2019 IEEE Conference on Communications and Network Security (CNS): Workshops: SPC: 5th IEEE Workshop on Security and Privacy in the Cloud 2019

# Secure Delegation to a Single Malicious Server: Exponentiation in RSA-type Groups

Giovanni Di Crescenzo*, Matluba Khodjaeva†, Delaram Kahrobaei‡, Vladimir Shpilrain§
*Perspecta Labs. Basking Ridge, NJ, USA. Email: gdicrescenzo@perspectalabs.com
†CUNY John Jay College of Criminal Justice. New York, NY, USA. Email: mkhodjaeva@jjay.cuny.edu
‡University of York. York YO10 5DD, UK. Email: delaram.kahrobaei@york.ac.uk
§CUNY Graduate Center. New York, NY, USA. Email: shpil@groups.sci.ccny.cuny.edu

***Abstract*—In cloud computing application scenarios involving computationally weak clients, the natural need for applied cryptography solutions requires the delegation of the most expensive cryptography algorithms to a computationally stronger cloud server. Group exponentiation is an important operation used in many public-key cryptosystems and, more generally, cryptographic protocols. Solving the problem of delegating group exponentiation in the case of a single, possibly malicious, server, was left open since early papers in the area. Only recently, we have solved this problem for a large class of cyclic groups, including those commonly used in cryptosystems proved secure under the intractability of the discrete logarithm problem.**

**In this paper we solve this problem for an important class of non-cyclic groups, which includes RSA groups when the modulus is the product of two safe primes, a common setting in applications using RSA-based cryptosystems. We show a delegation protocol for fixed-exponent exponentiation in such groups, satisfying natural correctness, security, privacy and efficiency requirements, where security holds with exponentially small probability. In our protocol, with very limited offline computation and server computation, a client can delegate an exponentiation to an exponent of the same length as a group element by only performing two exponentiations to an exponent of much shorter length (i.e., the length of a statistical parameter). We obtain our protocol by a non-trivial adaptation to the RSA group of our previous protocol for cyclic groups.**

***Index Terms*—Secure Outsourcing, Secure Delegation, Modular Exponentiations, RSA, Cryptography, Group Theory**

## I. Introduction

As cloud computing is effectively becoming the reference computation paradigm for large-scale data processing, new solutions for privately and securely sharing the processing of client data are needed. This is especially the case in application scenarios involving computationally weak clients (e.g., RFID networks, Internet-of-Things, LPWA sensor networks, etc.). In such scenarios, the natural need for applied cryptography solutions requires the delegation of the most expensive cryptography algorithms to a computationally stronger cloud server. Research on similar types of problems has been observed in cryptography sub-areas, like server-aided cryptography, commodity-based cryptography, etc. In particular, in server-aided cryptography, researchers have posed and studied the problem of clients delegating cryptographic computations to servers. Ideas related to this area have circulated in the literature already many years ago (see, e.g., [10], 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).

The first formal model for delegation of cryptographic operations was introduced in [24], where the authors especially studied the delegation of modular exponentiation, as this operation is a cornerstone of so many cryptographic protocols, secure under the most typically used assumptions (e.g., the hardness of discrete logarithm, of inverting the RSA function, etc.). In [24], the authors considered secure delegation of exponentiation to 2 servers, assumed to be physically separated, of which at most one can be malicious, and to 1 server, who is assumed to behave honestly on almost all inputs. Since then, the problem of delegating exponentiation to a single, arbitrarily malicious server, has remained unsolved. Here, the challenge is to simultaneously satisfy, in addition to efficiency, correctness and privacy requirements, a security requirement, where it is demanded that no efficient malicious server can convince the client of an incorrect computation result, except with very small probability. This open problem has also been reiterated in [29].

### A. Previous results

As also mentioned in [24], 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 [24] 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 [12], or two non-colluding servers [11], or do not target input privacy [18], or only achieve constant security probability (of detecting a cheating server) [9], [29]. The schemes proposed in [8], [27] do not satisfy our privacy requirement and the scheme proposed in [31] does not satisfy our security requirement. The closest result to what we present in this paper is our previous protocol in [15] that solves the above open problem for the delegation of fixed-base exponentiation in a large class of cyclic groups (including groups commonly used in cryptographic protocols secure under the discrete logarithm assumption). Also of some interest is our delegation protocol in [17] for arbitrary (including non-abelian) groups where an efficient protocol with constant security probability is transformed into one with exponentially small security probability, where the transformation is more efficient than direct parallel

978-1-5386-7117-7/19/$31.00 ©2019 IEEE

repetition (but has non-constant client and offline work for each delegated exponentiation).

In the literature there are also a few general-purpose delegation protocols in different, not applicable, models. In [23] the authors studied the problem of interactive and zero-knowledge proofs with low-complexity verifiers. Later, [19] proposed a protocol for securely delegating arbitrary polynomial-time functions using garbled circuits [30] and fully homomorphic encryption [21]. This protocol delegates functions in settings where the client is powerful enough to run encryption and decryption algorithms of a fully homomorphic encryption scheme, but not enough to homomorphically evaluate a circuit that computes decryption steps in the garbling scheme for the function. Different protocols, not using garbled circuits, were later proposed in [13]. These protocols delegate functions in settings where the client is assumed to be powerful enough to run encryption and decryption algorithms of a fully homomorphic encryption scheme, but not enough to homomorphically evaluate the delegated function.

