Batch Delegation of Exponentiation in Discrete Log and RSA Groups

GIOVANNI DI CRESCENZO, Perspecta Labs Inc., Basking Ridge, NJ, USA
MATLUBA KHODJAEVA, CUNY John Jay College of Criminal Justice. New York, NY, USA
DELARAM KAHROBAEI, University of York, UK, New York University (USA)
VLADIMIR SHPILRAIN, CUNY Graduate Center. New York, NY, USA

We consider the problem of a client efficiently, privately and securely delegating the computation of multiple group exponentiations to a computationally more powerful server (e.g. a cloud server). We provide the first practical and provable solutions to this batch delegation problem for groups commonly used in cryptography, based on discrete logarithm and RSA hardness assumptions. Our theoretical analysis suggests that our protocol reduce the client's running time by about 1 order of magnitude with respect to non-delegated computation. We show results from a software implementation on a commodity platform that essentially confirms this improvement without need for optimizations. Previous results either solved delegation of a single group exponentiation with limited security properties, or verification of multiple group exponentiations in prime-order groups (not applicable to RSA) and under certain simplifying assumptions on the exponentiation values (not applicable to some discrete logarithm groups). Our results directly solve batch delegation of algorithms in widely used cryptosystems, such as RSA encryption with large exponents and key contribution in Diffie-Hellman key agreement protocols.

CCS Concepts: • **Security and privacy → Public key (asymmetric) techniques**; • **Theory of computation → Cryptographic protocols**;

Additional Key Words and Phrases: Secure Outsourcing, Secure Delegation, Multiple Modular Exponentiations, Discrete Logarithms, RSA Cryptography, Group Theory

1 INTRODUCTION

Server-aided cryptography is an active research area addressing the problem of clients delegating or outsourcing cryptographic computations to computationally more powerful servers. Batch cryptography is another active research area addressing the problem of performing multiple cryptographic operations faster than independently repeating each operation. Currently, both areas are seeing a renewed interest because of the increasing popularity of the cloud computing computation paradigm. The problem studied in this paper lies in the intersection of these two

Authors' addresses: Giovanni Di Crescenzo, Perspecta Labs Inc., Basking Ridge, NJ, USA, gdicrescenzo@perspectalabs.com; Matluba Khodjaeva, CUNY John Jay College of Criminal Justice. New York, NY, USA, mkhodjaeva@jjay.cuny.edu; Delaram Kahrobaei, University of York, UK, New York University (USA), delaram.kahrobaei@york.ac.uk,dk2572@nyu.edu; Vladimir Shpilrain, CUNY Graduate Center. New York, NY, USA, shpil@groups.sci.ccny.cuny.edu.

areas. Computing group exponentiation has been object of study in both areas, since this operation is a cornerstone of many cryptographic protocols.

Server-aided cryptography: state of the art summary. The first formal model for delegation of cryptographic operations was introduced in [18] (but see references therein for earlier areas related to server-aided cryptography and earlier attempts to delegate group exponentiation), Overall, literature results in this area can be characterized depending on what functions are being delegated (the most popular being either group exponentiation, starting with [18], or an arbitrary polynomial-time computation, starting with [17]), on how many servers are considered (a single logical server generally being preferred to many), and on the assumption on server behavior (a malicious server being a more desirable and challenging assumption than that of a semi-honest server). While theoretical solutions exist for arbitrary polynomial-time computations, it is unclear if their claimed asymptotic efficiency has any hope of becoming practical. Known efficient solutions for delegating a single group exponentiation either work with a semi-honest server (see, e.g., [11] and references therein), or bound the probability that a malicious server successfully cheats by a constant (see, e.g., [8, 13] and references therein). Our very recent result [15] shows a way to provably and efficiently reduce this probability to exponentially small, but only works for prime-order groups.

In this paper we study the problem of a client delegating *multiple* group exponentiations to a possibly malicious server while performing less work than by a non-delegated computation of all group exponentiations, for groups typically encountered in cryptography (i.e., discrete logarithm and RSA groups).

