Efficient and Secure Delegation to a Single Malicious Server: Exponentiation over Non-abelian Groups

Giovanni Di Crescenzo[1](✉), Delaram Kahrobaei[2], Matluba Khodjaeva[3], and Vladimir Shpilrain[4]

[1] Perspecta Labs, Basking Ridge, NJ, USA
gdicrescenzo@perspectalabs.com

[2] City University of New York, New York, NY, USA
DKahrobaei@gc.cuny.edu

[3] John Jay College, City University of New York, New York, NY, USA
mkhodjaeva@jjay.cuny.edu

[4] City University of New York, New York, NY, USA
shpil@groups.sci.ccny.cuny.edu

Abstract. Group exponentiation is an important and 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 a formal model was introduced in [8]. Recently, in [10] we proposed practical and secure solutions applicable to a class of cyclic groups. In this paper, we propose efficient and secure solutions applicable to a large class of multiplicative groups, possibly beyond groups currently subject to quantum cryptanalysis attacks.

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

D. Kahrobaei and V. Shpilrain—Research of Delaram Kahrobaei was partially supported by a PSC-CUNY grant from the CUNY research foundation, as well as the City Tech foundation. Research of Vladimir Shpilrain was partially supported by the NSF grant CNS-1117675. Research of Delaram Kahrobaei and Vladimir Shpilrain was also supported by the ONR (Office of Naval Research) grant N000141210758.

J. H. Davenport et al. (Eds.): ICMS 2018, LNCS 10931, pp. 137–146, 2018.
https://doi.org/10.1007/978-3-319-96418-8_17

groups; specifically, groups related to discrete logarithm or factoring problems (see, e.g., [4,5,8,10] and references therein).

As progresses are being made towards building a large-scale quantum computer, much attention is being devoted in the cryptography community to early quantum computer algorithms such as Shor's [9], 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. 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., [1,6,7] and references therein).

In this paper we consider the delegation of group exponentiation over a large class of general multiplicative groups, not limited to abelian groups and thus going beyond groups currently subject to quantum cryptanalysis attacks.

Our Contributions. We show two interactive protocols allowing a client to delegate exponentiation in a general class of groups to a single, possibly malicious, server, while satisfying 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 (i.e., the client's runtime is smaller than in a non-delegated computation of the exponentiation). Our first protocol, in Sect. 3.1, consists of a direct parallel repetition of (a slightly simplified version of) a protocol from [3] that achieves security probability 1/2. Our main result, in Sect. 3.2, is a parameterized class of protocols where, for some parameter values, 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., 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. Our software implementation, in Python 3.6, using commodity computing resources and the gmpy2 package, 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 client or another party can precompute fixed-base exponentiations to random exponents, and store them on the client's device to be later used in the online protocol phase. We also consider a model where a client can efficiently run group multiplications, which is partially justified by known application on some RFID devices results (see, e.g., [2]). Our protocols are written so to delegate $F_{G,exp,g}(x) = g^x$ (i.e., variable-exponent, fixed-base exponentiation over multiplicative group G), but can be reformulated so to delegate function $F_{G,exp,k}(x) = x^k$ (i.e., fixed-exponent, variable-base exponentiation).

2 Models and Definitions

In this section we define delegation protocols, and their correctness, security, privacy and efficiency requirements, building on the definitional approach from [3] (also based on [5,8]), and describe group notations and protocol preliminaries.

Participant and Protocol Models. We consider 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 C and S is not subject to confidentiality, integrity, or replay attacks, and note that such attacks can be separately addressed using well-known cryptography techniques. A *client-server protocol for the delegated computation of function* F is an interactive protocol between C and S, where both parties have a description of a function F, C knows an input x, and at the end of a protocol execution, C outputs a value y (intended to be $= F(x)$). The protocol can have two phases: an *offline phase*, including expensive computations not based on input x, such as evaluating F on other inputs, and an *online phase*, where C's computations are based on input x but take less time than what required to compute $F(x)$. We require such protocols to satisfy the following requirements of *correctness*, *security*, *privacy* and *efficiency*.

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 output of function F on C's input x.

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 $y = F(x)$ 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, 128.

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.

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, the offline phase requires t_P atomic operations, C requires t_C atomic operations in the online phase, S requires 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 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. Our theoretical target are protocols where t_C is 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. 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 *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 let $F_{G,exp,g} : \mathbb{Z}_q \to G$ denote the function that maps every $x \in \mathbb{Z}_q$ to the exponentiation (in G) of g to the x-th power.

Protocol Preliminaries. In all our protocols, inputs common to client and server include a description of the function $F_{G,exp,g}$ to be delegated, a description of group G, a group element g, a computational parameter 1^σ and a security parameter 1^λ. Other inputs to the client include g's order q and exponent $x \in \mathbb{Z}_q$.