### B. Our Results

We reconsider the open problem from [24] of delegating exponentiation beyond the case of cyclic groups solved in [15]. A natural candidate is the (non-cyclic) $\mathbb{Z}_n^*$ group used in RSA-based cryptosystems (starting with [28]). A natural question is whether our previous solution in [15] can be used or adapted to use also for non-cyclic group. We answer this in the affirmative, in that we (1) uncover the technical gaps in making our previous protocol for cyclic groups work for non-cyclic groups; (2) modify the protocol to solve these technical gaps for a class of non-cyclic groups, which include the $\mathbb{Z}_n^*$ group used in RSA-based cryptosystems, where $n$ is the product of two, same-length, safe primes (which is the typically recommended setting for $n$).

Our main result is a protocol for the delegation to a single server of fixed-exponent exponentiation in such $\mathbb{Z}_n^*$ groups, which satisfies the desired correctness, privacy, security and efficiency (on client runtime, especially) requirements. This solves the open problem of [24] for (non-cyclic) RSA groups.

In our protocol, security is satisfied with probability exponentially small in a statistical security parameter $\lambda$ (which can be set equal to, for instance, 128). The privacy and security properties do not rely on any additional complexity assumptions, in that they hold even if the adversary corrupting the server is *not limited to run in polynomial time*. The client delegates an exponentiation with an exponent as long as a group element (e.g., of $\sigma = 2048$ bits), while only performing 2 exponentiations 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 $\sigma$-bit exponents. 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.

As a direct application of our result, RSA encryption (with large exponents) can be delegated by reducing the client's computation by about 1 order of magnitude.

## II. Notations and Definitions

In this section we formally define delegation protocols, and their correctness, security, privacy and efficiency requirements, mainly building on the definitional approach from [15], in turn based on those in [19] and [24]. We also introduce group notations and definitions that will be used in the rest of the paper. We start with some basic notations.

### A. 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.

### B. 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 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 communication security techniques. As in all previous work in the area, we consider a model with an offline phase, where computations of the delegated function on random inputs 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.

Let $\sigma$ denote the computational security parameter (i.e., the parameter derived from hardness considerations on the underlying computational problem), and let $\lambda$ denote the statistical security parameter (i.e., a parameter such that events with probability $2^{-\lambda}$ are extremely rare). Both parameters are

expressed in unary notation (i.e., $1^\sigma, 1^\lambda$) and implicitly assumed as inputs to all algorithms. 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 $(C(desc(F), x), S(desc(F)))$, and consisting of the following steps:

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

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 $= F(x)$) and $S$ learns $y_S$ (usually an empty string in this paper). In addition to parameters $1^\sigma, 1^\lambda$, we will often omit $desc(F)$ 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).

### C. 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 1:* Let $\sigma, \lambda$ 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 $\delta_c$-*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 $\delta_c$ 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

### D. 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:* Let $\sigma, \lambda$ 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 $\epsilon_s$-*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 $\epsilon_s$ 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.

### E. 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 3:* Let $\sigma, \lambda$ 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 $\epsilon_p$-*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 $\epsilon_p$ close to 0, where experiment PrivExp is detailed below:

$\text{PrivExp}_{\text{F,A}}(1^\sigma, 1^\lambda)$

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.

### F. 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, $C$ can be run using $t_C(\sigma, \lambda)$ atomic operations, $S$ can be run using $t_S(\sigma, \lambda)$ atomic operations, and $C$ and $S$ exchange a total of at most $mc$ messages, of total length at most $cc$. We also define the *round complexity* of $(C, S)$ as $mc$, the *communication complexity* of $(C, S)$ as $cc$, *C's runtime complexity* as $t_C(\sigma, \lambda)$, *S's runtime complexity* as $t_S(\sigma, \lambda)$ and the *offline runtime complexity* as $t_P(\sigma, \lambda)$.

In our runtime analysis, we only count 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.

### G. Group notations

Let $p_1, p_2$ be primes of the same length. We say that $p_1, p_2$ are *safe primes* if they can be written as $p_1 = 2p_1' + 1$ and $p_2 = 2p_2' + 1$, for some primes $p_1', p_2'$. Let $n = p_1 p_2$, and let $\mathbb{Z}_n^*$ denote the set of integer coprime with $n$. Note that the order $\phi(n)$ of $\mathbb{Z}_n^*$ satisfies $\phi(n) = (p_1 - 1)(p_2 - 1) = 4p_1'p_2'$. We consider the group $(\mathbb{Z}_n^*, \cdot)$, where $\cdot$ denotes multiplication modulo $n$, and a fixed exponent $e$ such that $gcd(e, \phi(n)) = 1$. In this group, with parameter values $n, e$, we define the *fixed-exponent exponentiation function* as

$$\mathsf{feExp}_{n,e} : x \in \mathbb{Z}_n^* \to y \in \mathbb{Z}_n^*, \text{ such that } y = x^e \mod n.$$

Note the notation difference with the *fixed-base exponentiation function* in a cyclic multiplicative group $G$ with generator $g$, defined as

$$\mathsf{fbExp}_g : x \in G \to y \in G, \text{ such that } y = g^x.$$

By Lagrange's theorem, the order of an element in $\mathbb{Z}_n^*$ has to divide $\phi(n)$. However, it turns out that no element in $\mathbb{Z}_n^*$ has order 4 (see, e.g., [16], for a detailed proof). Thus, we have the following (a similar fact was also used in [20]):

*Fact 2.1* For any $x$ in $\mathbb{Z}_n^*$, the order of $x$ is equal to 1 or 2, or is greater than or equal to $\min(p_1', p_2')$.

Let $\sigma$ be the computational security parameter associated with group $\mathbb{Z}_n^*$, and let $\ell$ denote the length of the binary representation of elements in $\mathbb{Z}_n^*$. Typically, in cryptographic applications we generate parameter $\ell$ as about equal to $\sigma$. The textbook algorithm to compute function $\mathsf{feExp}_{n,e}$ is the *square-and-multiply* algorithm, which requires up to $2\ell$ multiplications modulo $n$.

