Efficient and Secure Delegation of Exponentiation in General Groups to a Single Malicious Server

Giovanni Di Crescenzo, Matluba Khodjaeva, Delaram Kahrobaei and Vladimir Shpilrain

Abstract. Group exponentiation is an important and relatively expensive 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 delegated from a computationally weaker client to a computationally stronger server. Solving this problem in the case of a single, possibly malicious, server, has remained open since the introduction of a formal model. In previous work we have proposed practical and secure solutions applicable to two classes of specific groups, related to well-known cryptosystems. In this paper, we investigate this problem in a general class of multiplicative groups, possibly going beyond groups currently subject to quantum cryptanalysis attacks. Our main results are efficient delegation protocols for exponentiation in these general groups. The main technique in our results is a reduction of the protocol's security probability (i.e., the probability that a malicious server convinces a client of an incorrect exponentiation output) that is more efficient than by standard parallel repetition. The resulting protocols satisfy natural requirements such as correctness, security, privacy and efficiency, even if the adversary uses the full power of quantum computers. In particular, in our protocols the client performs a number of online group multiplications smaller by 1 to 2 orders of magnitude than in a non-delegated computation.

Mathematics Subject Classification (2010). Primary 99Z99; Secondary 00A00.

Keywords. Secure Outsourcing, Secure Delegating, Group Exponentiation, Cryptography, Group Theory.

1. Introduction

In emerging applications related to Cloud Computing and the Internet of Things, including RFID networks, interest is growing on deploying cryptography solutions onto computationally weaker devices. To achieve that goal, it has been advocated that the most expensive cryptographic operations are delegated from a computationally weaker client to a computationally stronger server. Group exponentiation is an important operation and among the most expensive ones used in many public-key cryptosystems and, more generally, cryptographic protocols. Many studies have already been performed towards various types of delegation of group exponentiation, but almost exclusively in the case of abelian groups; specifically, groups related to discrete logarithm or factoring problems (see, e.g., [30, 24, 21, 19] and references therein).

As progress is being made towards building a large-scale quantum computer [39], much attention is being devoted in the cryptography community to early quantum computer algorithms such as Shor's [37], capable of solving in quantum polynomial time both the discrete logarithm and the factoring problem. More specifically, the problem at the heart of Shor's algorithms, also known as the hidden subgroup problem, can be solved in quantum polynomial time over any finite abelian group, but currently seems much harder over non-abelian groups [31]. Therefore, the study of cryptographic solutions over non-abelian, or just general, groups is an appealing research direction within quantum-resistant cryptography (see, e.g., [28, 2, 29] and references therein).

In this paper we consider the problem of a client delegating to a server the exponentiation function over a large class of general multiplicative groups, not limited to abelian groups and thus going beyond groups currently subject to quantum cryptanalysis attacks. We target solutions that satisfy natural requirements of correctness (i.e., if client and server follow the protocol, then at the end of the protocol execution, the client's output is the desired exponentiation), security (i.e., if the client follows the protocol, no malicious adversary corrupting the server can convince the client of an incorrect exponentiation, except with small probability), privacy (i.e., if the client follows the protocol, no malicious adversary corrupting the server can obtain some information about the client's input exponent), and efficiency (most notably, the client's runtime in the online phase is smaller than in a non-delegated computation of the exponentiation).

Related Work. Secure delegation of computation can be seen as a successor of previous research areas such as computational program checking and testing (see, e.g., [8, 9, 1]). Early research in secure delegation include [4], which studied delegation of scientific computations and [30], which presented the first formal security definition of secure delegation of computation. In particular, this latter paper considered the delegation of exponentiation over a specific cyclic group in the presence of two servers of which one was untrusted and of a single server almost always returning a correct computation. Several papers have been published in this area since then. We can classify all secure (single-server) delegation protocols we are aware of in 3 main classes, depending on whether they delegate (a) exponentiation in a specific group; (b) other specific operations (e.g., linear algebra operations, group inverses, elliptic curve pairings); and (c) an arbitrary polynomial-time computable function.

With respect to (a), protocols were proposed for a single exponentiation in specific groups related to discrete logarithm or factoring problems (see, e.g., [30, 13, 19, 20] and references therein). These protocols delegate exponentiation in settings where the client is assumed to be powerful enough to run a not-too-large number of group multiplications, but not powerful enough to evaluate the delegated exponentiation function (unless in an offline phase, before the exponentiation input is known). There are also many protocols in the literature for delegating a single exponentiation, not targeting or achieving all of our requirements (see, e.g., [21, 33, 38, 14]). Almost all of these solutions can improve the client efficiency also in the offline phase, under a pseudo-random powers generation assumption, in turn based on the hidden-subset-sum hardness assumption [10, 35], or a the (stronger) subset sum hardness assumption. We note that these assumptions need to be reevaluated in light of more recent results. In [18] we proposed batch delegation of exponentiation for specific discrete logarithm and RSA groups from batch verification, leveraging the small exponent test introduced and studied by [6] over prime-order groups.

With respect to (b), a number of protocols for delegating linear algebra operations and/or scientific computation were proposed (see, e.g., [34, 4, 7, 5, 23]). These protocols delegate various linear algebra operations in settings where the client is assumed to be powerful enough to run some other linear algebra operations of lower time complexity, but not powerful enough to evaluate the delegated linear algebra function. Efficient and secure delegation of group inverses for a general group, from a client powerful enough to run group multiplication, was presented in [12]. The delegation of the computation of elliptic curve pairings also has received much attention (see, e.g., [26, 15, 11]). In these protocols bilinear pairings are delegated by a client that only performs group exponentiations and/or multiplications, but in all of these protocols the delegation has been experimentally evaluated to be more costly or only slightly less costly than computing a pairing without delegation.

With respect to (c), in [24] the authors proposed a protocol using garbled circuits (see [40]) and fully homomorphic encryption [25]. 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 powerful 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 [16]. 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.

Our Contributions. We show a number of interactive protocols allowing a client to securely delegate exponentiation in a general class of groups to a single, possibly malicious, server. Our protocols mainly target the delegation of function $F_{G,exp,k}(x) = x^k$ (i.e., fixed-exponent, variable-base exponentiation over multiplicative group G), but we also reformulate then so to delegate function $F_{G,exp,g}(x) = g^x$ (i.e., variable-exponent, fixed-base exponentiation).

Our first protocol, in Section 3.1, consists of a direct parallel repetition of (a slightly simplified version of) a protocol from [12] that achieves security probability $1/2$. Our main result, in Section 3.2, is a class of protocols where the security probability is reduced more efficiently than by direct parallel repetition. Their privacy and security properties are satisfied even if the adversary corrupting the server is not limited to run in (classical or quantum) polynomial time, and they achieve an efficiency tradeoff, in that they improve the client's runtime during the online protocol phase, while increasing the server's runtime and requiring offline computations returning data to be stored on the client's device. Our theoretical analysis, only considering group exponentiations and multiplications, and neglecting simpler operations such as equality checks and random element generations, suggests that our first (resp., second) protocol reduces the client's online runtime by 1 (resp., 1 to 2) orders of magnitude with respect to the textbook exponentiation algorithm, while increasing the server runtime and the protocol communication complexity by 2 (resp., 1) orders of magnitude and the offline client runtime between a constant and 1 order of magnitude.

