties follow the protocol, C obtains some output at the end of the protocol, and this output is, with high probability, equal to the value obtained by evaluating function F on C's input. A formal definition follows.

Definition 2.1. Let σ, λ be the security parameters, let F be a function, and let (C, S) be a client-server protocol for the outsourced computation of F. We say that (C, S) satisfies δ_c*-correctness* if for any x in F's domain, it holds that

$$\text{Prob}\left[out \leftarrow \text{CorrExp}_\text{F}(1^\sigma, 1^\lambda) \;:\; out = 1 \right] \geq \delta_c,$$

for some δ_c close to 1, where experiment CorrExp is detailed below:

```
$\text{CorrExp}_\text{F}(1^\sigma, 1^\lambda)$
1. $pp \leftarrow \text{Offline}(desc(F))$
2. $(y_C, y_S) \leftarrow (C(pp, x), S)$
3. if $y_C = F(x)$ then **return:** 1
    else **return:** 0
```

Security Requirement. Informally, the most basic security requirement would state the following: if C follows the protocol, a malicious adversary corrupting S cannot convince C to obtain, at the end of the protocol, some output y' different from the value y obtained by evaluating function F on C's input x. To define a stronger and more realistic security requirement, we augment the adversary's power so that the adversary can even choose C's input x, before attempting to convince C of an incorrect output. We also do not restrict the adversary to run in polynomial time. A formal definition follows.

Definition 2.2. Let σ, λ be the security parameters, let F be a function, and let (C, S) be a client-server protocol for the outsourced computation of F. We say that (C, S) satisfies ϵ_s*-security against a malicious adversary* if for any algorithm A, it holds that

$$\text{Prob}\left[out \leftarrow \text{SecExp}_\text{F,A}(1^\sigma, 1^\lambda) \;:\; out = 1 \right] \leq \epsilon_s,$$

for some ϵ_s close to 0, where experiment SecExp is detailed below:

```
$\text{SecExp}_\text{F,A}(1^\sigma, 1^\lambda)$
1. $pp \leftarrow \text{Offline}(desc(F))$
2. $(x, aux) \leftarrow A(desc(F))$
3. $(y', aux) \leftarrow (C(pp, x), A(aux))$
4. if $y' = \perp$ or $y' = F(x)$ then **return:** 0
    else **return:** 1.
```

Privacy Requirement. Informally, the privacy requirement should guarantee the following: if C follows the protocol, a malicious adversary corrupting S cannot obtain any information about C's input x from a protocol execution. This is formalized by extending the indistinguishability-based approach typically used in formal definitions for encryption schemes. That is, the adversary can pick two inputs x_0, x_1, then one of these two inputs is chosen at random and used by C in the protocol with the adversary acting as S, and then the adversary tries to guess which input was used by C. As for security, we do not restrict the adversary to run in polynomial time. A formal definition follows.

Definition 2.3. Let σ, λ be the security parameters, let F be a function, and let (C, S) be a client-server protocol for the outsourced computation of F. We say that (C, S) satisfies *ϵ_p-privacy (in the sense of indistinguishability) against a malicious adversary* if for any algorithm A, it holds that

$$\text{Prob}\left[\, out \leftarrow \text{PrivExp}_{\text{F,A}}(1^\sigma, 1^\lambda) \;:\; out = 1 \,\right] \leq \epsilon_p,$$

for some ϵ_p close to 0, where experiment PrivExp is detailed below:

```
PrivExp_{F,A}(1^σ, 1^λ)
1. $pp \leftarrow \text{Offline}(desc(F))$
2. $(x_0, x_1, aux) \leftarrow A(desc(F))$
3. $b \leftarrow \{0, 1\}$
4. $(y', d) \leftarrow (C(pp, x_b), A(aux))$
5. if $b = d$ then **return:** 1
     else **return:** 0.
```

Efficiency Metrics and Requirements. Let (C, S) be a client-server protocol for the outsourced computation of function F. We say that (C, S) has *efficiency parameters* $(t_F, t_P, t_C, t_S, cc, mc)$, if F can be computed (without outsourcing) using $t_F(\sigma, \lambda)$ atomic operations, C can be run in the offline phase using $t_P(\sigma, \lambda)$ atomic operations and in the online phase using $t_C(\sigma, \lambda)$ atomic operations, S can be run using $t_S(\sigma, \lambda)$ atomic operations, C and S exchange a total of at most mc messages, of total length at most cc. In our analysis, we only consider the most expensive group operations as atomic operations (e.g., group multiplications and/or exponentiation), and neglect lower-order operations (e.g., equality testing, additions and subtractions between group elements). While we naturally try to minimize all these protocol efficiency metrics, our main goal is to design protocols where

(1) $t_C(\sigma, \lambda) << t_F(\sigma, \lambda)$, and
(2) $t_S(\sigma, \lambda)$ is not significantly larger than $t_F(\sigma, \lambda)$,

based on the underlying assumption, consistent with the state of the art in cryptographic implementations at least for many group types, that group multiplication requires significantly less computing resources than group exponentiation.

Group notations and definitions. Let $(G, \times)$ be a group, let σ be its computational security parameter, and let L denote the length of the binary representation of elements in G. Typically, in cryptographic applications we set L as about equal to σ.

We also assume that $(G, \times)$ is *cyclic*, has order q, and denote as g one of its generators. By $y = g^x$ we denote the *exponentiation (in G)* of g to the x-th power; i.e., the value $y \in G$ such that $g \times \cdots \times g = y$, where the multiplication operation $\times$ is applied $x - 1$ times. Let $\mathbb{Z}_q = \{0, 1, \ldots, q - 1\}$, and let $F_{exp,g,q} : \mathbb{Z}_q \to G$ denote the function that maps every $x \in \mathbb{Z}_q$ to the exponentiation (in G) of g to the x-th power. By $desc(F_{exp,g,q})$ we denote a conventional description of the function $F_{exp,g,q}$ that includes its semantic meaning as well as generator g, order q and the efficient algorithms computing multiplication and inverses in G. By $t_{exp}(\ell)$ we denote a parameter denoting the number of multiplications in G used to compute an exponentiation (in G) of a group value to an arbitrary ℓ-bit exponent. By $t_{m,exp}(\ell)$ we denote a parameter denoting the number of multiplications in G used to compute m exponentiations (in G) of the same group value to m arbitrary ℓ-bit exponents.

We define an *efficiently verifiable membership protocol* for G as a one-message protocol, denoted as the pair (mProve,mVerify) of algorithms, satisfying

- *completeness*: for any $w \in G$, mVerify(w,mProve(w))=1;
- *soundness*: for any $w \notin G$, and any mProve′,
  mVerify(w,mProve′(w))=0;
- *efficient verifiability*: the number of multiplications $t_{mVerify}(\sigma)$ in G executed by mVerify is $o(t_{exp})$;
- *efficient provability*: the number of multiplications $t_{mProve}(\sigma)$ in G executed by mProve is not significantly larger than t_{exp}.

We say that a group is *efficient* if its description is short (i.e., has length polynomial in σ), its associated operation $\times$ and the inverse operation are efficient (i.e., they can be executed in time polynomial in σ), and it has an efficiently verifiable membership protocol. Note that for essentially all cyclic groups frequently used in cryptography, the description is short and its associated $\times$ and inverse operations can be run in time polynomial in σ. The only non-trivial property to establish is whether the group has an efficiently verifiable membership protocol. In the rest of the paper we present our results for any arbitrary efficient cyclic group, of which we now show two examples that are often used in cryptography and that do have efficiently verifiable membership protocols.

Example 1: $(G, \times) = (\mathbb{Z}_p^*, \cdot \mod p)$, for a large prime p. This group was one of the most recommended for early foundational cryptographic schemes like the Blum-Micali pseudo-random generator [5], etc. Note that multiplication and inverses modulo p can be computed in time polynomial in $\log p$, and an efficiently verifiable membership protocol goes as follows:

(1) on input w, mProve does nothing;
(2) on input w, mVerify returns 1 if $0 < w < p$ and 0 otherwise.