By $desc(\mathsf{feExp}_{n,e})$ we denote a conventional description of the function $\mathsf{feExp}_{n,e}$ that includes an encoding of the function's semantic meaning, and a binary encoding of values $n, e$. By $t_{exp}(\ell)$ we denote a parameter denoting the number of multiplications in $\mathbb{Z}_n^*$ used to compute an exponentiation (in $\mathbb{Z}_n^*$) of a group value to an arbitrary $\ell$-bit exponent. We will use values of $\ell$ from $\{\sigma, \lambda\}$.

## III. Delegation of Exponentiation in an RSA-type Group

In this section we present a delegation protocol for exponentiation in an RSA-type group. First, in Section III-A we formally state our result. Then, we provide an informal discussion of the protocol design in Section III-B and a formal description of the protocol in Section III-C. Finally, we detail the proof of the protocol's correctness, privacy, security and efficiency properties in Section III-D and discuss a protocol generalization in Section III-E.

### A. Formal Statement of Main Result

We formally state our main result as the following

*Theorem 1:* Let $n$ be an integer as defined in Section II, let $(\mathbb{Z}_n^*, \cdot)$ be the associated multiplicative group, where $\cdot$ is multiplication modulo $n$, and let $\sigma$ be its computational security parameter. Also, let $\lambda$ be a statistical security parameter. There exists (constructively) a client-server delegation protocol $(C, S)$ for function $\mathsf{feExp}_{n,e}$, which satisfies

1) $\delta_c$-correctness, for $\delta_c = 1$;
2) $\epsilon_s$-security, for $\epsilon_s = 2^{-\lambda}$;
3) $\epsilon_p$-privacy, for $\epsilon_p = 0$;
4) efficiency with parameters $(t_F, t_S, t_P, t_C, cc, mc)$, for
   - $t_F = t_{exp}(\sigma)$
   - $t_S = 2 \cdot t_{exp}(\sigma)$
   - $t_P = 2 \cdot t_{exp}(\sigma)$ + one inversion in $\mathbb{Z}_n^*$
   - $t_C = 2 \cdot t_{exp}(\lambda) + 5$ multiplications in $\mathbb{Z}_n^*$
   - $cc = 4$ values from $\mathbb{Z}_n^*$
   - $mc = 2$ messages.

In other words, protocol $(C, S)$ for the delegation of exponentiation to a fixed $\sigma$-bit exponent in $\mathbb{Z}_n^*$ satisfies correctness with no error probability, security with error probability $2^{-\lambda}$, privacy with no error probability and requires the following computation resources: 2 exponentiations with $\sigma$-bit exponents from $S$, 2 exponentiations with $\lambda$-bit exponents plus 5 modular multiplications from $C$, 2 exponentiations with $\sigma$-bit exponents and one inversions in the offline phase. Furthermore, the online phase of $(C, S)$ only requires 2 messages and 4 group values to be exchanged.

We remark that with the currently recommended numeric settings of parameters $\sigma = 2048$ and $\lambda = 128$, and realizing for instance a modular exponentiation using the square-and-multiply algorithm, in the protocol from Theorem 1, the upper bound on modular multiplications in a non-delegated computation is 4096. On the other hand, the upper bound

on $C$'s modular multiplications in the protocol $(C, S)$ from Theorem 1 can be, for instance, 517, if we ask for a security probability of $2^{-128}$.

Also remarkable are the online runtime of $S$ and the offline phase, whose work is only twice as much as a non-delegated computation.

Finally, we note that the protocol's message complexity of 2 is clearly minimal in this model.

### B. Informal description of our protocol

The main challenge in coming up with delegation protocols in our model consists of allowing a client to efficiently verifying an exponentiation performed by a single, possibly malicious, server. Since we work in a single-server model, we cannot consider techniques in the multi-server model (as in, e.g., [24]) where the client interacts with a server to check another server's computation. Also, note that the efficiency constraint on the client prevents us to consider general conversion techniques in the cryptography literature to transform a protocol secure against a honest party into one secure against a malicious one (e.g., [22]).

Our starting point is the protocol for efficient, private and secure delegation of fixed-base exponentiation in cyclic groups in [15], also reviewed in Appendix A. There, one main idea consists of a probabilistic verification equation which is verifiable using a much smaller number of modular multiplications (i.e., about $\lambda$, instead of $\sigma$, multiplications). Specifically, in that protocol, $C$ injects an additional random element in the inputs on which $S$ is asked to computed the value of the exponentiation function $F_{exp}$, so to satisfy the following properties: (a) if $S$ returns correct computations of $F_{exp}$, then $C$ can use these random values to correctly compute $y$; (b) if $S$ returns incorrect computations of $F_{exp}$, then $S$ either does not meet some deterministic verification equation or can only meet $C$'s probabilistic verification equation for at most 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 the group, this results in a considerable efficiency gain on $C$'s running time.

In the design of our protocol proving Theorem 1, we first of all attempt to adapt the delegation protocol for fixed-base exponentiation over cyclic groups in [15] to a delegation protocol for fixed-exponent exponentiation over an RSA group. The conversion from a delegation protocol for fixed-base exponentiation to one for fixed-exponent exponentiation is somewhat standard, and is performed by notation changes. However, after that, we note that the analysis of the probabilistic verification test in [15] makes two assumptions that are not true or not known to be true in our group: (1) there exists an efficient protocol for the client to verify that a value sent by the server actually belongs to the group; and (2) the group is cyclic. Specifically, (1) is unknown to be true (note that, for instance, verifying whether a value is coprime with $n$ requires computing a GCD which is not significantly more efficient than an exponentiation in $\mathbb{Z}_n^*$); and (2) is not true since $\mathbb{Z}_n^*$ is not cyclic when $n$ is the product of two primes.