In Section 3.3 we adapt these protocols so to delegate variable-exponent fixed-base exponentiation. The resulting protocols have similar efficiency properties, with a slight improvement on the client's online runtime, which can be about 2 orders of magnitude less than in the textbook exponentiation algorithm, according to our theoretical analysis.

Finally, in Section 3.4, we present our software implementation, in Python 3.6, using commodity computing resources and the gmpy2 package, which confirms that both our protocols improve the client's online runtime with respect to the exponentiation algorithm available in the same package. As in all previous work in the area, we consider a model with an offline phase, where a trusted party can precompute exponentiations, and store them on the client's device to be later used in the online phase, when the client becomes aware of the input to the exponentiation function.

2. Models and Definitions

In this section we formally define delegation protocols, and their correctness, security, privacy and efficiency requirements, building on the definitional approach from [12] (also based on [24], [30]), and describe group notations and protocol preliminaries.

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 those of a server, 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 authenticated or not subject to 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 for instance exponentiations to random exponents can be precomputed and made somehow available to the client. This model has been justified in several ways, all motivated by 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 may not have significant computation constraints, the client can itself perform the precomputed exponentiations and store them on its own device. For simplicity of description, we will consider a generic Offline algorithm keeping in mind that it is run by either a trusted party or a client without significant computation constraints.

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 be a function, and let $desc(F)$ 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 delegated 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. $pp \leftarrow$ 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 $= F(x)$) and S learns y_S (usually an empty string in this paper). S. We will often omit $desc(F), 1^\sigma, 1^\lambda$ for brevity of description. Executions of delegated 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. Informally speaking, the correctness requirement states that if both parties follow the protocol, at the end of the protocol execution, C's output y is, with high probability, equal to the evaluation of function F on C's input x. 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 delegated 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:

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

1. $pp \leftarrow$ 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. Informally speaking, the security requirement states that if C follows the protocol, a malicious adversary corrupting S and even choosing C's input x can only convince C with a small probability to output, at the end of the protocol, some y' different from value $F(x)$ or some failure symbol $\perp$. We will also call this probability as the *security probability*, and denote it as ϵ_s. A desirable value for it will be $2^{-\lambda}$, for some *statistical security parameter* λ, concretely set as, for instance, equal to 128. 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 delegated 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. Informally speaking, the privacy requirement states 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 at the end of the protocol the adversary can only guess which input was used by C with probability $1/2$. 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 delegated 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:

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

Efficiency. We measure the efficiency of a client-server protocol (C, S) for the delegated computation of function F by the *efficiency metrics* (t_F, t_P, t_C, t_S, cc), meaning that F can be computed (without delegation) using t_F atomic operations, C can be run in the offline phase using t_P atomic operations and in the online phase using t_C atomic operations, S can be run using t_S atomic operations, and C and S exchange messages of total length at most cc. In our theoretical 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, random element generations, additions and subtractions over $\mathbb{Z}_n$-type groups). While we naturally try to minimize all these protocol efficiency metrics, our main goal is to design protocols where $t_C << t_F$, even if possibly resulting in t_S being somewhat larger than t_F and cc being somewhat larger than the length of F's input and output. We note that, according to the textbook 'square-and-multiply' algorithm, t_F is, on average, $= 1.5\sigma$ group multiplications, where σ denotes the length of the binary representation of a group element. As a theoretical goal, we target protocols where t_C is much smaller than σ group multiplications.

Group notations. Let ℓ denote the length of the binary representation of a group's elements. We say that a group is *efficient* if its description is short (i.e., has length polynomial in ℓ), its associated operation $*$ and the inverse operation are efficient (i.e., they can be executed in time polynomial in ℓ). The security parameter σ and the group element length ℓ are typically set as the same value. In the rest of the paper we study two types of exponentiation in any efficient group, depending on whether the base or the exponent are the input to the exponentiation function.

Fixed-exponent variable-base exponentiation. Let $(G, *)$ be an efficient group of order q, and let k be an integer known to both parties, and assumed, for simplicity, less than q. Also, let $y = x^k$ denote the *fixed-exponent variable-base exponentiation (in G)* of x to the k-th power; i.e., the value $y \in G$ such that $x * \cdots * x = y$, where the multiplication operation $*$ is applied $k-1$ times. Then we denote by $F_{G,exp,k} : G \to G$ the function that maps every $x \in G$ to the variable-exponent fixed-base exponentiation (in G) of x to the k-th power.

Variable-exponent fixed-base exponentiation. Let $(G, *)$ be an efficient group, and let g be an element with order q, for some large integer q known to the client, and let $y = g^x$ denote the *variable-exponent fixed-base exponentiation (in G)* of g to the x-th power; i.e., the value $y \in G$ such that $g * \cdots * g = y$, where the multiplication operation $*$ is applied $x-1$ times. Also, let $\mathbb{Z}_q = \{0, 1, \ldots, q-1\}$ and $F_{G,exp,g} : \mathbb{Z}_q \to G$ denote the function that maps every $x \in \mathbb{Z}_q$ to the variable-exponent fixed-base exponentiation (in G) of g to the x-th power.

Protocol preliminaries. In all our protocols, inputs commons to client and server include a description of the group, the exponentiation function to be delegated, a computational parameter 1^σ and a security parameter 1^λ. The input to the exponentiation function will be known to the client only. Our main protocol descriptions will be for the fixed-exponent variable-base exponentiation function $F_{G,exp,k}$ and therefore k will also be known to both client and server. When we describe our protocols for the variable-exponent fixed-base exponentiation function $F_{G,exp,g}$, the group element g will also be known to both client and server, and the client will also know g's order q.

3. Delegating Exponentiation in General Groups

In this section we present our protocols for the delegation of exponentiation in a general class of groups to a single (possibly malicious) server.

We note that general conversion techniques are known in the cryptography literature to transform a protocol secure against a honest adversary into one secure against a malicious adversary. Typically these techniques are based on zero-knowledge proofs of knowledge of secrets that certify the correctness of the computation, a methodology often used in cryptography papers since [27]. 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 require the verifier to perform group exponentiations, which is precisely what the client is trying to delegate in our protocols. Accordingly, new techniques are needed. Our first protocol, in Section 3.1, uses a direct parallel repetition of an efficient subprotocol that achieves security probability $1/2$, this latter subprotocol being an improved version of our scheme from Section 5 of [12]. Our second protocol, in Section 3.2, is actually a parameterized class of protocols where, for some values of two parameters c, m, the security probability is reduced more efficiently than by direct parallel repetition. The protocols in Sections 3.1 and 3.2 are presented for fixed-exponent variable-base exponentiation. Analogue protocols are achieved for variable-exponent fixed-base exponentiation and are briefly presented in Section 3.3. Finally, in Section 3.4 we show performance results from our software implementation of these protocols.

3.1. Delegating Fixed-Exponent Variable-Base Exponentiation: a Cut-and-choose Approach

We first describe a basic protocol (bC_1, bS_1) with constant security probability (obtained by simplifying the protocol in Section 5 of [12]) and then the final protocol (fC_1, fS_1), obtained as a parallel repetition of the basic protocol.