Batch cryptography: state of the art summary. First discussed in [16], batch cryptography problems are concerned with performing multiple cryptographic protocol executions at lower costs than by independent executions. Literature results can be characterized as focusing on (non-interactively) verifying multiple outputs of a function, the most studied being functions related to group exponentiation, as well as on applications domains like operations (i.e., generation, verification) within encryption or signature schemes. In 1989, Fiat [16] first discussed batch cryptography problems, specifically targeting a batch cryptography solution for a variant of RSA. In 1994, Naccache et. al. [23] introduced a batch verifier for DSA signature. In 1995, Yen et. al. [30] improved the test of batch verification of DSA and RSA signatures by using a small exponent set of the type $\{0, 1\}^{\ell}$ for smaller parameter ℓ; however, the RSA batch verifier was broken by Boyd and et. al [3] in 2000. In 1998, Bellare et. al. [1] showed a formal model for batch verification and proposed three algorithms: (1) Random Subset Test (RST), (2) Small Exponent Test (SET), and (3) Bucket Test (BT), by combining the ideas from [23, 30]. Variants of SET have been studied in [9], where Cheon et. al. introduced two algorithms: (1) Sparse Exponent Test (SPET) and (2) Complex Exponent Test (CET), and claimed that their experiments show that SPET and CET are at least twice faster than SET. In 2015, Cheon et al. [10] proposed improved version of SPET and applied them to modified DSA and ECDSA signatures. In 2017, Hwang et al. proposed in [20] a simplified variant of SET that can provably provide batch verification of n instances using only $n - 1$ randomized exponents. In all of these batch verification tests, it is assumed that the alleged exponentiations belong to the group.

In 2000, Boyd et. al. [3] showed several attacks on batch verification tests, mostly on algorithms involving SET when they are used on non prime order subgroups. In addition to that, they try to repair those algorithms by using a loose verification test instead of a strict verification test (e.g., if a strict verification test checks that $y = g^x$ then a loose verification test can check whether $y = \pm g^x$); but note that with a loose verification test, one cannot guarantee a unique computation result for the underlying cryptographic function. In [26], Peng et al. discussed the importance of group membership tests and showed weaknesses of the tests used in papers [1, 19] due to their lack of a

membership test. In addition to that, the authors studied the group membership problem in the context of zero knowledge proofs in the equality of logarithms of common base.

In this paper we show that using delegation these tests can be efficiently transformed into tests that work for any alleged exponentiations (even values not belonging to the group or just arbitrary values). More generally, we use batch verifiers for group exponentiation as a solution tool to verify outputs returned by a (possibly malicious) server in batch delegation of group exponentiation. We show novel extensions of the Small Exponents test to efficiently, privately and securely verify exponentiations over Discrete Log and RSA groups by a malicious server.

Our contribution and techniques. We formally define client-server protocols for the batch delegation of group exponentiation that satisfies desirable requirements like efficiency (mainly in terms of client computation, but also server computation and when relevant, off-line client computation), correctness, privacy (of client's inputs) and security (against a malicious server trying to convince a client to accept an incorrect computation). We design and prove solutions for the following exponentiation functions and relevant groups, where p, q are primes:

(1) $F_{p,q,g}(x) = y$, where $x \in \mathbb{Z}_q$, $y = g^x \mod p$, and g is a generator of the q-order subgroup of $\mathbb{Z}_p^*$, $p = 2q + 1$,
(2) $F_{n,e}(x) = y$, where $x \in \mathbb{Z}_n^*$, $y = x^e \mod n$, $n = pq$, $p = 2p_1 + 1$, $q = 2q_1 + 1$, and p_1, q_1 are primes.

Note that these group settings are among the most recommended for many cryptographic algorithms and protocols based on the hardness of computing discrete logarithms and inverting the RSA function, or related problems. As a direct consequence, we obtain efficient, correct, private and secure batch delegation protocols for algorithms that are part of widely used cryptosystems, such as RSA encryption with large exponents and key contribution in Diffie-Hellman key agreement protocols, etc.

The starting point in our approach is the following protocol (for simplicity, not yet targeting privacy of the inputs):

(1) client C sends the m inputs to server S
(2) S sends m (alleged) exponentiations to C
(3) C checks that they were correctly computed using a Small Exponents Test (as in [1, 3, 21, 23]).