To deal with (1), in our protocol the client can efficiently test whether values sent by the server are in $\mathbb{Z}_n$, and then we show that if the server sends values in $\mathbb{Z}_n \backslash \mathbb{Z}_n^*$ the probabilistic test either cannot be satisfied or can only be satisfied for at most one value of $C$'s random element.

To deal with (2), we further study the analysis in [15] of the probabilistic test and observe that it can be adapted to hold with respect to values in a non-cyclic subgroup of $\mathbb{Z}_n^*$, as long as elements of this subgroup have high (i.e., greater than $2^{\lambda}$) order. Although the textbook choice of $n$ as the product of two large, same-length, primes may not always satisfy this condition, we observe that a very common choice of $n$ in cryptographic applications satisfies a very similar condition. Specifically, by choosing $n$ as the product of two safe primes, the group $\mathbb{Z}_n^*$ only contains elements of either order (much) higher than $2^{\lambda}$, or of rather low order. Furthermore, the latter case can be efficiently tested by $C$ using a single multiplication (i.e., testing whether the group element is a non-trivial root of 1). In the case $C$'s input is a value of low order, to preserve the protocol's correctness, $C$ efficiently calculates the function $\mathsf{feExp}_{n,e}$ by himself and halts the protocol.

### C. Formal description of our protocol

*Input to C and S:*

1) parameters $1^{\sigma}$ and $1^{\lambda}$
2) $desc(\mathsf{feExp}_{n,e})$, including $n, e$

*Private input to C:* $x \in \mathbb{Z}_n^*$

*Offline phase instructions:*

1) Randomly choose $u_i \in \mathbb{Z}_n^*$, for $i = 0, 1$
   Set $v_0 = u_0^{-e} \mod n$, $v_1 = u_1^{e} \mod n$ 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 $z_0 := x \cdot u_0 \mod n$, $z_1 := x^b \cdot u_1 \mod n$
   $C$ sends $z_0, z_1$ to $S$
2) For $i = 0, 1$
   $S$ computes $w_i := z_i^e \mod n$
   $S$ sends $w_i$ to $C$
3) if $x^2 = 1 \mod n$
   $C$ returns $y = x$ and the protocol halts
   $C$ computes $y := w_0 \cdot v_0 \mod n$
   if $w_i \notin \mathbb{Z}_n$ for some $i \in \{0, 1\}$ then
   $C$ returns $\perp$ and the protocol halts
   If $w_1 \neq y^b \cdot v_1 \mod n$ (the 'probabilistic test') then
   $C$ returns $\perp$ and the protocol halts
   $C$ returns $y$

### D. Proof of properties of our protocol

The *efficiency* properties are verified by protocol inspection:

- With respect to *round complexity,* the online phase of the protocol only requires one round, consisting of one

message from $C$ to $S$, followed by one message from $S$ to $C$.
- With respect to *communication complexity*, the online phase of the protocol requires the transfer of 2 elements in $\mathbb{Z}_n^*$ from $S$ to $C$ and 2 elements in $\mathbb{Z}_n^*$ from $C$ to $S$.
- The *offline runtime complexity* consists of 2 fixed-exponent exponentiations with a random base. *S's runtime complexity* consists of 2 fixed-exponent exponentiations. *C's runtime complexity* consists of 5 multiplications and 2 exponentiations to a random exponent $\leq 2^\lambda$. In the typical setting $\lambda = 128$, and using the the square-and-multiply algorithm for multiplication, $C$ only performs at most 517 (or an average of 389) group multiplications in $\mathbb{Z}_n^*$.

*The correctness* property is demonstrated by observing that if $C$ and $S$ follow the protocol then the following holds:
- if $x^2 = 1$ then $C$ returns $y = x$. Note that
  - if the power of $x$ is an even number (say, $2a$) in $\mathbb{Z}_n$, then $x^{2a} = (x^2)^a = 1^a = 1 \mod n$
  - if the power of $x$ is an odd number (say, $2a+1$) in $\mathbb{Z}_n$, then $x^{2a+1} = x^{2a} \cdot x = x \mod n$

  Since $\gcd(e, \phi(n)) = 1$, we know that $e$ is an odd positive integer which implies that $y = \mathsf{feExp}_{n,e}(x) = x^e = x \mod n$.
- Otherwise, $C$ performs 2 more verifications and outputs $y$ such that $y = x^e$. Both verifications always pass as follows:
  - the test checking membership of $w_0, w_1$ to $\mathbb{Z}_n$ is always passed since $w_i$ is computed by $S$ as $z_i^e \mod n$, for some $z_i \in \mathbb{Z}_n^*$ and all $i = 0, 1$ and thus $w_i \in \mathbb{Z}_n^* \subseteq \mathbb{Z}_n$ for all $i = 0, 1$
  - *the probabilistic test* is always passed since $w_1 = z_1^e = (x^b \cdot u_1)^e = (x^e)^b \cdot u_1^e = y^b \cdot v_1 \mod n$

  Thus, $C$ never returns $\perp$, but does return $y$. To see that $y$ is the correct output, note that

$$y = w_0 \cdot v_0 = z_0^e \cdot u_0^{-e} = (x \cdot u_0)^e \cdot u_0^{-e} = x^e \mod n.$$

The *privacy* property of the protocol against a malicious $S$ follows by observing that $C$'s message to $S$ does not leak any information about $x$. This message is a pair $(z_0, z_1)$ where $z_0 = (x \cdot u_0) \mod n$, $z_1 = x^b \cdot u_1 \mod n$. Note that as $u_0$ and $u_1$ are uniformly and independently distributed in $\mathbb{Z}_n^*$, so are $z_0$ and $z_1$. For the same reason, it satisfies the following two properties:

1) for any $x$, $z_0$ and $z_1$ are uniformly and independently distributed in $\mathbb{Z}_n^*$;
2) the distribution of $b$, conditioned on $x$, $z_0$ and $z_1$, is uniform over $\{1, \ldots, 2^\lambda\}$.