A protocol (bC_1, bS_1) *with constant security probability.* In an offline phase, bC_1 randomly chooses $u_0, u_1 \in G, b \in \{0, 1\}$ and computes $v_b = u_b^k$ and $v_{1-b} = u_{1-b}^{-k}$. In the online delegation phase, bC_1 computes $z_b = u_b$ and $z_{1-b} = x * u_{1-b}$, and sends (z_0, z_1) to bS_1. Next, bS_1 computes $w_i = z_i^k$, for $i = 0, 1$ and sends (w_0, w_1) to bC_1. Finally, bC_1 checks that $w_b = v_b$; if not, bC_1 returns failure symbol $\perp$; otherwise, bC_1 returns $y = w_{1-b} * v_{1-b}$.

We now observe that protocol (bC_1, bS_1) satisfies correctness, privacy, security (with probability 1/2), and efficiency (with $t_C = 2$ multiplications, $t_S = 2$ exponentiations, and $t_P = 2$ exponentiations plus 1 inversion in G).

The *efficiency* properties are verified by protocol inspection. The *correctness* property follows by observing that if bC_1 and bS_1 follow the protocol, bC_1's equality verification is satisfied, and thus C's output y satisfies $y = w_{1-b} * v_{1-b} = z_{1-b}^k * u_{1-b}^{-k} = (x * u_{1-b})^k * u_{1-b}^{-k} = x^k$, which implies that $y = F_{G,exp,k}(x)$ for each $x \in G$. The *privacy* property follows by observing that the message z_0, z_1 sent by bC_1 does not leak any information about x, since they are randomly and independently distributed in G, as so are chosen u_0 and u_1. To see that the *security* property is satisfied, for any probabilistic polynomial-time adversary corrupting bS_1, consider the values w_0, w_1 returned by the adversary to bC_1. If the adversary honestly computes $w_i = z_i^k$ for both $i = 0, 1$, then the probability it fools bC_1 into an incorrect output y is 0. Thus, assume the adversary computes $w_c \neq z_c^k$, for some bit $c \in \{0, 1\}$. Then note that bC_1 will find this out and return failure symbol $\perp$ when $b = c$, and, since the message (z_0, z_1) leaks no information about b, the equality $b = c$ holds with probability at least $1/2$. This implies that the probability that the adversary fools bC_1 into an incorrect output y is $\leq 1/2$.

A protocol (fC_1, fS_1) *with exponentially small security probability.* Protocol (fC_1, fS_1) consists of λ parallel executions of the basic protocol (bC_1, bS_1), with the only additional modification that the output of fC_1 is defined as y if in all λ parallel executions bC_1 would return the same value y, or as failure symbol $\perp$ otherwise (that is, if bC_1 returns $\perp$ in any one of the parallel executions, or two different values $\neq\perp$ in any two of the parallel executions).

Protocol (fC_1, fS_1) satisfies correctness, privacy, security (with probability $1/2^\lambda$), and efficiency (with $t_C = 2\lambda$ multiplications, $t_S = 2\lambda$ exponentiations, $t_P = 2\lambda$ random bases to known-exponent exponentiations plus λ inversion in G, and $cc = O(\lambda\sigma)$). The proof of these properties is a direct extension of the proofs for the properties of (bC_1, bS_1).

We remark that for the typical setting $\lambda = 128$, C only performs 256 group multiplications. This is about 1 order of magnitude smaller than 1.5σ, the average number of group multiplications in the square-and-multiply algorithm, which can be $= 3072$ for the setting $\sigma = 2048$, which has been recommended on some commonly used groups in cryptography.

3.2. Delegating Fixed-Exponent Variable-Base Exponentiation: Improved Security Probability Reduction

In this subsection we improve the approach in Section 3.1 by studying computation-efficient (in terms of C's parameters t_P, t_C) reductions of the security probability ϵ_s. Our overall approach towards this goal can be briefly summarized as follows: first, we propose a basic protocol (bC_2, bS_2) with improved constant security probability and then we define a final protocol (fC_2, fS_2) that performs a suitable parallel repetition (with a reduced number of repetitions) of this basic protocol.

Theorem 3.1. Let σ, λ be security parameters and c, m be protocol parameters. There exists (constructively) a client-server protocol (bC_2, bS_2) for delegated computation of function $F_{G,exp,k}$ which satisfies

1. δ_c-correctness, for $\delta_c = 1$
2. ϵ_s-security, where ϵ_s is a constant that depends on parameter c (see Table 1 for exact values, ranging between 0.10763, for $c = 2, m = 100$, to 0.04538, for $c = 9, m = 100$)
3. ϵ_p-privacy, for $\epsilon_p = 0$
4. efficiency with parameters (t_F, t_P, t_C, t_S, cc), where
   - t_F is = 1 group exponentiation in G
   - t_S is = m group exponentiations in G
   - t_P is = c group exponentiations with random bases in G and 1 inversion in G
   - t_C is = 2 group multiplications in G
   - $cc = 2m$ elements in G

We remark that this protocol strictly improves the security probability achievable when bC_2 only performs 2 group multiplication in G during the protocol. As a comparison, the atomic protocol from Section 3.1 was only achieving security probability $1/2$. However, there is a tradeoff with the other metrics, as this protocol does increase the number of group exponentiations from bS_2 and the number of precomputed group exponentiations with random bases. Other comparisons for specific values of parameter c appear in the proof of the protocol's properties.

Informal description of protocol (bC_2, bS_2). Our main approach consists of reducing the security probability by a more time-efficient approach than the direct parallel repetition approach in Section 3.1. While we do not know how to avoid the above parallel repetition, we show that we can reduce the number of repetitions by designing a more efficient protocol with security probability much smaller than $1/2$. As a first simple example of this approach, by starting from protocol (bC_1, bS_1) with security probability $1/2$ from Section 3.1, and including 2 random 'decoy' values in G in the client's message to the server, we obtain a protocol with the following properties: (1) it does *not* increase the client's number of multiplications, (2) it only slightly increases computation by the server; and (3) it can be seen to reduce the security probability from 1/2 to $1/3$. Our protocol generalizes this idea of using random decoy values in G to a parameterized number m, also representing an upper bound on the number of values that the client sends to the server. This generalization reduces the security probability, even though not as much as we would like. Accordingly, the other idea is that of increasing the number of equality checks, since these are much less expensive than modular multiplications. We then introduce a second parameter c, representing an upper bound on the number of equality checks that the client wants to execute, as well as the number of pre-computed exponentiations that the client can afford. Specifically, in the resulting protocol, of the m values in G sent by the client to the server, one value is used to compute the function output, $c - 1$ values are used to perform equality checks, and $m - c$ values are decoy values. The resulting protocol achieves a security probability which is, very roughly speaking, linear in $1/c$, and thus the number of repetitions to reduce the probability to $2^{-\lambda}$, can be reduced to about $\lambda / \log_2 c$. We can actually define a class of protocols that is parameterized by c and m and analyze what values for these parameters give us a more time-efficient reduction of the security probability than what achieved in Section 3.1. The two main high-level takeaways on that analysis are: (1) a moderately large value for m is just as good as a huge value; (2) values of $c \in \{4, \ldots, 9\}$ result in a reduced number of group multiplications from the client.