The completeness, soundness, efficient provability properties of this protocol are easily seen to hold. The efficient verifiability property follows by noting that mVerify runs in time linear in $\log p$, which is strictly smaller than the time for exponentiation $\mod p$ (in fact, even the time for multiplication $\mod p$).

Example 2: $(G, \times) = (G_q, \cdot \mod p)$, for large primes p, q such that $p = 2q + 1$, where G_q is the q-order subgroup of $\mathbb{Z}_p^*$. This group is one of the most recommended for cryptographic schemes like the Diffie-Hellman protocol [15], El-Gamal encryption [16], Cramer-Shoup encryption [12], etc. It is known that G_q is cyclic and is characterized as the set of quadratic residues modulo p; i.e., the set of $z \in \mathbb{Z}_p^*$ for which there exists an $r \in \mathbb{Z}_p^*$ such that $z = r^2 \mod p$. Using results from [13] based on this characterization, an efficiently verifiable membership protocol can be built as follows:

(1) on input w, mProve computes $r = w^{(q+1)/2} \mod p$ and returns r;
(2) on input w, r, mVerify returns 1 if $w = r^2 \mod p$ and 0 otherwise.

The completeness and soundness properties of this protocol are easily seen to hold. The efficient provability follows by noting that mProve only performs 1 exponentiation $\mod p$. The efficient verifiability property follows by noting that mVerify only requires one multiplication $\mod p$.

3 OUTSOURCING EXPONENTIATION IN ANY EFFICIENT CYCLIC GROUP

In this section we present our first and main protocol for outsourcing exponentiation in cyclic groups. We first formally state our result, then informally and formally describe the protocol, and finally prove its correctness, security, privacy and efficiency properties.

Formal theorem statement. We obtain the following

Theorem 3.1. Let $(G, \times)$ be an efficient cyclic group, let σ be its computational security parameter, and let λ be a statistical security parameter. There exists (constructively) a client-server protocol (C, S) for outsourcing the computation of function $F_{exp,g,q}$, which satisfies

1. δ_c-correctness, for $\delta_c = 1$;
2. ϵ_s-security, for $\epsilon_s = 2^{-\lambda}$;
3. ϵ_p-privacy, for $\epsilon_p = 0$;
4. efficiency with parameters $(t_F, t_S, t_P, t_C, cc, mc)$, where
   - t_F is $= t_{exp}(\sigma)$;
   - t_S is $= 2\, t_{exp}(\sigma) + 2\, t_{mProve}(\sigma)$;
   - t_P is $= 2\, t_{exp}(\sigma)$, with random exponents from $\mathbb{Z}_q$;
   - t_C is $\leq t_{exp}(\lambda) + 2\, t_{mVerify}(\sigma) + 2$ multiplications in G and 1 multiplication in $\mathbb{Z}_q$;
   - $cc = 4$ elements in G and $mc = 2$.

The main takeaway from Theorem 3.1 is that C outsources the computation of an exponentiation with a σ-bit exponent to S while C only performs an exponentiation with a λ-bit exponent, 2 group membership verifications in G, 2 multiplications in G and 1 modular multiplication in $\mathbb{Z}_q$. Also remarkable are the running time of S, who only performs 2 exponentiations and 2 group membership proof generations in G, and of the offline phase, where only 2 known-base exponentiations with random exponents are needed. Finally, the protocol only requires 2 messages, which is clearly minimal in this model, and only requires the communication of 4 elements in G.

Informal description of protocol (C, S)**.** The main challenge in coming up with our desired protocol consists of allowing C to efficiently verify computations performed by the possibly malicious server. It should be noted that general conversion techniques are known in the cryptography literature to transform a protocol secure against a honest party into one secure against a malicious one. Typically, these techniques are based on zero-knowledge proofs of knowledge of secrets that certify the correctness of the computation [19]. In their most general version, these techniques do not perform well with respect to many efficiency metrics. Even considering their most simplified version, basic proofs of knowledge of exponents in the literature (such as, e.g., [9]) require the verifier to perform group exponentiations, which is precisely what the client is trying to delegate. Accordingly, we need a substantially different protocol technique to allow the client's efficient verification of the server's computations.

Our main idea consists of a probabilistic verification equation which is verifiable using only a small number of modular multiplications. More specifically, C injects an additional random element in the inputs on which S is asked to compute the value of function F, so to satisfy the following properties: (a) if S returns correct computations of F, then C can use these random values to correctly compute y; (b) if S returns incorrect computations of F, then S either does not meet some deterministic verification equation or can only meet C's probabilistic verification equation for one possible value of the random elements; (c) C's messages hide the values of the random element as well as C's input to the function. By choosing a large enough domain (i.e., $\{1, \ldots, 2^\lambda\}$) from which this random value is chosen, the protocol achieves a very small security probability (i.e., $2^{-\lambda}$). As this domain is much smaller than G, this results in a considerable efficiency gain on C's running time.

Our probabilistic verification equation involves the answer from S, a correct exponentiation available to C (and computed in an offline phase by C or another trusted party), and the potential function output computed using the answer from S and another correct exponentiation computed in the offline phase. Only one of these values is exponentiated by C, and to an exponent that is $\leq 2^\lambda$, which can be much smaller than $|G|$. In itself, this probabilistic test is not always successful, but we characterize the condition under which it fails, and this condition can be expressed as 2 computationally simple deterministic tests: a value distinctness test and a group membership test.

The value distinctness test can be efficiently verified by C with no exponentiation or multiplication in G, but it introduces a minor case (i.e., C's input is $= 0$) where the probabilistic test is not passed; however, in this case, to preserve the protocol's correctness, C calculates the function $F_{exp,g,q}$ by himself (i.e., by setting $F_{exp,g,q}(0) = g^0 = 1$), and ignores all other verifications.

The group membership test is realized via the assumed efficiently verifiable group membership protocol. While we do not know of such a protocol for any arbitrary cyclic group, we showed in Section 2 that groups commonly used in cryptography have one.

Formal description of protocol (C, S)**.** Let G be an efficient cyclic group, and let (mProve,mVerify) denote its efficiently verifiable membership protocol.

Input to S: $1^\sigma, 1^\lambda, desc(F_{exp,g,q})$

Input to C: $1^\sigma, 1^\lambda, desc(F_{exp,g,q}), x \in \mathbb{Z}_q$

Offline phase instructions:

```
(1) Randomly choose $u_i \in \mathbb{Z}_q$, for $i = 0, 1$
(2) Set $v_i = g^{u_i}$ and store $(u_i, v_i)$ on $C$, for $i = 0, 1$
```

Online phase instructions:

```
(1) $C$ randomly chooses $b \in \{1, \ldots, 2^\lambda\}$
    $C$ sets $z_0 := (x - u_0) \mod q$, $z_1 := (b \cdot x + u_1) \mod q$
    $C$ sends $z_0, z_1$ to $S$
(2) $S$ computes $w_i := g^{z_i}$ and $\pi_i$ :=mProve($w_i$), for $i = 0, 1$
    $S$ sends $w_0, w_1, \pi_0, \pi_1$ to $C$
(3) If $x = 0$
        $C$ returns: $y = 1$ and the protocol halts
    if mVerify($w_i, \pi_i$) = 0 for some $i \in \{0, 1\}$, then
        $C$ returns: $\perp$ and the protocol halts
    $C$ computes $y := w_0 * v_0$
    $C$ checks that
        $y \neq 1$, also called the 'distinctness test'
        $w_1 = y^b * v_1$, also called the 'probabilistic test'
        mVerify($w_0, \pi_0$) = mVerify($w_1, \pi_1$) = 1,
            also called the 'membership test'
```

```
if any one of these tests is not satisfied then
    $C$ **returns:** $\perp$ and the protocol halts
$C$ **returns:** $y$
```

Properties of protocol (C, S): *The efficiency properties* are verified by protocol inspection.