We will use both facts in the proof of the security property.

To prove the *security* property against a malicious $S$ we need to compute an upper bound $\epsilon_s$ on the security probability that $S$ convinces $C$ to output a $y$ such that $y \neq \mathsf{feExp}_{n,e}(x)$. If $x^2 = 1 \mod n$, $C$ does not use any information from $S$; in other words, $C$ calculates $\mathsf{feExp}_{n,e}(x) = x^e$ without using $w_0$, $w_1$ sent by $S$. Thus, $\epsilon_s = 0$ in this case. Now, assume that $x^2 \neq 1 \mod n$. We obtain that $\epsilon_s = 2^{-\lambda} + \epsilon$, as consequence of the following 4 claims:

1) for all $i = 0, 1$ if $w_i \notin \mathbb{Z}_n$ then $C$ always outputs $\perp$
2) If exactly one between $w_0$ and $w_1$ is in $\mathbb{Z}_n \backslash \mathbb{Z}_n^*$, then $C$ outputs $\perp$ after running the probabilistic test;
3) if $w_0, w_1 \in \mathbb{Z}_n \backslash \mathbb{Z}_n^*$ or $w_0, w_1 \in \mathbb{Z}_n^*$, then $C$ does not output $\perp$ after running the probabilistic step with probability at most $2^{-\lambda}$;
4) if all of $C$'s verifications in the protocol are satisfied, then, except with probability $2^{-\lambda}$ it holds that $y = x^e \mod n$.

Claim 1 directly follows by inspection of $C$'s instructions.

To prove Claim 2, we consider two cases, depending on which among $w_0$ and $w_1$ is in $\mathbb{Z}_n \backslash \mathbb{Z}_n^*$. First assume $w_1 \in \mathbb{Z}_n \backslash \mathbb{Z}_n^*$ and $w_0 \in \mathbb{Z}_n^*$. In this case both $y$ and $y^b \cdot v_1 \mod n$ are in $\mathbb{Z}_n^*$ and the probabilistic test cannot be satisfied as it tests equality between two values in different subsets of $\mathbb{Z}_n$. Now, assume $w_0 \in \mathbb{Z}_n \backslash \mathbb{Z}_n^*$ and $w_1 \in \mathbb{Z}_n^*$. In this case, we can write, without loss of generality, $w_0 = p_1 w_0'$, for some integer $w_0 < p_2$, and, by setting $t$ as the quotient of the division $y^b/n$ over the integers, we can further write the value $y^b$, for any integer $b$, as

$$\begin{aligned} y^b &= (w_0 v_0)^b = (p_1 w_0' v_0)^b = p_1^b (w_0' v_0)^b - tn \mod n \\ &= p_1 (p_1^{b-1} (w_0' v_0)^b - t p_2) \mod n, \end{aligned}$$

where the quantity $(p_1^{b-1}(w_0 v_0)^b - tq)$ is strictly less than $p_2$. This implies the following fact (also useful later):

*Fact 3.D.1.* For any $y \in \mathbb{Z}_n \setminus \mathbb{Z}_n^*$ and integer $b > 0$, it holds that $y^b \mod n \in \mathbb{Z}_n \setminus \mathbb{Z}_n^*$

Then we have that both $y$ and $y^b \mod n$ belong to $\mathbb{Z}_n \setminus \mathbb{Z}_n^*$ and thus the probabilistic test is not satisfied as it tests equality between two values in different subsets of $\mathbb{Z}_n$.

To prove Claim 3, 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 $y \neq \mathsf{feExp}_{n,e}(x)$'
- $e_\perp$, defined as '$C$ outputs $\perp$'

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

*Fact 3.D.2.* 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 $w_1 = (w_0 \cdot v_0)^b \cdot v_1 \mod n$'
- $e_{>1,b}$, defined as '$\exists$ more than one $b$ such that $S$'s message $(w_0, w_1)$ satisfies $w_1 = (w_0 \cdot v_0)^b \cdot v_1 \mod n$'.

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 $\{1, \ldots, 2^\lambda\}$ 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 $2^\lambda$ of values of $b$ that are still equally likely even when conditioning over message $(z_0, z_1)$. We obtain the following

*Fact 3.D.3.* $\text{Prob}\,[\,\neg\, e_\perp | e_{1,b}\,] \leq 1/2^\lambda$

We now show that if $S$ is malicious then it cannot produce in step 2 of the protocol values $w_0, w_1 \in \mathbb{Z}_n \setminus \mathbb{Z}_n^*$ or values $w_0, w_1 \in \mathbb{Z}_n^*$ satisfying both of $C$'s verifications for two distinct values $b_1, b_2 \in \{1, \ldots, 2^\lambda\}$. To prove this statement we assume, towards contradiction, that there exist two such values $b_1$ and $b_2 \in \{1, 2, \ldots, 2^\lambda\}$, and, without loss of generality, assume that $b_1 > b_2$. Then $b_1 - b_2 \in \{1, \ldots, 2^\lambda - 1\}$ and we have that

$$w_1 = y^{b_1} \cdot v_1 \mod n \text{ and } w_1 = y^{b_2} \cdot v_1 \mod n$$
$$y^{b_1} \cdot v_1 = y^{b_2} \cdot v_1 \mod n$$
$$y^{b_1 - b_2} = 1 \mod n.$$

We divide the rest of the proof into two cases, depending on which set both of $w_0, w_1$ belong to.