Formal description of protocol (bC_2, bS_2)**.** Let G be an efficient group of order q, and a known exponent value $k \in \mathbb{Z}_q$.

Input to bS_2 *and* bC_2*:* 1^σ, $desc(F_{G,exp,k})$, and parameters $1^c, 1^m$

Input to bC_2*:* $x \in G$

Offline instructions:

1. bC_2 randomly chooses distinct $j_1, \ldots, j_m \in \{1, \ldots, m\}$
2. bC_2 randomly chooses $u_i \in G$, sets $v_i = u_i^k$, for $i = 1, \ldots, c-1$, $v_c = u_c^{-k}$ and $z_{j_i} = u_i$, for $i = 1, \ldots, c$
3. bC_2 randomly and independently chooses $z_{j_{c+1}}, \ldots, z_{j_m} \in G$

Online instructions:

```
1. $bC_2$ sets $z_{j_c} = x * u_c$ and sends $z_1, \ldots, z_m$ to $bS_2$
2. $bS_2$ computes $w_j = z_j^k$ for $j = 1, \ldots, m$
   $bS_2$ sends $w_1, \ldots, w_m$ to $bC_2$
3. if $w_{j_1} \neq v_{j_1}$ or $w_{j_2} \neq v_{j_2}$ or ... or $w_{j_{c-1}} \neq v_{j_{c-1}}$ then
        $bC_2$ returns: $\perp$ and the protocol halts
   $bC_2$ computes $y = w_{j_c} * v_c$ and returns: $y$
```

Properties of protocol (bC_2, bS_2)**:** The *efficiency properties* are verified by protocol inspection. In particular, note that during the protocol bC_2 only performs 2 multiplications in G, and S performs c exponentiations in G. During the offline phase, bC_2 performs c exponentiations in G and 1 subtraction in G.

The *correctness* properties follows by observing that if bC_2 and bS_2 follow the protocol, none of the inequality verifications in step 3 will be satisfied. Thus, bC_2's output is $\neq \perp$ and is equal to $y = w_{j_c} * v_c = z_{j_c}^k * v_c = (x * u_c)^k * u_c^{-k} = x^k$, which implies that bC_2's output is $= F_{G,exp,k}(x)$ for each $x \in G$.

The *privacy* property follows by observing that the message $z_1, \ldots, z_m$ sent by bC_2 does not leak any information about x. Note that values in this message are generated in 3 different ways, and are in all 3 cases uniformly and independently distributed in G. Specifically, bC_2 sets $z_{j_1}, \ldots, z_{j_{c-1}}$ as equal to $u_1, \ldots, u_{c-1}$, respectively, and the latter are uniformly and independently chosen from G. Moreover, C sets z_{j_c} as $x * u_c$, which is still uniformly distributed in G, since so is u_c, for any $x \in G$. Finally, bC_2 uniformly and independently chooses $z_{j_{c+1}}, \ldots, z_{j_m}$ from G. This concludes the proof of the privacy, as defined in Section 2. We also observe that, by the same reasons of this proof, protocol (bC_2, bS_2) satisfies the following property: for any x, $z_1, \ldots, z_m$ are uniformly and independently distributed in G. We will use this latter fact in the proof of the security property.

To prove the *security* property against a malicious bS_2 we need to compute an upper bound ϵ_s on the security probability that bS_2 convinces bC_2 to output a y such that $y \neq F_{G,exp,k}(x)$. With respect to a random execution of (bC_2, bS_2) where bC_2 uses x as input, we define the following events:

- $e_{y,\neq}$, defined as 'bC_2 outputs y such that $y \neq F_{G,exp,k}(x)$'
- $e_{y,=}$, defined as 'bC_2 outputs y such that $y = F_{G,exp,k}(x)$'
- $e_\perp$, defined as 'bC_2 outputs $\perp$'

By inspection of (bC_2, bS_2), we see that the events $e_{y,\neq}, e_{y,=}, e_\perp$ are mutually exclusive, and one of them always happens. We obtain the following fact.

Fact 3.1. Event $e_{y,\neq}$ happens if and only if event $(\neg\, e_\perp) \wedge (\neg\, e_{y,=})$ happens.

With respect to a random execution of (bC_2, bS_2) where bC_2 uses x as input, we recall that $j_1, \ldots, j_m$ denote distinct values randomly chosen from $\{1, \ldots, m\}$ in step 1 of the protocol in offline phase, and $(w_1, \ldots, w_m)$ denotes the message sent by bS_2 to bC_2 in step 2 of the protocol in online phase. Then, we further define eW as the set $\{j | \, w_j = F_{G,exp,k}(z_j)\}$and dW the set $\{j | \, w_j \neq F_{G,exp,k}(z_j)\}$. Note that (eW, dW) is a partition of $\{1, \ldots, m\}$. We observe that bC_2 does not output $\perp$ whenever $j_1, \ldots, j_{c-1}$ belong to eW, and that bC_2 does not output $y = F_{G,exp,k}(x)$ whenever j_c belong to dW. We obtain the following fact.

Fact 3.2. It holds that:

1. Prob $[\neg \, e_\perp] =$ Prob $[j_1 \in eW \, \wedge \ldots \wedge \, j_{c-1} \in eW]$,
2. Prob $[\neg \, e_{y,=}] =$ Prob $[j_c \in dW]$.

Note that since bS_2 is malicious, eW could even be the empty set. However, note that the strategy of incorrectly computing all w_i's is not a good one for bS_2 as the resulting message $(w_1, \ldots, w_m)$ will not pass bC_2's verifications and bC_2 will then output failure symbol $\perp$. More generally, a malicious bS_2 can choose the w_i, as some arbitrary function of the message $(z_1, \ldots, z_m)$, as well as of other information public to both protocol participants. However, we now observe that bS_2 cannot choose this message as a function of the random values $j_1, \ldots, j_m$ chosen by bC_2 in offline phase of the protocol. This follows from the definition of the values $z_1, \ldots, z_m$, which makes them to have a distribution independent from that of $j_1, \ldots, j_m$. Specifically, as already observed when proving the privacy property of (bC_2, bS_2), the values $z_1, \ldots, z_m$ are uniformly and independently distributed in G, regardless of values $j_1, \ldots, j_m$. We obtain the following

Fact 3.3. The distribution of values $w_1, \ldots, w_m \in G$ is independent from the distribution of values $j_1, \ldots, j_m \in \{1, \ldots, m\}$.

This latest fact is important while studying strategies available to S, implying that S cannot compute the w_i's based on the random values $j_1, \ldots, j_c$ chosen by C in step 1 of the protocol.

The rest of the proof consists of computing an upper bound ϵ_s on the probability of event $e_{y,\neq}$ via an analysis of the best strategy for bS_2, and considers 4 cases, depending on the value of parameter c in protocol (bC_2, bS_2).