This protocol does not meets our requirements because of the mentioned limitations to the Small Exponent Tests, which limit the validity of the above protocol only when the alleged exponentiation values sent by S are guaranteed to be group members (raising a challenge for our Discrete Log groups), and only when these values belong to a prime-order group (raising a challenge for RSA groups). We overcome these challenges by providing an efficiently verifiable membership proof for the discrete log case (C only needs 1 additional multiplication per batch element) and a fix for the RSA case based on an efficiently verifiable proof that the exponentiation values provided by S do not contain low-order elements (here, C only needs 2 additional multiplications per batch element). Our resulting protocols allow a client to delegate the computation of multiple exponentiations in these groups to a single (possibly malicious) server, more efficiently than by computing them without any delegation. For each of the two considered groups, we present two protocols: a protocol with no offline phase or input privacy, and a protocol which achieves input privacy using an offline phase. We provide theoretical analysis showing that in all 4 protocols the client's online running time is about one order of magnitude faster than non-delegated computation. We also show performance results of our software implementation of these protocols that essentially confirms the theoretical analysis on a commodity platform and with no particular need for software optimizations.

This paper can be considered an improved version of [14], specifically contributing 4 main improvements. First, we have slightly improved the client's online efficiency of our 2 main protocols

(with off-line phase and input privacy) by moving some client computations from the online phase to the offline phase. Second, the security of our protocols for the exponentiation function $F_{n,e}$ is now proved differently, and without any computational assumption, thus holding even against computationally unlimited malicious servers (while previously we had only proved this property assuming that the server cannot factor n). Third, we have detailed our main protocol for the exponentiation function $F_{n,e}$ and given a sketch of proof of its correctness, efficiency, privacy and security properties (previously, this protocol was only briefly mentioned). Fourth, we have produced a software implementation of our 4 protocols and reported on their performance.

2 DEFINITIONS AND PRELIMINARIES

In this section we formally define batch delegation protocols for multiple group exponentiations (with their correctness, security, privacy and efficiency requirements) and review the notion of batch verification of multiple group exponentiations, as well as the main algorithm in the area: the small exponents verification test. We start with some basic notations.

Basic notations. The expression $z \leftarrow T$ denotes randomly and independently choosing z from a set T. By $z \leftarrow A(x_1, x_2, \ldots)$ we denote running the (probabilistic) algorithm A on input $x_1, x_2, \ldots$ and any random coins, and returning z as output. By $(z, tr) \leftarrow (A(x_1, x_2, \ldots), B(v_1, v_2, \ldots))$ we denote running the (probabilistic) interactive protocol between A, with input $x_1, x_2, \ldots$ and any random coins, and B, with input $v_1, v_2, \ldots$ and any random coins, where tr denotes $A's$ and B's messages in this execution, and z is A's final output.

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 thus clients are interested in delegating the computation of expensive functions to servers. In all our solutions, we consider a single client, denoted by C, and a single server, denoted by S. We ignore communication attacks as such attacks can be separately addressed using known security techniques. As in previous work in the area, we consider an offline phase, where exponentiations to random exponents can be precomputed by the client, or made somehow available to the client by its deploying entity. (Some previous work in the area mentions the pseudo-random power generator from [4, 24] as a potential source efficiently generating such exponentiations; we note that recent attacks may force parameter settings on this generator that nullify the supposed performance gains.)

Let σ denote the computational security parameter (derived from hardness considerations of the underlying computational problem and used to set public parameter lengths), and let λ denote the statistical security parameter (defined so that statistical test failure events with probability $2^{-\lambda}$ are extremely rare). Both parameters are expressed in unary notation (i.e., $1^\sigma, 1^\lambda$). We think of σ as being larger than λ. Let $F : Dom(F) \rightarrow CoDom(F)$ denote a function, where $Dom(F)$ is F's domain, $CoDom(F)$ is F's co-domain, and let $desc(F)$ denote F's description. Assuming $1^\sigma, 1^\lambda, desc(F)$ are known to both C and S, we define a *client-server protocol for the delegated (m-instance) computation of F* as the execution:

(1) $pp \leftarrow \text{Offline}(1^\sigma, 1^\lambda, desc(F))$,
(2) $((z_1, \ldots, z_m), tr) \leftarrow (C(pp, x_1, \ldots, x_m), S)$.