*Case (a):* $w_0, w_1 \in \mathbb{Z}_n \setminus \mathbb{Z}_n^*$. In this case, recall that $y = w_0 \cdot v_0 \mod n$, and since $v_0 \in \mathbb{Z}_n^*$, we have that $y \in \mathbb{Z}_n \setminus \mathbb{Z}_n^*$. Moreover, since $b_1 - b_2 > 0$, by applying again Fact 3.D.1, we obtain that $y^{b_1 - b_2}$ is also in $\mathbb{Z}_n \setminus \mathbb{Z}_n^*$, and thus cannot ever be equal to 1, which implies that the equality $y^{b_1 - b_2} = 1 \mod n$ cannot hold, which is the desired contradiction.

*Case (b):* $w_0, w_1 \in \mathbb{Z}_n^*$. In this case, since $y = w_0 \cdot v_0 \mod n$ and $v_0 \in \mathbb{Z}_n^*$, we have that $y \in \mathbb{Z}_n^*$. Recall that by Fact 2.1 the elements of $\mathbb{Z}_n^*$ can only have order equal to 1 or 2, or greater than or equal to $\min(p_1', p_2')$. Since $C$ checks that $y^2 \neq 1$, then $y$ cannot have order 1 or 2, and thus must have order at least $\min(p_1', p_2')$. Since by definition of $b_1, b_2$ we know that $0 < b_1 - b_2 < \min\{p_1', p_2'\}$, we have that the equality $y^{b_1 - b_2} = 1 \mod n$ cannot hold, which is the desired contradiction.

We obtain the following fact.

*Fact 3.D.4.* $\text{Prob}\,[\,e_{>1,b}\,] = 0$

The rest of the proof of Claim 3 consists of computing an upper bound $\epsilon_s$ on the probability of event $e_{y,\neq}$. We have the following

$$\begin{aligned}
\Pr(e_{y,\neq}) &\leq \Pr(\neg\, e_\perp) \\
&= \Pr(e_{1,b}) \cdot \Pr(\neg\, e_\perp | e_{1,b}) \\
&\quad + \Pr(e_{>1,b}) \cdot \Pr((\neg\, e_\perp | e_{>1,b}) \\
&= \Pr(e_{1,b}) \cdot \Pr(\neg\, e_\perp | e_{1,b}) \\
&\leq \Pr(e_{1,b}) \cdot \frac{1}{2^\lambda} \\
&\leq \frac{1}{2^\lambda},
\end{aligned}$$

where the first inequality follows from Fact 3.D.2, 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.D.4, and the second inequality follows from Fact 3.D.3. We then obtain that $\epsilon_s = 2^{-\lambda}$, and the claim 3 follows.

Claim 4 follows by protocol inspection and combining claims 1-3, which concludes the proof of the security property for $(C, S)$. ∎

### *E. A generalization of our protocol*

We considered the natural question of which groups does our protocol $(C, S)$ in Section III-C generalize to. We believe our protocol generalizes to groups $(\mathbb{Z}_n^*, \cdot)$, where all elements of $\mathbb{Z}_n^*$ satisfy one of these two properties:

1) have order larger than $2^\lambda$, for a statistical security parameter $\lambda$; and
2) are efficiently detectable to have much smaller order (say, a small constant independent of $\lambda, \sigma$).

Note that for these generalized $\mathbb{Z}_n^*$ groups, the first test in step 3 of the protocol (currently checking whether $x^2 = 1 \mod n$) is updated into a test to detect whether $C$'s input has low order (which exists from above property 2), and the analysis related to the probabilistic test in step 3 is still satisfied (thanks to the above property 1). Examples of integers $n$ for such groups include products of a constant number of safe primes of the same length, products of two safe primes of different lengths $> \lambda$, and products of two same-length primes $p_1, p_2$ such that each factor of product $(p_1 - 1)(p_2 - 1)$ is either a small power of 2 or $> 2^\lambda$.

## IV. Performance Analysis

In this section we describe parametric (as a function of parameters $\sigma, \lambda$) and numeric performance evaluations of our protocol in Section III. We also compare our protocol with the non-delegated computation of the same function.

So far we have expressed the performance of our protocol in terms of group multiplications and two parameter functions: $t_{exp}(\ell)$, the number of multiplications in $\mathbb{Z}_n^*$ to compute one exponentiation to an arbitrary $\ell$-bit exponent; and $t_{inv}(\ell)$, the number of multiplications in $\mathbb{Z}_n^*$ to compute one inversion to an arbitrary $\ell$-bit value. A more concrete evaluation of the performance of our protocol requires a (possibly optimized) instantiation of these two functions. As there can be different algorithms for the computation of exponentiation or inversion for arbitrary $\ell$-bit value, we optimize for the following two representative settings for $t_{exp}$:

1) *Basic setting for exponentiation runtime:* using the text-book square-and-multiply algorithm to evaluate group exponentiation we can set
   - $t_{exp}(\ell) = 2\ell$
2) *Improved setting for exponentiation runtime*: using the closed-form estimate for the number of multiplications in Brauer's 1939 exponentiation algorithm, as described in [5], we can set