Case $c = 2$. In this case, bC_2 is including $m - 2$ decoy elements in G to its message $z_1, \ldots, z_m$ in step 1 and then performing a single inequality check in step 3 of the protocol. We have the following

$$\begin{aligned} \text{Prob}\,[\,e_{y,\neq}\,] &= \text{Prob}\,[\,\neg\, e_\perp\,]\,\text{Prob}\,[\,\neg\, e_{y,=} | \neg\, e_\perp\,] \\ &= \text{Prob}\,[\,j_1 \in eW\,]\,\text{Prob}\,[\,j_2 \in dW | j_1 \in eW\,] \\ &= \frac{|eW|}{m}\,\frac{|dW|}{m-1} = \frac{|eW|(m - |eW|)}{m(m-1)}, \end{aligned}$$

where the first equality follows from Fact 3.1, the second equality follows from Fact 3.2, the third equality follows from Fact 3.3, and the last equality from definitions of eW, dW.

Now, note that S can choose message $(w_1, \ldots, w_m)$ arbitrarily so to maximize the above probability. The above ratio is maximized by setting $|eW| = \lfloor m/2 \rfloor$ (as well as $|eW| = \lceil m/2 \rceil$), in which case we obtain that $\epsilon_s =$ Prob $[e_{y,\neq}] = m/(4(m-1))$, indicating that ϵ_s gets very close to $1/4$ as m grows.

This result can be interpreted by saying that using $m - 2$ random decoy elements in G as part of bC_2's message reduces the probability from $1/2$ (as in the atomic protocol from Section 3.1) to almost $1/4$, while requiring *no additional computation* of group multiplications from bC_2.

Case $c = 3$. In this case, bC_2 is including $m - 3$ decoy elements in G to its message $z_1, \ldots, z_m$ in step 1, using 3 pre-computed exponentiations with random exponents, and then performing 2 inequality checks in step 3 of the protocol.

By directly adapting the same probability derivation steps as in the case $c = 2$, we obtain that

$$\text{Prob}\,[\, e_{y,\neq} \,] = \frac{|eW|(|eW|-1)(m-|eW|)}{m(m-1)(m-2)}.$$

As before, note that bS_2 can choose message $(w_1,\ldots,w_m)$ arbitrarily so to maximize the above probability, and, after differentiation, we see that the above ratio is maximized by setting $|eW| = (m+1+\sqrt{m^2-m+1})/3$. By setting $m = 100$, we obtain that $\epsilon_s = \text{Prob}\,[\, e_{y,\neq} \,] = 0.1504 < 1/6$. Further increasing m does not help reducing the probability by much as, for instance, by setting $m = 1000$ we obtain that $\epsilon_s = \text{Prob}\,[\, e_{y,\neq} \,] = 0.1484$.

This result can be interpreted by saying that using $m-3$ random decoy elements in G as part of bC_2's message and 1 additional precomputed exponentiation of a random exponent reduces the probability from $1/2$ (as in the atomic protocol from Section 3.1) to slightly less than $1/6$, while requiring *no additional computation* of group multiplications from bC_2. As detailed later, when this protocol is combined with direct parallel repetition to reduce ϵ_s to $2^{-\lambda}$, the overall number of C's group multiplications is smaller than in the case $c = 2$.

Case $c = 4,\ldots,m-1$. In this case, bC_2 is including $m-c$ decoy elements in G to its message $z_1,\ldots,z_m$ in step 1, using c pre-computed exponentiations with random exponents, and then performing $c-1$ inequality checks in step 3 of the protocol.

By directly adapting the same probability derivation steps as in the case $c = 2, 3$, we obtain that

$$\begin{aligned}\text{Prob}\,[\, e_{y,\neq} \,] &= \text{Prob}\,[\, \neg\, e_{\perp} \,]\,\text{Prob}\,[\, \neg\, e_{y,=} | \neg\, e_{\perp} \,] \\ &= \text{Prob}\,[\, j_1,\ldots,j_{c-1} \in eW \,]\,\text{Prob}\,[\, j_c \in dW | j_1,\ldots,j_{c-1} \in eW \,] \\ &= \frac{\binom{|eW|}{c-1}}{\binom{m}{c-1}} \cdot \frac{m-|eW|}{m-c+1}.\end{aligned}$$

As before, note that bS_2 can choose message $(w_1,\ldots,w_m)$ arbitrarily so to maximize the above probability. To analyze the above ratio, differentiation does not help since there is a high-degree polynomial and known upper bounds on binomial coefficients are too loose. A tool-based trend analysis revealed that this ratio approximately behaves like $O(1/c)$ when studied as a function of c. Furthermore, we exactly computed the value

$$\epsilon_s = \text{Prob}\,[\, e_{y,\neq} \,] = \max_{j=4,\ldots,m-1} \frac{\binom{j}{c-1}}{\binom{m}{c-1}} \cdot \frac{m-j}{m-c+1},$$

for all values of c that guarantee some improved efficiency on the number t_C (of bC_2's group multiplications during the protocol) without making the number t_P (of group exponentiations with random exponents computed during the offline phase) too much worse. Specifically, we looked at all values of c such that the obtained ϵ_s is smaller than what could be obtained by a parallel repetition of $\lfloor c/2 \rfloor$ executions of the atomic protocol from Section 3.1 with security probability 1/2. It turns out that values $c = 4, 5, 6, 7, 8$ and 9 guarantee some improved efficiency on t_C without making t_P much worse, but starting from $c = 10$, the dependency of this protocol's value t_P on c starts becoming much larger than the one for the protocol in Section 3.1. The obtained values for ϵ_s when $c = 4,\ldots,10$ are reported in Table 1 below.

Note that when $c = 4,\ldots,9$ the obtained value for ϵ_s is strictly smaller than the value $2^{-\lfloor c/2 \rfloor}$ that could be obtained using the protocol from Section 3.1. Instead, when $c = 10$, the value $\epsilon_s = 0.03894$ is $> 0.03125 = 2^{-5}$, and the protocol from Section 3.1 starts arguably offering a much better efficiency tradeoff.

Overall, this result can be interpreted by saying that using $m-c$ random decoy elements in G as part of bC_2's message and $c-2$ additional precomputed exponentiations with random exponent reduces the probability from $1/2$ (as in the atomic protocol from Section 3.1) to approximately

TABLE 1. Values of ϵ_s for protocol (bC_2, bS_2), for $c = 4, \ldots, 10$ and $m = 100, 1000$

$c =$	4	5	6	7	8	9	10
$m = 10$	.13333	.11111	.10000	.10000	.10000	.10000	.10000
$m = 20$	.11739	.09391	.07982	.06842	.06316	.05789	.05263
$m = 40$	.11106	.08212	.07249	.06219	.05465	.04918	.04442
$m = 60$	.10912	.08403	.07053	.06024	.05117	.04692	.04232
$m = 80$	.10818	.08457	.06963	.05930	.05169	.04588	.04135
$m = 100$	.10763	.08403	.06906	.05874	.05117	.04538	.04080
$m = 1000$	.10568	.08212	.06718	.05685	.04928	.04350	.03894

$O(1/c)$, while requiring *no* additional group multiplication from bC_2. As detailed later, when this protocol is combined with direct parallel repetition to reduce ϵ_s to $2^{-\lambda}$, the overall number of bC_2's group multiplications gets smaller as c increases. This comes at a cost of increasing the number of precomputed exponentiations with random exponent, but this tradeoff might be acceptable in applications where many precomputed exponentiations with random exponents are available or easy to obtain, especially in the parameter setting $c \leq 9$.