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 computation correctness. 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 1st protocol, in Sect. 3.1, uses a direct parallel repetition of an efficient subprotocol with security probability 1/2, this latter subprotocol being an improved version of our scheme from (Sect. 5 of) [3]. Our 2nd protocol, in Sect. 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.

3.1 Delegating 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 Sect. 5 of [3]) 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 $u_0, u_1 \in \mathbb{Z}_q$ and computes $v_0 = g^{u_0}$ and $v_1 = g^{u_1}$. In the delegation phase, bC_1 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_1. Next, bS_1 computes $w_i = g^{z_i}$, 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 show that protocol (bC_1, bS_1) satisfies correctness, privacy, security (with probability 1/2), and efficiency (with $t_C = 1$ multiplication plus 1 subtraction, $t_S = 2$ exponentiations, and $t_P = 2$ exponentiations).

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 $y = w_{1-b} * v_{1-b} = g^{z_{1-b}} * g^{u_{1-b}} = g^{x-u_{1-b}} * g^{u_{1-b}} = g^x$, which implies that $y = F_{G,exp,g}(x)$ 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 $\mathbb{Z}_q$, 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 $w_i = g^{z_i}$ for both $i = 0, 1$, then the probability it fools bC_1 into an incorrect output y is 0. Thus, assume the adversary computes $w_c \neq g^{z_c}$, 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 $t_C = \lambda$ multiplications plus λ subtractions, $t_S = 2\lambda$ exponentiations, $t_P = 2\lambda$ known-base exponentiations to random exponents, 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 128 group multiplications and 128 subtractions modulo q. 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 Exponentiation: Improved Probability Reduction

In this subsection we improve the approach in Sect. 3.1 by a 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 define a final protocol (fC_2, fS_2) that performs a suitable parallel repetition (with a smaller number of repetitions) of this basic protocol.

Informal Discussion. Our main approach consists of reducing the security probability by a more time-efficient approach than the direct parallel repetition approach in Sect. 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 Sect. 3.1, and including 2 random 'decoy' values in $\mathbb{Z}_q$ 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; (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 $\mathbb{Z}_q$ 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, and introducing a second parameter c, representing an upper bound on the number of equality checks that the client wants to execute (and thus, the number of pre-computed exponentiations that the client can afford). Specifically, in the resulting protocol, of the m values in $\mathbb{Z}_q$ 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 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 Sect. 3.1. The two main high-level takeaways on that analysis are: (1) a somewhat 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.

A protocol (bC_2, bS_2) *with constant security probability.* We first formally describe the basic protocol (bC_2, bS_2) and then discuss its properties.

Offline instructions:

1. bC_2 randomly chooses distinct $j_1, \ldots, j_m \in \{1, \ldots, m\}$
2. bC_2 randomly chooses $u_i \in \mathbb{Z}_q$, sets $v_i = g^{u_i}$ and $z_{j_i} = u_i$, for $i = 1, \ldots, c$
3. bC_2 randomly and independently chooses $z_{j_{c+1}}, \ldots, z_{j_m} \in \mathbb{Z}_q$

Online instructions:

```
1. $bC_2$ sets $z_{j_c} = (x - u_c) \mod q$ and sends $z_1, \ldots, z_m$ to $bS_2$
2. $bS_2$ 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 ... 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:** $y$
```

We now observe that protocol (bC_2, bS_2) satisfies correctness, privacy, security (with probability $O(1/c)$), and efficiency (with $t_C = 1$ multiplication in G plus 1 subtraction in $\mathbb{Z}_q$, $t_S = m$ exponentiations, and $t_P = m$ exponentiations).

The *efficiency* properties of (bC_2, bS_2) are verified by protocol inspection. 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 $y = w_{j_c} * v_c = g^{z_{j_c}} * v_c = g^{z_{j_c}} * g^{u_c} = g^{x-u_c} * g^{u_c} = g^x$, which implies that bC_2's output is $= F_{G,exp,g}(x)$ for each $x \in \mathbb{Z}_q$. The *privacy* property follows by observing that the message $z_1, \ldots, z_m$ sent by bC_2 is distributed as m random and independent group values and therefore does not leak any information about x.

To prove the *security* property against a malicious bS_2 we compute an upper bound ϵ_s on the security probability that bS_2 convinces bC_2 to output a y such that $y \neq F_{G,exp,g}(x)$. This is performed by a long case analysis (omitted here for lack of space) for all c and m, and depending on whether the server replies to the client with correct or incorrect answers. Examples of results from this analysis include the following: (1) when $c = 2$, using $m - 2$ decoy elements in $\mathbb{Z}_q$, we have that ϵ_s gets very close to $1/4$ as m grows; (2) when $c = 3$, using $m - 3$ decoy elements in $\mathbb{Z}_q$, we obtain $\epsilon_s < 1/6$ when $m = 100$, and increasing m to 1000 does not reduce ϵ_s significantly. For general c, m, we computed the exact value for ϵ_s for all values of c that guarantee some improved efficiency on the number t_C (of client's group multiplications during the protocol). 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 Sect. 3.1 with security probability $1/2$. It turns out that only values $c = 4, 5, \ldots, 9$ guarantee some improved efficiency on t_C, with respect to the protocol in Sect. 3.1. The obtained values for ϵ_s when $c = 4, \ldots, 10$ are 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 Sect. 3.1. Instead, when $c = 10$, the value $\epsilon_s = 0.03894$ is $> 0.03125 = 2^{-5}$, and the protocol from Sect. 3.1 starts offering a much better efficiency tradeoff.