Step 1, if present, is executed in an *offline phase*, when no input to the function F is available yet, and, if needed by the application, could also be executed by a third party, not colluding with S. Step 2 is executed in the *online phase*, when the inputs $x_1, \ldots, x_m$ to the function F are available to C. At the end of both phases, C learns $z_1, \ldots, z_m$ (where z_i is intended to be equal to $y_i = F(x_i)$, for $i = 1, \ldots, m$), and tr is the transcript of the communication exchanged between C and S. (We will often omit $desc(F), 1^\sigma, 1^\lambda, tr$ for brevity.)

Correctness Requirement. Informally, if both parties follow the protocol, C obtains some output at the end of the protocol, and this output is, with high probability, equal to the m-tuple of values obtained by evaluating function F on C's m inputs.

DEFINITION 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* m-*instance computation of* F. *We say that* (C, S) *satisfies* δ_c-*correctness if for any* $x_1, \ldots, x_m$ *in* F's *domain,*

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

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

1. $pp \leftarrow \text{Offline}(desc(F))$
2. $((z_1, \ldots, z_m), tr) \leftarrow (C(pp, x_1, \ldots, x_m), S)$
3. if $z_i = F(x_i)$ for all $i = 1, \ldots, m$, then **return:** 1
       else **return:** 0

Security Requirement. Informally, a malicious adversary corrupting S and choosing C's inputs $x_1, \ldots, x_m$, cannot convince C to obtain, at the end of the protocol, some output z_i' different from the value y_i that is obtained by evaluating function F on C's input x_i, for some $i \in \{1, \ldots, m\}$.

DEFINITION 2. *Let* σ, λ *be the security parameters, let* F *be a function, and let* (C, S) *be a 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,

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

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

1. $pp \leftarrow \text{Offline}(desc(F))$
2. $(x_1, \ldots, x_m, aux) \leftarrow A(desc(F))$
3. $((z_1, \ldots, z_m), tr) \leftarrow (C(pp, x_1, \ldots, x_m), A(aux))$
4. if $z_i \in \{\bot, F(x_i)\}$ for all $i = 1, \ldots, m$, then **return:** 0
       else **return:** 1.

Privacy Requirement. Informally, if C follows the protocol, a malicious adversary corrupting S cannot obtain any information about C's inputs $x_1, \ldots, x_m$ from a protocol execution. This is formalized by extending the indistinguishability-based approach typically used in definitions for encryption schemes.

DEFINITION 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*

$$\text{Prob}\left[\,out \leftarrow \text{PrivExp}_{\text{F,A}}(1^{\sigma}, 1^{\lambda}) \,:\, out = 1\right] \leq 1/2 + \epsilon_p,$$

for some ϵ_p (intended to be 0 or very close to 0), where experiment PrivExp is:

1. $pp \leftarrow \text{Offline}(desc(F))$
2. $((x_{0,1}, \ldots, x_{0,m}), (x_{1,1}, \ldots, x_{1,m}), aux) \leftarrow A(desc(F))$
3. $b \leftarrow \{0, 1\}$
4. $((z_1, \ldots, z_m), tr) \leftarrow (C(pp, (x_{b,1}, \ldots, x_{b,m})), A(aux))$
5. $d \leftarrow A(tr, aux)$
6. if $b = d$ then **return:** 1 else **return:** 0.