- *Round complexity:* the protocol only requires one round, consisting of one message from C to S followed by one message from S to C.
- *Communication complexity:* the protocol requires the transfer of 2 elements in G and 2 proofs of group membership from S to C, and 2 elements in $\mathbb{Z}_q$ from C to S.
- *Runtime complexity:* During the offline phase, 2 exponentiations in base g and with random σ-bit exponents are performed. Note that known-base exponentiations can be executed faster than unknown-base ones using pre-computation techniques (see, e.g.,[7, 22]). During the online phase, S computes 2 exponentiations to σ-bit exponents in G and 2 group membership proofs; and C verifies 2 group membership proofs and computes 2 multiplications in G, 1 modular multiplication in $\mathbb{Z}_q$, and 1 exponentiation in G to a random exponent that is much smaller ($\leq 2^\lambda$) than 2^σ.

The *correctness* property follows by showing that if C and S follow the protocol, C always outputs $y = g^x$. First, assume $x = 0$; in this case, C returns $y = 1 = g^0$ at the beginning of step 3. Now, assuming $x \neq 0$, we show that the 3 tests performed by C are always passed. The membership test is always passed since w_i is computed by S as g^{z_i}, for $i = 0, 1$, and g is a generator of group G; the probabilistic test is always passed since

$$w_1 = g^{z_1} = g^{bx+u_1} = (g^x)^b g^{u_1} = y^b v_1.$$

The distinctness test is always passed, since we assume $x \neq 0$, which implies that $-u_0 \neq (x - u_0) \mod q$, which implies, using the fact that g is a generator and has thus order q, that $v_0^{-1} = g^{-u_0} \neq g^{x-u_0} = g^{z_0} = w_0$, equivalently saying that $y = w_0 * v_0 \neq 1$. This implies that C never returns $\perp$, and thus returns y. To see that this returned value y is the correct output, note that

$$y = w_0 * v_0 = g^{z_0} * g^{u_0} = g^{x-u_0} * g^{u_0} = g^x.$$

The *privacy* property of the protocol against any arbitrary malicious S follows by observing that C's only message to S does not leak any information about x. This message is a pair (z_0, z_1) where $z_0 = (x - u_0) \mod q$, $z_1 = (bx + u_1) \mod q$, and z_0 and z_1 are uniformly and independently distributed in $\mathbb{Z}_q$, as so are u_0 and u_1. Then, no information is leaked by z_0, z_1 about x as: (a) for any $x \in \mathbb{Z}_q$, there is exactly one u_0 corresponding to z_0; that is, $u_0 = x - z_0 \mod q$; (b) for any $x \in \mathbb{Z}_q$, for any $b \in \{1, \dots, 2^\lambda\}$ chosen by C, there is exactly one u_1 corresponding to z_1; that is, $u_1 = z_1 - bx \mod q$. This implies that, since u_0, u_1 are uniformly and independently distributed in $\mathbb{Z}_q$, the distribution of x conditioned on z_0, z_1 is also uniform in $\mathbb{Z}_q$. Moreover, by essentially the same proof, protocol (C, S) satisfies the following property: for any x, z_0 and z_1 do not leak any information about b. We will use this latter privacy property in the proof of the security property.

To prove the *security* property against any malicious S we need to compute an upper bound ϵ_s on the security probability that S convinces C to output a y such that $y \neq F_{exp,g,q}(x)$. If $x = 0$, C can calculate $F_{exp,g,q}(x) = g^0 = 1$ and it does not need to check whether S is honest or dishonest. Thus $\epsilon_s = 0$ when $x = 0$. Now assume that $x \neq 0$. We start by defining the following events with respect to a random execution of (C, S) where C uses x as input:

- $e_{y,\neq}$, defined as 'C outputs y such that $y \neq F_{exp,g,q}(x)$'
- $e_\perp$, defined as 'C outputs $\perp$'

By inspection of (C, S), we directly obtain the following fact.

Fact 3.1. If event $e_{y,\neq}$ happens then event $(\neg\, e_\perp)$ happens.

With respect to a random execution of (C, S) where C uses x as input, we now define the following events:

- $e_{1,b}$, defined as '$\exists$ exactly one b such that S's message (w_0, w_1) satisfies $w_1 = (w_0 * v_0)^b * v_1$'
- $e_{>1,b}$, defined as '$\exists$ more than one b such that S's message (w_0, w_1) satisfies $w_1 = (w_0 * v_0)^b * v_1$'.

By definition, events $e_{1,b}, e_{>1,b}$ are each other's complement event.

In our proof of the privacy property of (C, S), we proved that for any x, C's message (z_0, z_1) does not leak any information about b. This implies that all values in $\{1, \dots, 2^\lambda\}$ are still equally likely even when conditioning over message (z_0, z_1). Then, if event $e_{1,b}$ is true, the probability that S's message (w_0, w_1) satisfies the probabilistic test, is 1 divided by the number 2^λ of values of b that are still equally likely even when conditioning over message (z_0, z_1). We obtain the following

Fact 3.2. Prob $\left[\, \neg\, e_\perp | e_{1,b} \,\right] \leq 1/2^\lambda$

We now show the main technical claim, saying that if S is malicious then it cannot produce in step 2 of the protocol values w'_0, w'_1 satisfying all of C's 3 tests relatively to two distinct values $b_1, b_2 \in \{1, \dots, 2^\lambda\}$:
Since S can be malicious, in step 2 it can send arbitrary values to C. Differently saying, C can send w'_i for $i = 0, 1$ for $w'_i = w_i$ or $w'_i \neq w_i$, where $w_i = g^{z_i}$. Since the group G is cyclic, g is generator of G and C uses π_0, π_1 to check in step 3 that $w'_i \in G$, we can write

$$w'_0 = g^u * w_0 \text{ and } w'_1 = g^v * w_1 \text{ for some } u, v \in \mathbb{Z}_q$$

then $y = w'_0 * v_0 = g^u * w_0 * v_0 = g^u * g^x$. Now, recall that the goal of a malicious S is to pass C's three verification tests and force C's output to be $y \neq g^x$; then, assume that $u \neq 0 \mod q$. Now, consider the following equivalent rewritings of C's probabilistic test, obtained by variable substitutions and simplifications:

$$\begin{aligned} w'_1 &= y^b * v_1 \\ g^v * w_1 &= (g^u * g^x)^b * g^{u_1} \\ g^v * g^{z_1} &= g^{ub} * g^{bx+u_1} \\ g^v * g^{bx+u_1} &= g^{ub} * g^{bx+u_1} \\ g^v &= g^{ub} \\ v &= ub \mod q. \end{aligned}$$

Notice that if $u = 0 \mod q$ then the above calculation implies that $v = 0 \mod q$, and thus S is honest, from which we derive that $\epsilon_s = 0$. Now consider the case S is dishonest, in which case we have that $u \neq 0 \mod q$. We want to show that b is unique in this case. If there exist two distinct b_1 and b_2 such that

$$ub_1 = v \mod q \text{ and } ub_2 = v \mod q$$

then $u(b_1 - b_2) = 0 \mod q$ then $b_1 - b_2 = 0 \mod q$ (i.e $b_1 = b_2$) because $u \neq 0 \mod q$. This shows that b is unique.

We obtain the following fact.

Fact 3.3. $\text{Prob}\left[\, e_{>1,b} \,\right] = 0$

The rest of the proof consists of computing an upper bound ϵ_s on the probability of event $e_{y,\neq}$. We have the following

$$\begin{aligned}
\text{Prob}\left[\, e_{y,\neq} \,\right] &\leq \text{Prob}\left[\, \neg\, e_{\perp} \,\right] \\
&= \text{Prob}\left[\, e_{1,b} \,\right] \cdot \text{Prob}\left[\, \neg\, e_{\perp} | e_{1,b} \,\right] \\
&\quad + \text{Prob}\left[\, e_{>1,b} \,\right] \cdot \text{Prob}\left[\, \neg\, e_{\perp} | e_{>1,b} \,\right] \\
&= \text{Prob}\left[\, e_{1,b} \,\right] \cdot \text{Prob}\left[\, \neg\, e_{\perp} | e_{1,b} \,\right] \\
&\leq \text{Prob}\left[\, e_{1,b} \,\right] \cdot \frac{1}{2^{\lambda}} \\
&\leq \frac{1}{2^{\lambda}},
\end{aligned}$$

where the first inequality follows from Fact 3.1, the first equality follows from the definition of events $e_{1,b}, e_{>1,b}$ and the conditioning rule, the second equality follows from Fact 3.3, and the second inequality follows from Fact 3.2.