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
$m = 100, \epsilon_s =$	.10763	.08403	.06719	.05875	.05118	.04538	.04080
$m = 1000, \epsilon_s =$	.10568	.08213	.06529	.05686	.04929	.04351	.03894

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).

Protocol (fC_2, fS_2) satisfies correctness, privacy, security (with probability $1/2^{\lambda}$), and efficiency (with $t_C = r$ group multiplications, $rc - r$ group equality checks and r subtractions in $\mathbb{Z}_q$; $t_S = mr$ group exponentiations, $t_P = rc$ known-base group exponentiations to random exponents, and $cc = O(mr\sigma)$). 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 about 30 group multiplications, 30 subtractions in $\mathbb{Z}_q$, and less than 300 group equality checks. The number of group multiplications is about 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.

3.3 Implementation and Performance Results

We implemented our protocols in Sects. 3.1 and 3.2 for 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 of the offline phase, the client's online program and the server's program 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. Table 2 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 Sect. 3.1 and protocol (fC_2, fS_2) from Sect. 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. The main takeaway is that in the two protocols, the client's online running time is better than non-delegated computation by half or one order of magnitude, respectively.

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

t_F	.003838								
	(fC_1, fS_1)	(fC_2, fS_2)							
c	n/a	5		6		7		8	
m	n/a	60	100	60	100	60	100	60	100
r	128	36	36	34	33	32	32	30	30
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
$\frac{t_F}{t_C}$	4.92654	9.76534	14.3201	8.66315	14.2140	10.1528	13.2795	10.4572	12.6243

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 [8]. 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, possibly going beyond groups on which quantum cryptanalysis attacks are currently known. Problems of both theoretical and practical interest include: (a) achieving better efficiency tradeoffs as done in [10] for discrete logarithm groups; and (b) reducing the dependency of the offline computations on the number of delegated computations of F.

References

1. Anshel, I., Atkins, D., Goldfeld, D., Gunnels, P.E.: Post Quantum Group Theoretic Cryptography, November 2016. https://bit.ly/2svnv8z
2. Arbit, A., Livne, Y., Oren, Y., Wool, A.: Implementing public-key cryptography on passive RFID tags is practical. Int. J. Inf. Sec. **14**(1), 85–99 (2015)
3. Cavallo, B., Di Crescenzo, G., Kahrobaei, D., Shpilrain, V.: Efficient and secure delegation of group exponentiation to a single server. In: Mangard, S., Schaumont, P. (eds.) RFIDSec 2015. LNCS, vol. 9440, pp. 156–173. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-24837-0_10

4. Dijk, M., Clarke, D., Gassend, B., Suh, G., Devadas, S.: Speeding up exponentiation using an untrusted computational resource. Des. Codes Crypt. **39**(2), 253–273 (2006)
5. Gennaro, R., Gentry, C., Parno, B.: Non-interactive verifiable computing: outsourcing computation to untrusted workers. In: Rabin, T. (ed.) CRYPTO 2010. LNCS, vol. 6223, pp. 465–482. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-14623-7_25
6. Gryak, J., Kahrobaei, D.: The status of polycyclic group-based cryptography: a survey and open problems. Groups Complexity Cryptology **8**(2), 171–186 (2016)
7. Hart, D., Kim, D.H., Micheli, G., Pascual-Perez, G., Petit, C., Quek, Y.: A practical cryptanalysis of WalnutDSA$^{\mathrm{TM}}$. In: Abdalla, M., Dahab, R. (eds.) PKC 2018. LNCS, vol. 10769, pp. 381–406. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-76578-5_13
8. Hohenberger, S., Lysyanskaya, A.: How to securely outsource cryptographic computations. In: Kilian, J. (ed.) TCC 2005. LNCS, vol. 3378, pp. 264–282. Springer, Heidelberg (2005). https://doi.org/10.1007/978-3-540-30576-7_15
9. Shor, P.W.: Algorithms for quantum computation: discrete logarithms and factoring. In: Proceedings of 35th IEEE Symposium on Foundations of Computer Science (FOCS 1994), pp. 124–134 (1994)
10. Di Crescenzo, G., Khodjaeva, M., Kahrobaei, D., Shpilrain, V.: Practical and secure outsourcing of discrete log group exponentiation to a single malicious server. In: Proceedings of 9th ACM Cloud Computing Security Workshop (CCSW), pp. 17–28 (2017)