<u>Case $c = m$.</u> In this case, C is not including any decoy elements in G to its message $z_1, \ldots, z_m$ in step 1, is using $c = m$ pre-computed exponentiations with random exponents, and then performing $m-1$ inequality checks in step 3 of the protocol. By directly adapting the same probability derivation steps as in the case $c = 2, \ldots, m-1$, we obtain that the probability Prob $[\, e_{y,\neq} \,]$ is > 0 only when $|eW| = m - 1$ and $|dW| = 1$, in which case we obtain that

$$\begin{aligned} \epsilon_s = \text{Prob}\,[\, e_{y,\neq} \,] &= \text{Prob}\,[\, \neg\, e_{\perp} \,]\, \text{Prob}\,[\, \neg\, e_{y,=} | \neg\, e_{\perp} \,] \\ &= \text{Prob}\,[\, j_1, \ldots, j_{m-1} \in eW \,]\, \text{Prob}\,[\, j_m \in dW | j_1, \ldots, j_{m-1} \in eW \,] \\ &= \frac{1}{m} \cdot 1 = \frac{1}{m}. \end{aligned}$$

This result can be interpreted by saying that this protocol can reduce the security probability to $1/m$, even in the potentially interesting case $m = \omega(1)$. However, this comes at a cost, in terms of precomputed exponentiations of random exponents, that is higher than the cost incurred with the protocol from Section 3.1, when adapted to reach the same security probability. □

A protocol (fC_2, fS_2) with exponentially small security probability. Protocol (fC_2, fS_2) consists of $r = \lceil \lambda / \log(1/\epsilon_s) \rceil$ parallel executions of the basic protocol (bC_2, bS_2), with the only additional modification that the output of fC_1 is defined as y if in all λ parallel executions bC_1 would return the same value y, or as failure symbol $\perp$ otherwise (that is, if bC_1 returns $\perp$ in any one of the parallel executions, or two different values $\neq\perp$ in any two of the parallel executions).

We obtain the following

Theorem 3.2. Let σ, λ be security parameters and c, m be protocol parameters. There exists (constructively) a client-server protocol (fC_2, fS_2) for delegated computation of function $F_{G,exp,k}$ which satisfies

1. δ_c-correctness, where $\delta_c = 1$
2. ϵ_s-security, where $\epsilon_s = 2^{-\lambda}$
3. ϵ_p-privacy, for $\epsilon_p = 0$
4. efficiency with parameters (t_F, t_P, t_C, t_S, cc), where $r = \lceil \lambda / \log_2(1/\epsilon_s) \rceil$ and
   - t_F is = 1 group exponentiation in G
   - t_S is = $m \cdot r$ group exponentiations in G
   - t_P is = $c \cdot r$ group exponentiations with random bases and r inversions in G
   - t_C is = $2r$ group multiplications in G

- $cc = 2m \cdot r$ elements in G.

Protocol (fC_2, fS_2) satisfies correctness, privacy, security (with probability $1/2^\lambda$), and efficiency (with $t_C = 2r$ group multiplications, $t_S = mr$ group exponentiations, $t_P = cr$ group exponentiations with random bases and r inversions, and $cc = 2mr\sigma$).

The proof of these properties is obtained by extension of the proofs for the properties of (bC_2, bS_2). We remark that for the typical setting $\lambda = 128$, C performs as low as 56 group multiplications and the number of group multiplications is 1 to 2 orders of magnitude smaller than 1.5σ, the average number of group multiplications in the square-and-multiply algorithm, which can be $= 3072$ for the setting $\sigma = 2048$, recommended on some commonly used groups in cryptography.

To numerically evaluate the efficiency of this protocol, we evaluate its main efficiency metrics, with the typical setting of $\lambda = 128$, in Table 2 below.

TABLE 2. Performance and security parameters for (fC_2, fS_2) when $m = 100$ and $c = 2, \ldots, 9$. Here, t_P is number of exponentiations, and t_C is number of multiplications.

c	2	3	4	5	6	7	8	9	10
ϵ_s when $r = 1$	.2526	.1504	.1076	.0840	.0672	.0588	.0512	.0454	.0408
r	65	47	40	36	33	32	30	29	28
t_P	130	141	160	180	198	224	240	261	280
t_C	130	94	80	72	66	64	60	58	56

Note that it could be possible to further reduce the number of fC_2's group multiplications during the protocol below 28 (the number obtained for $m = 100, c = 10$). However, as mentioned before, starting from $c = 10$, the protocol offers an arguably worse tradeoff with the other efficiency metrics (mainly, the number of group exponentiations with random exponents from the offline phase, the number of group exponentiations from fS_2, etc.) than the protocol from Section 3.1.

3.3. Delegating Variable-Exponent Fixed-Base Exponentiation

We describe our protocols for the delegation of variable-exponent fixed-base exponentiation over our general class of groups. They are obtained by applying notation changes to the protocols in Section 3.1 and 3.2, which happen to result in slightly more efficient client online runtime.

Let G be an efficient group of order q, and let $g \in G$ be a base value known to both parties.

A protocol (bC_3, bS_3) with constant security probability. In an offline phase, bC_3 randomly chooses $u_0, u_1 \in \mathbb{Z}_q$ and computes $v_0 = g^{u_0}$ and $v_1 = g^{u_1}$. In the delegation phase, bC_3 randomly chooses bit $b \in \{0, 1\}$ and computes $z_b = u_b$ and $z_{1-b} = x - u_{1-b} \mod q$, and sends (z_0, z_1) to bS_3. Next, bS_3 computes $w_i = g^{z_i}$, for $i = 0, 1$ and sends (w_0, w_1) to bC_1. Finally, bC_3 returns failure symbol $\perp$ if $w_b \neq v_b$ or returns $y = w_{1-b} * v_{1-b}$ otherwise. We note that bC_3 requires one less multiplication than bC_1.

A protocol (fC_3, fS_3) with exponentially small security probability. Protocol (fC_3, fS_3) consists of λ parallel executions of the basic protocol (bC_3, bS_3), with the only additional modification that the output of fC_3 is defined as y if in all λ parallel executions bC_3 would return the same value y, or as failure symbol $\perp$ otherwise (that is, if bC_3 returns $\perp$ in any one of the parallel executions, or two different values $\neq\perp$ in any two of the parallel executions).

A protocol (bC_4, bS_4) with constant security probability.