We finally obtain that $\epsilon_s = \text{Prob}\left[\, e_{y,\neq} \,\right] = 2^{-\lambda}$, which concludes the proof for the security property for (C, S). □

Remark. Although we have only analyzed our protocol (C, S) with respect to a single execution, we note that the proofs of its properties naturally extend to multiple sequential, parallel or concurrent executions of the same protocol (both offline and online phase).

4 OUTSOURCING EXPONENTIATION IN ANY EFFICIENT CYCLIC GROUP WITH TRADEOFF RUNTIME PERFORMANCE

In this section we present our second protocol for outsourcing exponentiation in cyclic groups. This protocol can be seen as an extension of the first protocol, from Section 3, trying to further reduce the client's runtime, even at the cost of slightly increasing other computation metrics. The underlying motivation here is that in many applications slightly increasing a cloud server's time complexity may be worth to obtain further reduced client's time complexity. We first formally state this subsection's result, then describe the protocol, and finally prove its correctness, security, privacy and efficiency properties.

Formal theorem statement. We actually show a class of protocols varying according to a parameter $m \geq 1$, where in the case $m = 1$ we have the same protocol as in Section 3. Formally, we show the following

Theorem 4.1. Let $(G, *)$ be an efficient cyclic group, let σ be its computational security parameter, let λ be a statistical security parameters, let integer m be a protocol parameter, and let $\lambda' = \lceil \lambda/m \rceil$. There exists (constructively) a client-server protocol (C, S) for outsourced computation of function $F_{exp,g,q}$ which satisfies

1. δ_c-correctness, for $\delta_c = 1$;
2. ϵ_s-security, for $\epsilon_s = 2^{-\lambda}$;
3. ϵ_p-privacy, for $\epsilon_p = 0$;
4. efficiency with parameters $(t_F, t_S, t_P, t_C, cc, mc)$, where
   - t_F is $= t_{exp}(\sigma)$;
   - t_S is $= t_{m+1,exp}(\sigma) + (m+1) \cdot t_{mProve}(\sigma)$;
   - t_P is $= t_{m+1,exp}(\sigma)$;
   - t_C is $= t_{m,exp}(\lambda') + (m+1)$ multiplications in G + m multiplications in $\mathbb{Z}_q$ + $(m+1) \cdot t_{mVerify}(\sigma)$;
   - $cc = m+1$ elements in $\mathbb{Z}_q$, $m+1$ elements and membership proofs in G;
   - $mc = 2$.

The main takeaway from Theorem 4.1 is that C delegates an exponentiation with a σ-bit exponent to S while C only performs m same-base exponentiations with a λ'-bit exponent, $m+1$ multiplications and group membership verifications in G and m modular multiplications in $\mathbb{Z}_q$. Here, note that exponent λ' is $\lceil \lambda/m \rceil$, and that m same-base exponentiations can be performed faster than independently repeating m times the same exponentiation algorithm, by using optimized algorithms. The protocol still requires only 2 messages, which is minimal in this model. On the other hand, the running time of S has increased, in that S now performs $m+1$ exponentiations and generates $m+1$ group membership proofs in G. Similarly, the running time of the offline phase has increased, as now $m+1$ known-base exponentiations with random exponents are needed. The communication complexity also has increased to $m+1$ elements in $\mathbb{Z}_q$ and $m+1$ elements in G. While these latter 3 metrics increase with respect to the protocol in Section 3, it should be noted that parameter m can be chosen by the system designer, to select the desired tradeoff (as further discussed in Section 5).

Informal description of protocol (C, S)**.** In Section 3 we presented a protocol which satisfies δ_c-correctness, ϵ_p-privacy and ϵ_s-security, with $\delta_c = 1$, $\epsilon_p = 0$, and $\epsilon_s = 2^{-\lambda}$, with the following running time parameters: $t_C \leq t_{exp}(\lambda) + 2$ group multiplications, $t_S = 2$ group exponentiations, and $t_P = 2$ group exponentiations. In this section we design protocols that further decrease the value of t_C while keeping the same values for $\delta_c, \epsilon_p, \epsilon_s$, and only slightly increasing t_S, t_P.

The main idea in our new protocol is based on a detailed consideration of the role of the probabilistic verification equation '$w = y^b v$' in the protocol of Section 3. Specifically, it is possible to use $m > 1$ such equations, each with the same base y and an independent value of the random exponent $b(i)$, while:

(1) the worst-case number of C's multiplications increases by a factor strictly smaller than m;
(2) the security analysis performed using a single probabilistic verification equation can be extended to all m equations, thus reducing the security probability by an exponential factor of m.

Then, since it suffices to target a security probability of $2^{-\lambda}$ for the resulting protocol, we can choose a smaller, by a factor of m, domain (i.e., $\{1, \ldots, 2^{\lceil \lambda/m \rceil} - 1\}$) for the random exponents $b(1), \ldots, b(m)$. The efficiency gain is in the observation that performing m probabilistic verifications can require a number of C's group multiplication smaller than m times those needed in a single probabilistic verification. This is because any preprocessing techniques, including those described in [7, 27], can be run online and become effective if used to compute multiple same-base exponentiations. On the other hand, performing m probabilistic verification equations requires $m+1$ offline exponentiations from C and $m+1$ exponentiations from S (here, a direct m-fold repetition of the protocol from Section 3 would actually incur a number of $2m$ exponentiations but we reduce this number to $m+1$ by observing that the security property continues to hold even when using the same w_0 across all m equations). The actual efficiency tradeoff achieved by the resulting protocol (i.e., the reduction in the number of C's group multiplications and the increase in the number of offline exponentiations to random exponents and S's exponentiations) will vary depending on the specific algorithm used to compute m same-base exponentiations. In what follows, we describe protocol (C,S) based on a generic multiple-exponentiation algorithm and prove its correctness, privacy and security properties. Later, in Section 5, we analyze the protocol's performance metrics based on specific instantiations of this algorithm, and for different values of m (which can be chosen by the system designer to calibrate the tradeoff); in particular, we show that $m = 5$ minimizes the number t_C of C's group multiplications.

Formal description of protocol (C, S)**.**
Let ManyExp$(z, x(1), \ldots, x(s), \beta)$ denote an algorithm that, on input $z \in G$ and exponents $x(1), \ldots, x(s) \in \{1, \ldots, 2^\beta\}$, computes exponentiations $z^{x(1)}, \ldots, z^{x(s)}$ in G. Algorithm ManyExp could be instantiated using s executions of the textbook square-and-multiply algorithm, or using improved techniques based on precomputations (here perfomed in the online phase), such as those from [7, 22, 27]. Let m be a protocol parameter that can be chosen by the system designer, and let $\lambda' = \lceil \lambda/m \rceil$.