TABLE I
PERFORMANCE METRICS FOR PROTOCOL $(C, S)$, IN TERMS OF NUMBER OF MULTIPLICATIONS, FOR GENERAL $\sigma, \lambda$

| Metric | Basic or Improved | feExp (no delegation) | feExp (with delegation) |
|---|---|---|---|
| $t_F$ | B | $2\sigma$ | $2\sigma$ |
| | I | $\sigma(1+\frac{1}{\log\sigma})$ | $\sigma(1+\frac{1}{\log\sigma})$ |
| $t_P$ | B | 0 | $4\sigma+3$ |
| | I | 0 | $2\sigma(1+\frac{1}{\log\sigma})+3$ |
| $t_C$ | B | $2\sigma$ | $4\lambda+5$ |
| | I | $\sigma(1+\frac{1}{\log\sigma})$ | $2\lambda(1+\frac{1}{\log\lambda})+5$ |
| $t_S$ | B | 0 | $4\sigma$ |
| | I | 0 | $2\sigma(1+\frac{1}{\log\sigma})$ |
| $\epsilon_p$ | B & I | 0 | 0 |
| $\epsilon_s$ | B & I | 0 | $2^{-\lambda}+\epsilon$ |

TABLE II
PERFORMANCE METRICS FOR PROTOCOL $(C, S)$, IN TERMS OF NUMBER OF MULTIPLICATIONS, FOR $\sigma = 2048$ AND $\lambda = 128$

| Metric | Basic or Improved | feExp (no delegation) | feExp (with delegation) |
|---|---|---|---|
| $t_F$ | B | 4096 | 4096 |
| | I | 2235 | 2235 |
| $t_P$ | B | 0 | 4099 |
| | I | 0 | 4472 |
| $t_C$ | B | 4096 | 517 |
| | I | 2235 | 152 |
| $t_S$ | B | 0 | 4096 |
| | I | 0 | 4469 |

- $t_{exp}(\ell) = \ell(1+\frac{1}{\log\ell})$.

We also refer the reader to [6], [7], [14], [26] for other algorithms claiming runtime improvements, although note that these papers do not provide additional closed-form evaluations.

In order to evaluate group inversion of $\ell$-bits (i.e. $t_{inv}(\ell)$), we can also use the protocol for delegation of inversion described in [9], where the computationally weak device only needs to calculate 3 $\ell$-bits multiplications.

Tables I and II compare the performance of our delegation protocol in Section III with a non-delegated computation of the client under both basic (B) and improved (I) settings for functions $t_{exp}, t_{inv}$ where in Table I, we give the formula in general case for parameters $\sigma$ and $\lambda$ and in Table II we plug the values for $\sigma = 2048$ and $\lambda = 128$. The Table I and II reports expressions (as a function of $\sigma, \lambda$, for efficiency metrics $t_F$ (the number of multiplications to compute function feExp), $t_P$ (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), $\epsilon_p$ (the probability parameter in the privacy definition), and $\epsilon_s$ (the probability parameter in the security definition).

The main takeaway from the tables is that by comparing non delegated protocols with our protocol, we see that our protocol in Section III reduces $t_C$ by a multiplicative factor of about $\sigma/\lambda$ with respect to non-delegated computation when using both basic and improved settings

## V. CONCLUSIONS

We considered the problem of a computationally weaker client delegating group exponentiation to a single, possibly malicious, server, originally left open in [24], in some class of RSA-type groups. We solved this problem by showing a protocol that provably satisfy formal correctness, privacy, security and efficiency requirements, in the class of groups $\mathbb{Z}_n^*$, where $n$ is the product of two safe primes. In the presented protocol, the probability that a cheating server convinces the client of an incorrect computation result can be proved to be exponentially small.

Our techniques imply that expensive operations in RSA-based cryptographic protocols (specifically, encryption with a large exponent) can be delegated to a cloud server, with considerable savings on client computation.

## 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: Proc. of CRYPTO 1986, Springer LNCS 263, pp. 311–323, 1986.

[3] L. Batina, J. Guajardo, T. Kerins, N. Mentens, P. Tuyls and I. Verbauwhede, *Public-Key Cryptography for RFID-Tags.* In: Proc. of 5th IEEE Int. Conf. on Pervasive Computing and Communications - Workshops (PerCom Workshops 2007), pp. 217-222, 2007.

[4] M. Bellare, J. Garay, and T. Rabin, *Fast batch verification for modular exponentiation and digital signatures.* In: Proc. of Eurocrypt 1998, Springer LNCS 1403, pp. 236–250, 1998.

[5] D. J. Bernstein, *Pippenger's Exponentiation Algorithm*. In: Citeseer (2002)

[6] B. Möller, *Improved techniques for fast exponentiation*. In: Proc. of ICISC 2002, Springer LNCS 2587, pp.298-312, 2002

[7] E. Brickell, D. Gordon, K. McCurley, and D. Wilson, *Fast Exponentiation with Precomputation.* In: Proc. of Eurocrypt 1992, Springer LNCS 658, pp. 200–207, 1992. See also 1995 preprint.

[8] J. Cai, Y. Ren, and T. Jiang, *Verifiable Outsourcing Computation of Modular Exponentiations with Single Server*, In: Int. Journal of Network Security, vol. 19 (3), pp. 449-457, 2017

[9] B. Cavallo, G. Di Crescenzo, D. Kahrobaei, and V. Shpilrain, *Efficient and Secure Delegation of Group Exponentiation to a Single Server*, In: Proc. of RFIDSec 2015, Springer LNCS 9440, pp. 156–173, 2015.

[10] D. Chaum, J. Evertse, J. van de Graaf, and R. Peralta, *Demonstrating Possession of a Discrete Logarithm without Revealing it*, In: Proc. of CRYPTO 1985, Springer LNCS 263, pp. 200–212, 1986.