Input to bS_4 and bC_4: 1^σ, $desc(F_{G,exp,g})$, $g \in G$, parameters $1^c, 1^m$

Input to bC_4: $x \in \mathbb{Z}_q$

Offline instructions:

TABLE 3. Performance of Protocols (bC_1, bS_1) and (bC_2, bS_2) on 2048-bit input lengths

	(bC_1, bS_1)	(bC_2, bS_2)							
c	NA	**5**		**6**		**7**		**8**	
m	NA	**60**	**100**	**60**	**100**	**60**	**100**	**60**	**100**
ϵ_s	.50000	.08403	.08403	.07053	.06719	.06024	.05875	.05117	.05118
t_F	.003534	.003631	.003702	.003685	.003650	.003686	.003534	.003728	.003889
t_P	.007150	.018332	.019067	.022506	.022261	.025946	.025042	.030026	.031786
t_C	.000012	.000021	.000023	.000021	.000022	.000022	.000021	.000023	.000027
t_S	.007084	.217909	.369758	.219471	.364578	.219965	.354359	.224488	.381499
$\frac{t_F}{t_C}$	290.439	170.183	159.103	173.823	162.595	169.744	169.855	161.170	145.072

1. bC_4 randomly chooses distinct $j_1, \ldots, j_m \in \{1, \ldots, m\}$
2. bC_4 randomly chooses $u_i \in \mathbb{Z}_q$, sets $v_i = g^{u_i}$ and $z_{j_i} = u_i$, for $i = 1, \ldots, c$
3. bC_4 randomly and independently chooses $z_{j_{c+1}}, \ldots, z_{j_m} \in \mathbb{Z}_q$

Online instructions:

1. bC_4 sets $z_{j_c} = (x - u_c) \mod q$ and sends $z_1, \ldots, z_m$ to bS_2
2. bS_4 computes $w_j = g^{z_j}$ for $j = 1, \ldots, m$
   bS_2 sends $w_1, \ldots, w_m$ to bC_2
3. if $w_{j_1} \neq v_{j_1}$ or $w_{j_2} \neq v_{j_2}$ or $\ldots$ or $w_{j_{c-1}} \neq v_{j_{c-1}}$ then
       bC_4 **returns:** $\perp$ and the protocol halts
   bC_4 computes $y = w_{j_c} * v_c$ and **returns:** y

A protocol (fC_4, fS_4) with exponentially small security probability. Protocol (fC_4, fS_4) consists of $r = \lceil \lambda / \log(1/\epsilon_s) \rceil$ parallel executions of the basic protocol (bC_4, bS_4), with the only additional modification that the output of fC_4 is defined as y if in all λ parallel executions bC_1 would return the same value y, or as failure symbol $\perp$ otherwise (that is, if bC_4 returns $\perp$ in any one of the parallel executions, or two different values $\neq \perp$ in any two of the parallel executions). This protocol satisfies correctness, privacy, security (with probability $1/2^\lambda$), and efficiency with complexity similar to those in (fC_2, fS_2), with the notable difference that the number of online group multiplications by the client is reduced by a multiplicative factor of 2.

3.4. Software Implementation and Performance Results

We implemented our protocols in Sections 3.3, choosing as example group the multiplicative group $(\mathbb{Z}_p^*, \cdot \mod p)$, for $p = 2q + 1$, and p, q are large primes such that $|p| = 2048$.

Our implementation was carried out on a macOS High Sierra Version 10.13.4 laptop with 2.7 GHz Intel Core i5 processor with memory 8 GB 1867 MHz DDR3. The protocols were coded in Python 3.6 using the gmpy2 package.

The obtained performance data is grouped in two tables. Table 3 contains parameters c, m, ϵ_s, running times t_F, t_P, t_C, t_S and improvement ratio t_F/t_C for protocol (bC_1, bS_1) from Section 3.1 and protocol (bC_2, bS_2) from Section 3.2. Similarly, Table 4 contains parameters c, m, r, running times t_F, t_P, t_C, t_S and improvement ratio t_F/t_C for protocol (fC_1, fS_1) from Section 3.1 and protocol (fC_2, fS_2) from Section 3.2. Here, parameter r represents the number of parallel repetitions of (bC_1, bS_1) and (bC_2, bS_2) needed to get desired security probability $\epsilon_s = 2^{-128}$ in protocols (fC_1, fS_1) and (fC_2, fS_2), respectively.

A software implementation of protocols in Section 3.1 and 3.2 is expected to produce very similar (and only slightly less efficient) performance results.

TABLE 4. Performance of Protocols (fC_1, fS_1) and (fC_2, fS_2), for $\epsilon_s = 2^{-128}$

	(fC_1, fS_1)	(fC_2, fS_2)							
c	NA	**5**		**6**		**7**		**8**	
m	NA	**60**	**100**	**60**	**100**	**60**	**100**	**60**	**100**
r	128	36	36	34	33	32	32	30	30
t_F	.003651	.003799	.003637	.003851	.003804	.003705	.004338	.003709	.004046
t_P	.953298	.686819	.684862	.769721	.770282	.850836	.862347	.910533	.962034
t_C	.000779	.000393	.000268	.000443	.000270	.000378	.000289	.000367	.000304
t_S	.957278	8.05238	13.2509	7.58752	12.4730	7.15478	12.1795	6.70609	11.7280
$\frac{t_F}{t_C}$	4.68362	9.67187	13.5511	8.68811	14.0649	9.79045	15.0138	10.1090	13.3077

4. Conclusions

We studied the problem of a computationally weak client delegating group exponentiation to a single, possibly malicious, computationally powerful server, as originally left open in [30]. We solved this problem by two protocols that provably satisfy formal correctness, privacy (against adversaries of unlimited power), security (with exponentially small probability) and efficiency requirements, in a general class of multiplicative groups, including groups on which no quantum cryptanalysis attacks are currently known. Open problems include: (a) achieving better efficiency tradeoffs as done in [19] for discrete logarithm groups and in [20] for RSA groups, where similar improvements on the client's online runtime were achieved with only constant overhead to server runtime and communication complexity; and (b) reducing the dependency of the offline computations on the number of delegated computations of F (in our protocols, as well as previous protocols in the literature, when delegating many computations of F, the complexity of the offline phase increases at least linearly with such computations).

References

[1] L. Adleman, M. Huang and K. Kompella, *Efficient Checkers for Number-Theoretic Computations*, in Inf. Comput. 121(1): pages 93-102, 1995

[2] I. Anshel, D. Atkins, D. Goldfeld, and P. E. Gunnels, *Post Quantum Group theoretic Cryptography*, in https://www.securerf.com/wp-content/uploads/2017/01/SecureRF-GTDH-Quantum-Resistant-12-16.pdf, Nov 2016, vol. 10, pp. 1–11.

[3] A. Arbit and Y. Livne and Y. Oren and A. Wool, *Implementing public-key cryptography on passive RFID tags is practical*, in: International Journal of Information Security 2015, vol. 14 (1), pp. 85-99, Springer.

[4] M. Atallah, K. N. Pantazopoulos, J. Rice, E. Spafford, *Secure outsourcing of scientific computations*, in Advances in Computers 2001, vol. 54 (6), pp. 215-272.

[5] M. Atallah and K. Frikken, *Securely outsourcing linear algebra computations*, in Proceedings of the 5th ACM Symposium on Information, Computer and Communications Security, pp. 48-59.

[6] M. Bellare, J. Garay, and T. Rabin, *Fast batch verification for modular exponentiation and digital signatures*, in International Conference on the Theory and Applications of Cryptographic Techniques, 1998, vol. 1403, pp 236–250, Springer.