Input to S: $1^\sigma, 1^\lambda$, $desc(F_{exp,g,q})$, parameter 1^m

Input to C: $1^\sigma, 1^\lambda$, $desc(F_{exp,g,q})$, $x \in \mathbb{Z}_q$, parameter 1^m

Offline phase instructions:

```
1. Uniformly choose $u_0, \ldots, u_m \in \mathbb{Z}_q$
2. Run algorithm ManyExp$(g, u_0, u_1, \ldots, u_m, \sigma)$
       to compute $(v_0, v_1, \ldots, v_m)$
3. Store $(u_0, v_0), (u_1, v_1), \ldots, (u_m, v_m)$ on $C$
```

Online phase instructions:

```
1. $C$ sets $z_0 = (x - u_0) \mod q$
   For $i = 1, \ldots, m$,
       $C$ uniformly chooses $b(i) \in \{1, \ldots, 2^{\lambda'}\}$
       $C$ sets $z_i = (b(i) \cdot x + u_i) \mod q$
   $C$ sends $z_0, z_1, \ldots, z_m$ to $S$
2. $S$ runs algorithm ManyExp$(g, z_0, z_1, \ldots, z_m, \sigma)$
       to compute $(w_0, w_1, \ldots, w_m)$
   For $j = 0, \ldots, m$,
       $S$ computes $\pi_j$ = mProve$(w_j)$
   $S$ sends $(w_0, \pi_0), (w_1, \pi_1), \ldots, (w_m, \pi_m)$ to $C$
3. If $x = 0$
       $C$ **returns:** $y = 1$ and the protocol halts
   $C$ computes $y = w_0 * v_0$
   $C$ checks that $y \neq 1$, also called the 'distinctness test'
   $C$ checks that mVerify$(w_i, \pi_i)$ =1, for $i = 0, 1, \ldots, m$,
       also called the 'membership test'
   $C$ runs ManyExp$(y, b(1), \ldots, b(m), \lambda')$
       to compute $y^{b(1)}, \ldots, y^{b(m)}$
   For $i = 1, \ldots, m$,
       $C$ checks that $w_i = y^{b(i)} * v_i$,
           also called the '$i$-th probabilistic test'
   if any one of the above checks is not satisfied then
       $C$ **returns:** $\perp$ and the protocol halts.
4. $C$ **returns:** $y$
```

Properties of protocol (C, S)**:** *The efficiency properties* are verified by protocol inspection.

- *Round complexity.* The protocol only requires one round, consisting of one message from C to S followed by one message from S to C.
- *Communication complexity.* The protocol requires the transfer of $m+1$ elements in G and membership proofs from S to C and $m+1$ elements in $\mathbb{Z}_q$ from C to S.
- *Runtime complexity.* Only $\leq m+1$ exponentiations in G with the same base and random σ-bit exponents are performed during the offline phase. During the online phase, S computes $m+1$ exponentiations in G with the same base and random σ-bit exponents, and $m+1$ membership proofs, while C performs m multiplications in $\mathbb{Z}_q$, $m+1$ multiplications in G, $m+1$ membership proof verifications and computes m exponentiations in G with the same base and random λ'-bit exponents.

The correctness property follows by showing that if C and S follow the protocol, C always outputs $y = g^x$. This is proved for the case $x = 0$ exactly as done for the protocol in Section 3. Then, assume $x \neq 0$. We can prove that C passes the G-membership test and the distinctness test exactly as done for the protocol in Section 3. Even the i-th probabilistic test is passed, for all $i = 1, \ldots, m$, since

$$w_i = g^{z_i} = g^{b(i)x+u_i} = (g^x)^{b(i)} * g^{u_i} = y^{b(i)} * v_i.$$

This implies that C never returns $\perp$, and thus returns $y = F_{exp,g,q}(x)$, since

$$y = w_0 * v_0 = g^{z_0} * g^{u_0} = g^{x-u_0} * g^{u_0} = g^x.$$

The *privacy* property of the protocol against a malicious S follows, similarly as for the protocol in Section 3. Note that C's only message to S is a tuple $(z_0, \ldots, z_m)$ where $z_0 = (x - u_0) \mod q$ and $z_i = (b(i) \cdot x + u_i) \mod q$, for $i = 1, \ldots, m$. Here, values $u_0, u(1), \ldots, u(m)$ are uniformly and independently distributed in $\mathbb{Z}_q$, and thus so are $z_0, z(1), \ldots, z_m$. This is the cases for any x and for any $b(1), \ldots, b(m) \in \{1, \ldots, 2^{\lambda'}\}$. Thus, we obtain that the protocol satisfies the following two properties for all $j = 0, \ldots, m$: (1) for any x, $z_0, z_1, \ldots, z_m$ are uniformly and independently distributed in $\mathbb{Z}_q$; and (2) for any x and any $b(1), \ldots, b(m) \in \{1, \ldots, 2^{\lambda'}\}$, values $z_0, z_1, \ldots, z_m$ are uniformly and independently distributed in $\mathbb{Z}_q$. Property (1) implies that C's message does not leak any information about x, and property (2) implies that C's message does not leak any information about x and $b(1), \ldots, b(m)$. The first fact suffices to imply the privacy property. The second fact will be used in the proof of the security property.

To prove the *security* property against a malicious S, we need to compute an upper bound ϵ_s on the security probability that S convinces C to output a y such that $y \neq F_{exp,g,q}(x)$. If $x = 0$, C can calculate $F_{exp,g,q}(x) = g^0 = 1$ and it does not need to check whether S is honest or dishonest. Thus $\epsilon_s = 0$ when $x = 0$. Now, assume that $x \neq 0$. We start by defining the following events with respect to a random execution of (C, S) where C uses x as input:

- $e_{y,\neq}$, defined as 'C outputs y such that $y \neq F_{exp,g,q}(x)$'
- $e_{\perp}$, defined as 'C outputs $\perp$'
- $e_{\perp,i}$, defined as 'The value w_i sent by S does not satisfy the i-th probabilistic check'

By inspection of (C, S), we directly obtain the following facts.

Fact 4.1. If event $e_{y,\neq}$ happens then event $(\neg\, e_{\perp})$ happens.

Fact 4.2. If event $\neg e_{\perp}$ happens then event $((\neg\, e_{\perp,1}) \wedge \ldots \wedge (\neg\, e_{\perp,m}))$ happens.

With respect to a random execution of (C, S) where C uses x as input, we now define the following events for all $i = 1, \ldots, m$:

- $e_{1,b(i)}$, defined as '$\exists$ exactly one $b(i)$ such that S's message $(w_0, \ldots, w_m)$ satisfies $w_i = (w_0 * v_0)^{b(i)} * v_i$'
- $e_{>1,b(i)}$, defined as '$\exists$ more than one $b(i)$ such that S's message $(w_0, \ldots, w_m)$ satisfies $w_i = (w_0 * v_0)^{b(i)} * v_i$'.

By definition, events $e_{1,b(i)}, e_{>1,b(i)}$ are each other's complement event, for all $i = 1, \ldots, m$.

In our proof of the privacy property of (C, S), we proved that for any x, C's message $z_0, z_1, \ldots, z_m$ does not leak any information about $b(1), \ldots, b(m)$. By recalling that values $b(1), \ldots, b(m)$ are randomly and independently chosen by C, we obtain that all values in $\{1, \ldots, 2^{\lambda'}\}$ are still equally likely for each $b(i)$ even when conditioning over message $(z_0, z_1, \ldots, z_m)$ and over values $b(1), \ldots, b(i-1)$. Then, for each $i = 0, 1, \ldots, m$, if event $e_{1,b(i)}$ is true, the probability that values $(w_0, w_1, \ldots, w_m)$ in S's message satisfy the i-th probabilistic test, is 1 divided by the number $2^{\lambda'}$ of values of $b(i)$ that are still equally likely even after conditioning over message $(z_0, z_1, \ldots, z_m)$ and over values $b(1), \ldots, b(i-1)$.

We obtain the following

Fact 4.3. For all $i = 1, \ldots, m$, it holds that $\text{Prob}\,[\, d_i \mid d_1 \wedge \ldots \wedge d_{i-1}\,] \leq 2^{-\lambda'}$, where d_i denotes event $\neg\, e_{\perp,i} \wedge e_{1,b(i)}$.

By applying the rule of conditional probability, we can write

$$\text{Prob}\,[\, d_1 \wedge \ldots \wedge d_m\,] = \prod_{i=1}^{m} \text{Prob}\,[\, d_i \mid d_1 \wedge \ldots \wedge d_{i-1}\,]\,.$$

By further using the result from Fact 4.3, and the expression $\lambda' = \lceil \lambda/m \rceil$, we obtain the following

Fact 4.4. It holds that $\text{Prob}\,[\, d_1 \wedge \ldots \wedge d_m\,] \leq 2^{-\lambda}$, where d_i denotes event $\neg\, e_{\perp,i} \wedge e_{1,b(i)}$, for all $i = 1, \ldots, m$.