[11] X. Chen and J. Li and J. Ma and Q. Tang and W. Lou, *New algorithms for secure outsourcing of modular exponentiations.* In: Proc. of ESORICS 2012, Springer LNCS 7459, pp. 541–556, 2012.

[12] C. Chevalier, F. Laguillaumie, and D. Vergnaud, *Privately outsourcing exponentiation to a single server: cryptanalysis and optimal constructions*, In: Proc. of ESORICS 2016, Springer LNCS 9878, pp. 261–278, 2016.

[13] K. Chung, Y. Kalai and S. Vadhan, *Improved Delegation of Computation using Fully Homomorphic Encryption*. In Proc. of CRYPTO 2010, Springer LNCS 6223, pp. 483–501, 2010.

[14] B. Cubaleska, A. Rieke, and T. Hermann, *Improving and Extending the Lim/Lee Exponentiation Algorithm*. In: Proc. of SAC 1999, Springer LNCS 1758, pp. 163-174, 1999.

[15] G. Di Crescenzo, M. Khodjaeva, D. Kahrobaei, and V. Shpilrain, *Practical and Secure Outsourcing of Discrete Log Group Exponentiation to a Single Malicious Server*. In: Proc. of 9th ACM Cloud Computing Security Workshop (CCSW 2017), CCS 2017: 17-28.

[16] G. Di Crescenzo, M. Khodjaeva, D. Kahrobaei, and V. Shpilrain, *Computing multiple exponentiations in discrete log and RSA groups: From batch verification to batch delegation*. In: Proc. of 3rd IEEE Security and Privacy in the Cloud Workshop (SPC 2017), CNS 2017: pp. 531–539.

[17] G. Di Crescenzo, D. Kahrobaei, M. Khodjaeva, and V. Shpilrain, *Efficient and Secure Delegation to a Single Malicious Server: Exponentiation over Non-abelian Groups*. In: Proc. of ICMS 2018, Springer LNCS 10931, pp. 137-146, 2018.

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

[19] R. Gennaro, C. Gentry, and B. Parno, *Non-interactive verifiable computing: Outsourcing computation to untrusted workers.* In: Proc. of CRYPTO 2010, Springer, LNCS 6223, pp. 465–482, 2010.

[20] R. Gennaro, H. Krawczyk, and T. Rabin, *RSA-based Undeniable Signatures.* In: Proc. of CRYPTO 1997, Springer LNCS 1294, pp. 132–149, 1997.

[21] C. Gentry, *Fully Homomorphic Encryption using Ideal Lattices*. In Proc. of ACM STOC 2009, pp. 169-178, 2009.

[22] O. Goldreich, S. Micali, and A. Wigderson, *Proofs that yield nothing but their validity or all languages in NP have zero-knowledge proof systems*, In: Journal of the ACM, vol. 38 (3), pp. 690–728, 1991.

[23] S. Goldwasser, Y. Tauman Kalai, G. N. Rothblum, *Delegating Computation: Interactive Proofs for Muggles*. In: Journal of the ACM, vol. 62 (4), pp. 27:1–27:64, 2015

[24] S. Hohenberger and A. Lysyanskaya, *How to securely outsource cryptographic computations.* In: Proc. of TCC 2005, Springer LNCS 3378, pp. 264-282, 2005.

[25] M. Jakobsson and S. Wetzel, *Secure server-aided signature generation.* In: Public Key Cryptography, pp. 383–401, Springer LNCS 1992, 2001.

[26] C. H. Lim and P. J. Lee, *More flexible exponentiation with precomputation*, In: Proc. of CRYPTO 1994, pp. 95-107, Springer LNCS 839, 1994.

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

[28] Ronald L. Rivest, Adi Shamir, Leonard M. Adleman: *A Method for Obtaining Digital Signatures and Public-Key Cryptosystems*. In: Commun. ACM, vol. 21 (2), pp. 120–126, 1978.

[29] 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: Proc. of ESORICS 2014, Springer LNCS 8712, pp. 326–343, 2014.

[30] A. C. Yao, *Protocols for secure computations.* In: Proc. of 23rd IEEE FOCS Symposium, pp. 160–168, 1982.

[31] L. Zhao, M. Zhang, H. Shen, Y. Zhang, and J. Shen, *Privacy-preserving Outsourcing Schemes of Modular Exponentiations Using Single Untrusted Cloud Server*, In: KSII Transactions on Internet & Information Systems, vol. 11 (2), 2017.


## APPENDIX

### A. The delegation protocol from [15]

An *efficiently verifiable membership protocol* for group $G$ is 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 $t_{exp}$.

In [15] it is shown that $(\mathbb{Z}_p^*, \cdot \mod p)$, for a large prime $p$, and $(G_q, \cdot \mod p)$, for large primes $p, q$ such that $p = 2q+1$, where $G_q$ is the $q$-order subgroup of $\mathbb{Z}_p^*$, are groups used in cryptography with an efficiently verifiable membership protocol.

Let $(G, *)$ be a cyclic group with efficient operation, and let (mProve, mVerify) denote its efficiently verifiable membership protocol. The delegation protocol for the fixed-base exponentiation function $\mathsf{fbExp}_{g,q}$ in cyclic group $G$ with generator $g$ and order $q$ was formally defined in [15] as follows.

*Input to S:* $1^\sigma, 1^\lambda$, $desc(\mathsf{fbExp}_{g,q})$

*Input to C:* $1^\sigma, 1^\lambda$, $desc(\mathsf{fbExp}_{g,q})$, $x \in \mathbb{Z}_q$

*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: $y$