[7] D. Benjamin and M. Atallah, *Private and cheating-free outsourcing of algebraic computations*, in Sixth Annual Conference on Privacy, Security and Trust, PST 2008, pp. 240-245, Springer-Verlag.

[8] M. Blum and S. Kannan, *Designing Programs that Check Their Work* in J. ACM 42(1): 269-291 (1995)

[9] M. Blum, M. Luby and R. Rubinfeld, *Self-Testing/Correcting with Applications to Numerical Problems*, in J. Comput. Syst. Sci. 47(3): 549-595 (1993)

[10] V. Boyko and M. Peinado and R. Venkatesan, *Speeding up discrete log and factoring based schemes via precomputations*, in Advances in Cryptology, EUROCRYPT 1998, LNCS 1403, pp. 221-235, Springer.

[11] S. Canard, J. Devigne, O. Sanders: *Delegating a pairing can be both secure and efficient.* In Proc. of ACNS 2014. LNCS 8479, pp. 549-565, Springer.

[12] B. Cavallo, G. Di Crescenzo, D. Kahrobaei, and V. Shpilrain, *Efficient and secure delegation of group exponentiation to a single server*, in International Workshop on RFID: Security and Privacy Issues 2015, LNCS 9440, pp 156–173, Springer.

[13] 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, LNCS 7459, pp. 541–556, Springer.

[14] C. Chevalier, F. Laguillaumie, D. Vergnaud, *Privately Outsourcing Exponentiation to a Single Server: Cryptanalysis and Optimal Constructions*. In Proc. of ESORICS (1) 2016, LNCS 9878, pages 261-278, Springer.

[15] B. Chevallier-Mames, J. Coron, N. McCullagh, D. Naccache, M. Scott: *Secure delegation of elliptic-curve pairing*. In Proc. of CARDIS 2010, LNCS 6035, pp. 24-35, Springer.

[16] K. Chung, Y. Kalai, and S. Vadhan, *Improved delegation of computation using fully homomorphic encryption*, in Proc. of 30th Annual Cryptology Conference 2010, LNCS 6223, pp. 483–501, Springer.

[17] K. Chung, Y. Kalai, and F. Liu, *Memory delegation*, in Proc. of the 31st Annual Cryptology Conference, LNSC 6841, pp. 151-168 Springer.

[18] 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 Workshop on Security and Privacy in the Cloud, CNS 2017, IEEE.

[19] 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 2017 (CCSW2017), pp. 17–28.

[20] G. Di Crescenzo, M. Khodjaeva, D. Kahrobaei, and V. Shpilrain, *Secure Delegation to a Single Malicious Server: Exponentiation in RSA-type Groups*, in Proc. of 5th IEEE Workshop on Security and Privacy in the Cloud, CNS 2019, IEEE.

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

[22] Y. Ding, Z. Xu, J. Ye, and K. Choo, *Secure outsourcing of modular exponentiations under single untrusted programme model*, In Journal of Computer and System Sciences, vol.90, C, Academic Press, Inc., pp 1–13, 2017, Elsevier.

[23] D. Fiore and R. Gennaro, *Publicly verifiable delegation of large polynomials and matrix computations, with applications*. In Proc. of ACM Conference on Computer and Communications Security 2012, pp. 501512.

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

[25] C. Gentry, *Fully homomorphic encryption using ideal lattices*, in Proceedings of the forty-first annual ACM symposium on Theory of computing 2009, pp. 169178.

[26] M. Girault and D. Lefranc, *Server-aided verification: Theory and practice*. In Proc. of ASIACRYPT 2005, LNCS 3788, pp. 605-623, Springer.

[27] O. Goldreich, S. Micali, A. Wigderson, *How to Play any Mental Game or A Completeness Theorem for Protocols with Honest Majority*, in Proc. of ACM STOC 1987, pages 218-229

[28] J. Gryak and D. Kahrobaei, *The status of polycyclic group-based cryptography: A survey and open problems*, in Groups Complexity Cryptology 2016, vol. 8 (2), pp. 171–186, De Gruyter.

[29] D. Hart, D. Kim, G. Micheli, G. Pascual-Perez, C. Petit, and Y. Quek, *A Practical Cryptanalysis of WalnutDSA*, in IACR International Workshop on Public Key Cryptography 2018, pp. 381–406, Springer.

[30] S. Hohenberger and A. Lysyanskaya, *How to securely outsource cryptographic computations*, in Proc. of TCC 2005, LNCS 3378, pp. 264–282, Springer.

[31] K. Horan and D. Kahrobaei, *The Hidden Subgroup Problem and Post-quantum Group-Based Cryptography*, in Proc. of ICMS 2018, pages 218-226

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

[33] X. Ma and J. Li and F. Zhang, *Outsourcing computation of modular exponentiations in cloud computing*, in Cluster Computing 2013, vol. 16, pp. 787–796, Springer.

[34] T. Matsumoto, K. Kato and H. Imai, *An improved algorithm for secure outsourcing of modular exponentiations*. In Proc. of CRYPTO 1988, pp. 497506.

[35] P. Q. Nguyen and I. E. Shparlinski and J. Stern, *Distribution of modular sums and the security of the server aided exponentiation*, in Cryptography and Computational Number Theory 2001, progress in Computer Science and Applied Logic, vol 20, pp. 331–342, Springer.

[36] B. Parno, M. Raykova, V. Vaikuntanathan, *How to delegate and verify in public: verifiable computation from attribute-based encryption*, in Theory of Cryptography Conference 2012, vol. 7194, pp. 422439 Springer.

[37] P. W. Shor, *Algorithms for Quantum Computation: Discrete Logarithms and Factoring*, in Proceedings 35th annual symposium on foundations of computer science1994, pp. 124–134, IEEE.

[38] Y. Wang, Q. Wu, D. Wong, B. Qin, S. Chow, Z. Liu, and X. Tao, *Securely outsourcing exponentiations with single untrusted program for cloud storage*, in Computer Security-ESORICS 2014, LNCS 8712, pp.326–343, Springer.

[39] P. Wallden and E. Kashefi, *Cyber Security in the Quantum Era*, in Communications of the ACM, 62 (4): 120 (2019).

[40] A. C. Yao, *Protocols for secure computations*, in Proceedings of the 23rd Annual Symposium on Foundations of Computer Science 1982, pp. 160–168, IEEE.

[41] J. Ye, X. Chen, and J. Ma, *An improved algorithm for secure outsourcing of modular exponentiations*, in Proc. of 29th International Conference on Advanced Information Networking and Applications, 2015, pp. 73-76, IEEE.

Giovanni Di Crescenzo
Perspecta Labs, Basking Ridge, NJ, USA.
e-mail: gdicrescenzo@perspectalabs.com

Matluba Khodjaeva
John Jay College of Criminal Justice, CUNY, New York, NY, USA.
e-mail: mkhodjaeva@jjay.cuny.edu

Delaram Kahrobaei
University of York, Heslington, UK
e-mail: delaram.kahrobaei@york.ac.uk

Vladimir Shpilrain
City University of New York, New York, NY, USA.
e-mail: shpil@groups.sci.ccny.cuny.edu