We continue with the main technical claim, saying that if S is malicious then it cannot produce in step 2 of the protocol values $w'_0, \ldots, w'_m$ satisfying all of C's three tests relatively to two distinct values $b_1(i), b_2(i) \in \{1, \ldots, 2^{\lceil \frac{\lambda}{m} \rceil}\}$ for each $i = 1, \ldots, m$:
Since S can be malicious, in step 2 it can send correct answer or incorrect answer. Differently saying, it can send w'_j where $w'_j = w_j$ or $w'_j \neq w_j$ where $w_i = g^{z_j}$ for each $j = 0, \ldots m$. Since the group G is cyclic, g is generator of G and C checks in step 3 that $w'_j \in G$, we can write

$$w'_j = g^{\alpha_j} * w_j \text{ for some } \alpha_j \in \mathbb{Z}_q \text{ for } j = 0, \ldots, m$$

then $y = w'_0 * v_0 = g^{\alpha_0} * w_0 * v_0 = g^{\alpha_0} * g^x$. The goal of malicious S to pass all three checks and C's output $y \neq g^x$ then $\alpha_0 \neq 0 \mod q$. Now, consider 'probabilistic check' for each $i = 1, \ldots, m$:

$$\begin{aligned} w'_i &= y^{b(i)} * v_i \\ g^{\alpha_i} * w_i &= (g^{\alpha_0} * g^x)^{b(i)} * g^{u_i} \\ g^{\alpha_i} * g^{z_i} &= g^{\alpha_0 b(i)} * g^{b(i)x + u_i} \\ g^{\alpha_i} * g^{b(i)x+u_i} &= g^{\alpha_0 b(i)} * g^{b(i)x+u_i} \\ g^{\alpha_i} &= g^{\alpha_0 b(i)} \\ \alpha_i &= \alpha_0 \cdot b(i) \mod q \end{aligned}$$

Notice that if $\alpha_0 = 0 \mod q$ then $\alpha_i = 0 \mod q$ for all $i = 1, \ldots, m$ from above calculation which implies that S is honest then automatically $\epsilon_s = 0$. If $\alpha_i = 0 \mod q$ then $\alpha_0 = 0$ because $b(i) \neq 0$ for $i \in \{1, \ldots, m\}$. Now, if there exist some $j \in \{1, \ldots, m\}$ such that $\alpha_j \neq 0$, in this case we want to show that $b(j)$ is unique. If there exist two distinct $b_1(j)$ and $b_2(j)$ such that

$$\alpha_j b_1(j) = \alpha_0 \mod q \text{ and } \alpha_j b_2(j) = \alpha_0 \mod q$$

then $\alpha_j(b_1(j) - b_2(j)) = 0 \mod q$ then $b_1(j) - b_2(j) = 0 \mod q$ (i.e $b_1(j) = b_2(j)$) because $\alpha_j \neq 0 \mod q$. Which shows that each $b(i)$ is unique for all $i = 1, \ldots, m$.

We obtain the following fact.

Fact 4.5. $\text{Prob}\left[\, e_{>1,b(i)}\,\right] = 0$

The rest of the proof consists of computing an upper bound ϵ_s on the probability of event $e_{y,\neq}$. We have the following

$$\begin{aligned} &\text{Prob}\left[\, e_{y,\neq}\,\right] \\ &\quad \leq \text{Prob}\left[\, \neg\, e_{\perp}\,\right] \\ &\quad = \text{Prob}\left[\, \neg\, e_{\perp} \wedge (e_{1,b(1)} \wedge \ldots \wedge e_{1,b(m)})\,\right] \\ &\qquad + \text{Prob}\left[\, \neg\, e_{\perp} \wedge \neg(e_{1,b(1)} \wedge \ldots \wedge e_{1,b(m)})\,\right] \\ &\quad \leq \text{Prob}\left[\, (\neg\, e_{\perp,1} \wedge e_{1,b(1)}) \wedge \ldots \wedge (\neg\, e_{\perp,m} \wedge e_{1,b(m)})\,\right] \\ &\qquad + \text{Prob}\left[\, \neg\, e_{\perp} \wedge (e_{>1,b(1)} \vee \ldots \vee e_{>1,b(m)})\,\right] \\ &\quad = \text{Prob}\left[\, (\neg\, e_{\perp,1} \wedge e_{1,b(1)}) \wedge \ldots \wedge (\neg\, e_{\perp,m} \wedge e_{1,b(m)})\,\right] \\ &\qquad + \text{Prob}\left[\, (\neg\, e_{\perp} \wedge e_{>1,b(1)}) \vee \ldots \vee (\neg\, e_{\perp} \wedge e_{>1,b(m)}))\,\right] \\ &\quad \leq \prod_{i=1}^{m} 2^{-\lambda'} + \sum_{i=1}^{m} \text{Prob}\left[\, \neg\, e_{\perp} \wedge e_{>1,b(i)}\,\right] \\ &\quad = (2^{-\lambda'})^m + \sum_{i=1}^{m} \text{Prob}\left[\, e_{>1,b(i)}\,\right] \cdot \text{Prob}\left[\, \neg\, e_{\perp} | e_{>1,b(i)}\,\right] \\ &\quad = 2^{-\lambda} \end{aligned}$$

where the above equalities and inequalities are explained as follows. The first inequality is derived from Fact 4.1. The first equality follows from partitioning the event $\neg\, e_{\perp}$ into two disjoint events, using the definition of events $e_{1,b(i)}, e_{>1,b(i)}$. The second inequality follows from Fact 4.2. The second equality is obtained by a distributive rule. The third inequality follows from Fact 4.4 and a union bound. The third equality follows from the conditioning rule. The last equality follows from Fact 4.5 and by the definition of λ'.

We finally obtain that $\epsilon_s = \text{Prob}\left[\, e_{y,\neq} \,\right] \leq 2^{-\lambda}$, which concludes the proof of the security property for (C, S). □

5 PERFORMANCE ANALYSIS

In this section we describe parametric (as a function of parameters σ, λ, m) and numeric performance evaluations of our protocols in Section 3, 4. We also compare these two protocols with the non-outsourced computation of the same function.

So far we have expressed the performance of our protocols in terms of group multiplications and two parameter functions: $t_{exp}(\ell)$, the number of multiplications in G to compute one exponentiation to an arbitrary ℓ-bit exponent; and $t_{m,exp}(\ell)$, the number of multiplications in G to compute m exponentiations of the same group value to m arbitrary ℓ-bit exponents. A more concrete evaluation of the performance of our protocols requires a (possibly optimized) instantiation of these two functions. As there can be different algorithms for the computation of one exponentiation or of m exponentiations of the same base to m arbitrary ℓ-bit exponents, we opt for the following two representative istantiations:

(1) *Basic setting:* using the textbook square-and-multiply algorithm to evaluate group exponentiation, and computing m exponentiations by simply independently applying this same algorithm m times, we can set
   - $t_{exp}(\ell) = 2\ell$
   - $t_{m,exp}(\ell) = 2m\ell$

(2) *Improved setting*: using improved algorithms from the literature to evaluate group exponentiation, we can obtain even asymptotic improvements over the above expressions in the basic setting. In particular, we use the improved algorithms discussed in [27], which contains a detailed literature account and closed-form estimates for the number of multiplications of the main discussed algorithms. We obtain that:
   - $t_{exp}(\ell)$ is about $\ell(1+\frac{1}{\log \ell})$, using Brauer's 1939 algorithm, as described in [27],
   - $t_{m,exp}(\ell)$ is about $\ell(1+\frac{m}{\log \ell})$, using Yao's 1976 algorithm, as described in [27].

   We also refer the reader to [7, 22, 28, 29] for other algorithms claiming improvements, although note that these papers do not provide additional closed-form evaluations.

The table below compares the performance of our outsourcing protocols in Section 3 and Section 4 with a non-outsourced computation of the client under both basic (B) and improved (I) settings of functions $t_{exp}, t_{m,exp}$. The table reports expressions (as a function of $\sigma, \lambda, m, \lambda'$, for efficiency metrics t_F (the number of multiplications to compute function $F_{exp,g,q}$), t_P (the number of multiplications used in the protocol's offline phase), t_C (the number of multiplications by C in the protocol's online phase), t_S (the number of multiplications by S in the protocol's online phase), ϵ_p (the probability parameter in the privacy definition), and ϵ_s (the probability parameter in the security definition).

For simplicity of description, we only consider the case when the group membership protocols do not add any multiplication (as shown in Example 1 of Section 2, when $G = \mathbb{Z}_p^*$).

Perf	**B/I**	$F_{exp,g,q}$ **with no Outsourcing**	$F_{exp,g,q}$ **with Outsourcing**	
			Section 3	*Section 4*
t_F	B	2σ	2σ	2σ
	I	$\sigma(1+\frac{1}{\log\sigma})$	$\sigma(1+\frac{1}{\log\sigma})$	$\sigma(1+\frac{1}{\log\sigma})$
t_P	B	0	4σ	$2\sigma(m+1)$
	I	0	$\sigma(1+\frac{2}{\log\sigma})$	$\sigma(1+\frac{m+1}{\log\sigma})$
t_C	B	2σ	$2\lambda+3$	$2m\lambda'+2m+1$
	I	$\sigma(1+\frac{1}{\log\sigma})$	$\lambda(1+\frac{1}{\log\lambda})+3$	$\lambda'(1+\frac{m}{\log\lambda'})+2m+1$
t_S	B	0	4σ	$2\sigma(m+1)$
	I	0	$\sigma(1+\frac{2}{\log\sigma})$	$\sigma(1+\frac{m+1}{\log\sigma})$
ϵ_p	B & I	0	0	0
ϵ_s	B & I	0	$2^{-\lambda}$	$2^{-\lambda}$

Table 1: Parametric performance of our protocols.

The analysis for Example 2, when $G = G_q \subseteq \mathbb{Z}_p^*$, for $p = 2q + 1$ and p, q primes, is similarly obtained as it only adds $m + 1$ exponentiations in G for S and $m + 1$ multiplications in G for C.

We believe the main takeaways by analyzing the expressions in the above table are as follows:

- our protocol in Section 3 reduces t_C by a multiplicative factor of about σ/λ with respect to non-outsourced computation, when using both basic and improved settings;
- the protocol in Section 4 reduces t_C by a multiplicative factor of: (1) about σ/λ with respect to non-outsourced computation, when using the basic parameter settings, and (2) about σ/λ' with respect to non-outsourced computation, when using the improved parameter settings, where $\lambda' = \lceil \lambda/m \rceil$ by definition, and $m \geq 1$ can be chosen by the system designer;
- the protocol in Section 4 reduces t_C by a multiplicative factor of about λ/λ' with respect to the protocol in Section 3, however by increasing parameters t_P and t_S by a multiplicative factor of about m times, when using both basic and improved settings;
- the impact of using basic or improved settings for functions $t_{exp}, t_{m,exp}$ is only on lower-order terms of the performance metrics t_F, t_P, t_C, t_S.

As for the numeric analysis, in Figure 1 and 2, we show the performance of our protocols in Sections 3 (when $m = 1$) and Section 4 (for any $m \geq 1$). We set $\sigma = 2048$ and $\lambda = 128$, as these are the currently most recommended settings, and use the improved settings for functions $t_{exp}, t_{m,exp}$.

Figure 1 shows the number of multiplications by the client in the online phase (t_C) as m increases, both in the Example 1 case $G = Z_p^*$ and in the Example 2 case $G = G_q$. We notice that in both cases t_C decreases as m increases up to 5, and then either keeps the same value or increases for larger values of m.

Figure 2 shows the server's number of multiplications in the online phase (t_S) as m increases, both in the Example 1 case $G = Z_p^*$ and in the Example 2 case $G = G_q$. We note that as m increases, this

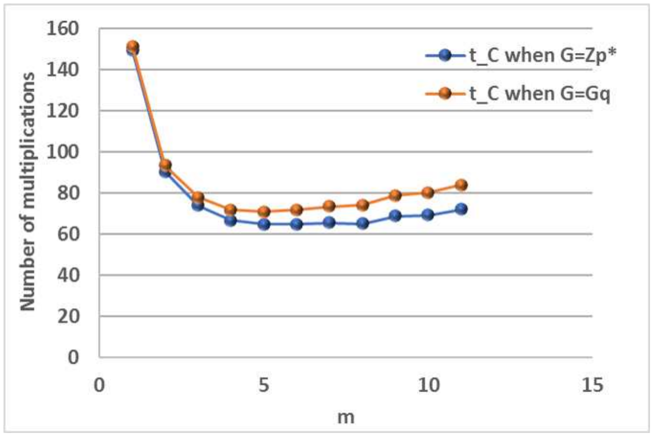

Figure 1: Metric t_C as a function of m

metric monotonically increases in both cases, exhibiting a much higher slope in the case $G = G_q$ than in the case $G = \mathbb{Z}_p^*$. This is because when $G = G_q$, the efficiently verifiable group membership protocol requires $m + 1$ variable-base exponentiations from S, which we cannot optimize using improved algorithms (working on multiple exponentiations on the same base).

As for metric t_P, we note that in both Example 1 and Example 2 cases, it is equal to the (plotted) value of t_S in the Example 1 case.

Interesting tradeoff points, obtained by instantiating parameters in the improved setting, are the following:

(1) $G = \mathbb{Z}_p^*$: $m = 1$, $t_C = 145$, $t_P = t_S = 2,420$, which minimizes t_S for Example 1;
(2) $G = \mathbb{Z}_p^*$: $m = 5$, $t_C = 65$, $t_P = t_S = 3,165$, which minimizes t_C for Example 1;
(3) $G = G_q$: $m = 1$, $t_C = 152$, $t_P = 2,420$, $t_S = 4,841$, which minimizes t_S for Example 2;
(4) $G = G_q$: $m = 5$, $t_C = 71$, $t_P = 3,165$, $t_S = 14,523$, which minimizes t_C for Example 2.

Finally, we considered benchmarks for modular multiplication and exponentiation modulo a 2048-bit prime executed using commodity computing resources and extrapolated the running time of t_C when $m = 5$ and $G = \mathbb{Z}_p^*$ as just below 0.24ms and the running time of t_C when $m = 5$ and $G = G_q$ as just above 0.26ms.

6 CONCLUSIONS

We considered the problem of outsourcing group exponentiation to a single, possibly malicious, server, originally left open in [20]. We solved this problem by showing protocols that provably satisfy formal correctness, privacy, security and efficiency requirements, in a large class of cyclic groups; specifically, cyclic groups whose multiplication and inverse operations can be efficiently computed, and which admit an efficiently verifiable protocol to prove that an element is in the group. In the presented protocols, the probability that a cheating server convinces the client of an incorrect computation result can be proved to be exponentially small. Previous best results could only achieve a constant probability. The considered class of cyclic groups includes groups often discussed in cryptography literature, such as

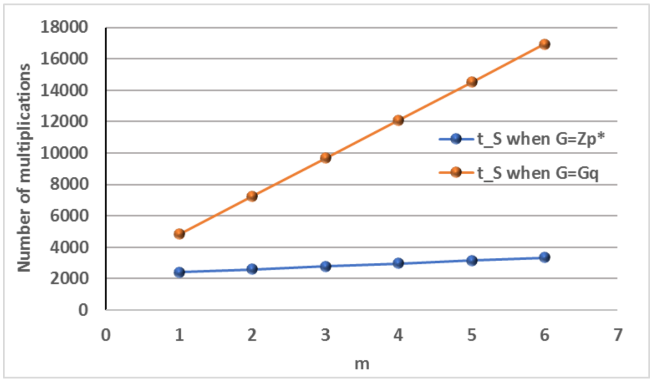

Figure 2: Metric t_S as a function of m

- $\mathbb{Z}_p^*$, for a large prime p, and
- $G_q \subseteq \mathbb{Z}_p^*$, for primes p, q such that $p = 2q + 1$.

Our methods suggest that expensive operations in many cryptographic protocols (specifically, those basing their security on assumptions related to the discrete logarithm problem in frequently discussed groups) can be outsourced to a cloud server, with considerable savings on client computation (e.g., by performing about 70 group multiplications instead of 1 group exponentiation).

Acknowledgement. Many thanks to Xiaowen Zhang for interesting discussions. Research by Delaram Kahrobaei and Vladimir Shpilrain was supported by the ONR (Office of Naval Research) grant N000141210758.

REFERENCES

[1] A. Arbit and Y. Livne and Y. Oren and A. Wool, *Implementing public-key cryptography on passive RFID tags is practical.* In: Int. J. Inf. Sec. 14(1): 85-99, 2015.
[2] P. Barrett, *Implementing the Rivest Shamir and Adleman public key encryption algorithm on a standard digital signal processor*, In: Advances in Cryptology, CRYPTO 1986, LNCS **263**, 311-323, 1986.
[3] L. Batina and J. Guajardo and T. Kerins and N. Mentens and P. Tuyls and I. Verbauwhede, *Public-Key Cryptography for RFID-Tags.* In: Fifth Annual IEEE International Conference on Pervasive Computing and Communications - Workshops (PerCom Workshops 2007), March 2007, White Plains, New York, USA, pp. 217-222, 2007.
[4] Bellare, Mihir and Garay, Juan A and Rabin, Tal, *Fast batch verification for modular exponentiation and digital signatures.* In: Proceedings of Eurocrypt 1998: 236-250, Springer, 1998.
[5] Manuel Blum, Silvio Micali, *How to Generate Cryptographically Strong Sequences of Pseudo-Random Bits.* In SIAM Journal of Computing, 13(4): 850-864 (1984)
[6] V. Boyko and M. Peinado and R. Venkatesan, *Speeding up discrete log and factoring based schemes via precomputations.* In: Advances in Cryptology, EUROCRYPT'98, pp. 221-235, Springer, 1998.
[7] Brickell, Ernest F and Gordon, Daniel M and McCurley, Kevin S and Wilson, David B, *Fast exponentiation with precomputation: algorithms and lower bounds*, In: Proceedings of Eurocrypt '92, pp. 200-207, Springer.
[8] Cavallo, Bren and Di Crescenzo, Giovanni and Kahrobaei, Delaram and Shpilrain, Vladimir, *Efficient and secure delegation of group exponentiation to a single server*, In: International Workshop on Radio Frequency Identification: Security and Privacy Issues: 156-173, Springer, 2015.
[9] Chaum, David and Evertse, Jan-Hendrik and van de Graaf, Jeroen and Peralta, Renér, *Demonstrating possession of a discrete logarithm without revealing it*, In: Proceedings of CRYPTO 1986, pages 200-212, Springer, 1986.
[10] X. Chen and J. Li and J. Ma and Q. Tang and W. Lou, *New algorithms for secure outsourcing of modular exponentiations.* In: Computer Security–ESORICS 2012, pp. 541-556, 2012.

[11] Chevalier, Céline and Laguillaumie, Fabien and Vergnaud, Damien, *Privately outsourcing exponentiation to a single server: cryptanalysis and optimal constructions*, In: European Symposium on Research in Computer Security: 261-278, Springer, 2016.

[12] Ronald Cramer, Victor Shoup, *Design and Analysis of Practical Public-Key Encryption Schemes Secure against Adaptive Chosen Ciphertext Attack*. In SIAM J. Comput. 33(1): 167-226 (2003)

[13] Di Crescenzo, Giovanni and Khodjaeva, Matluba and Kahrobaei, Delaram and Shpilrain, Vladimir, *Computing Multiple Exponentiations in Discrete Log and RSA Groups: From Batch Verification to Batch Delegation*, In: Proc. of 3rd IEEE Workshop on Security and Privacy in the Cloud, IEEE, 2017.

[14] M. Dijk, D. Clarke, B. Gassend, G. Suh, and S. Devadas, *Speeding Up Exponentiation using an Untrusted Computational Resource.* In: Designs, Codes and Cryptography, 39 (2), pp. 253-273, 2006.

[15] Whitfield Diffie, Martin E. Hellman, *New directions in cryptography.* In IEEE Transactions on Information Theory 22(6): 644-654 (1976)

[16] Taher El Gamal, *A public key cryptosystem and a signature scheme based on discrete logarithms.* In IEEE Transactions on Information Theory 31(4): 469-472 (1985)

[17] R. Gennaro, C. Gentry, and B. Parno, *Non-interactive verifiable computing: Outsourcing computation to untrusted workers,* in: Advances in Cryptology, CRYPTO 2010, Lecture Notes Comp. Sc. **6223**, 465–482, 2010.

[18] Craig Gentry: *Fully homomorphic encryption using ideal lattices.* In Proceedings of ACM STOC 2009, pages 169-178

[19] Goldreich, Oded and Micali, Silvio and Wigderson, Avi, *Proofs that yield nothing but their validity or all languages in NP have zero-knowledge proof systems*, In: Journal of the ACM (JACM), 38 (3): 690-728, ACM, 1991.

[20] S. Hohenberger and A. Lysyanskaya, *How to securely outsource cryptographic computations.* In: Proceedings of the Theory of Cryptography Conference 2005, pages 264-282, Springer, 2005.

[21] M. Jakobsson and S. Wetzel, *Secure server-aided signature generation.* In: Proceedings of the Public Key Cryptography conference, pp. 383-401, Springer, 2001.

[22] Lim, Chae Hoon and Lee, Pil Joong *More flexible exponentiation with precomputation*, In: Proceedings of CRYPTO 1994, pages 95-107, Springer, 1994.

[23] X. Ma and J. Li and F. Zhang, *Outsourcing computation of modular exponentiations in cloud computing.* In: Cluster Computing (2013) 16:787-796 (also INCoS 2012).

[24] P. Q. Nguyen and I. E. Shparlinski and J. Stern, *Distribution of modular sums and the security of the server aided exponentiation.* In: Proceedings of Cryptography and Computational Number Theory, pp. 331-342, Springer, 2001.

[25] Y. Wang and Q. Wu and D. Wong and B. Qin and S. Chow and Z. Liu and X. Tao, *Securely outsourcing exponentiations with single untrusted program for cloud storage.* In: Proceedings of Computer Security-ESORICS 2014, pp.326-343, Springer, 2014.

[26] A. C. Yao, *Protocols for secure computations.* In: Proceedings of the 23rd Annual Symposium on Foundations of Computer Science, pp. 160-168, IEEE Computer Society, 1982.

[27] Bernstein, Daniel J, *Pippenger's Exponentiation Algorithm*, Draft, In: Citeseer (2002)

[28] Möller, Bodo, *Improved techniques for fast exponentiation*, In: Proceedings of ICISC, (2587): pp.298-312, Springer (2002)

[29] Cubaleska, Biljana and Rieke, Andreas and Hermann, Thomas, *Improving and extending the Lim/Lee exponentiation algorithm*, In Proceedings of International Workshop on Selected Areas in Cryptography, pp.163-174, Springer (1999)

[30] Ma, Xu and Li, Jin and Zhang, Fangguo, *Outsourcing computation of modular exponentiations in cloud computing*, In: Cluster computing, 16 (4), pp.787-796, Springer (2013)

[31] Cai, Jianxing and Ren, Yanli and Jiang, Tiejin, *Verifiable Outsourcing Computation of Modular Exponentiations with Single Server*, In: International Journal of Network Security, 19 (3), pp.449-457, (2017)

[32] Zhao, Ling and Zhang, Mingwu and Shen, Hua and Zhang, Yudi and Shen, Jian, *Privacy-preserving Outsourcing Schemes of Modular Exponentiations Using Single Untrusted Cloud Server*, In: KSII Transactions on Internet & Information Systems, 11 (2), (2017)