Efficiency Metrics and Requirements. Let (C, S) be a client-server protocol for the delegated computation of a function F. We say that (C, S) has *efficiency parameters* $(t_F, t_P, t_C, t_S, cc, mc)$ if F can be computed (without 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, C and S exchange a total of at most mc messages, of total length at most cc. We only consider group multiplication as atomic operation, thus neglecting lower-order operations (e.g., equality/inequality testing, additions and subtractions between group elements). While we try to minimize all these protocol efficiency metrics, our main goal is to design protocols where $t_C << t_F$, and t_S is not significantly larger than t_F. By $t_{exp}(\ell)$ we denote the max number of group multiplications used to compute an exponentiation of a group element to an ℓ-bit exponent. By $t_{m,exp}(\ell)$ we denote the max number of group multiplications used to compute m exponentiations of the same group element (also called *fixed-base exponentiations*) to m arbitrary ℓ-bit exponents. By $t_{prod,m,exp}(\ell)$ we denote the max number of group multiplications used to compute a product of m exponentiations of (possibly different) group elements to m arbitrary ℓ-bit exponents. Finally, by $t_{inv}(\ell)$ we denote the max number of group multiplications used to compute the inverse of a group element.

Batch Verification Tests. A *batch verification test* for a function F is an efficient algorithm V that, on input $x_1, \ldots, x_m \in Dom(F)$, and $z_1, \ldots, z_m$, returns a bit b (where $b = 1$ denotes that V believes that $z_i = F(x_i)$, for $i = 1, \ldots, m$, and $b = 0$ denotes the complement event). We say that the batch verification test V satisfies *correctness* if V returns 1 when $z_i = F(x_i)$, for $i = 1, \ldots, m$. Informally, soundness requirements for V state that V returns 1 with small probability if at least one of the z_i values is different than the output of F on input x_i. Let G be a group. More formally, we say that V satisfies δ_s-*soundness (against a partially honest adversary)* if for any $z_1, \ldots, z_m \in G$, V returns 1 with probability $\leq \delta_s$ if $z_i \neq F(x_i)$, for at least one $i \in \{1, \ldots, m\}$.

In [1], based on [23, 30], the authors present a test, called the 'small exponents batch verification test' for the function computing (fixed-base) exponentiation in any prime-order group (i.e., $F_{q,g}(x) = g^x$, where g has order q and q is a prime). Their test satisfies correctness and δ_s-soundess against a partially honest adversary, for $\delta_s = 2^{-\lambda}$, and any statistical security parameter λ. The test, defined to verify that $y_i = g^{x_i}$, for $i = 1, \ldots, m$, goes as follows:

(1) randomly choose $s_1, \ldots, s_n \in \{0, 1\}^\lambda$
(2) compute $x = \sum_{i=1}^n x_i s_i \mod q$ and $y = \prod_{i=1}^n y_i^{s_i}$
(3) if $g^x = y$ then return: 1 else return: 0

We stress that [1] only proves the soundness of this test in the particular case where $y_1, \ldots, y_m$ belong to group G. In our model, the server can be malicious and may sent y_i's that are not in the group, and membership in the latter may not be more efficiently testable than performing one exponentiation (as it seemed the case in groups frequently used in discrete logarithm cryptosystems). Moreover, their test was only defined for prime-order groups, thus not directly applying to groups in cryptosystems related to RSA. In [3] it has been observed that this test can be defined for the RSA group $\mathbb{Z}_n^*$, and even satisfies correctness, but does not satisfy soundness.

In [1] the authors also present two additional verification tests for multiple exponentiation: the random subsets test, and the bucket test. The random subset test is always less efficient than the small exponents and the bucket tests are. The bucket test can be abstracted as a test that repeats the small exponents test multiple times and is more efficient than the small exponents test only when the number of exponentiations to be tested is rather large. All our protocols, although described by using the small exponents test, do generalize to this scenario by using the bucket test.

3 FROM BATCH VERIFICATION TO BATCH DELEGATION OVER DISCRETE LOG TYPE GROUPS

In this section we propose two client-server protocols for secure batch delegation to a single (and possibly malicious) server of exponentiations in the q-order subgroup G_q of $\mathbb{Z}_p^*$, where $p = 2q + 1$ and p, q are large primes, a group often used in cryptosystems that base their security on the hardness of the discrete logarithm or related problems.

The two protocols can be seen as different ways to realize the high-level approach described in Section 1 for discrete log groups, resulting in protocols satisfying security and uncomparable setup, efficiency and privacy properties.

We start by providing useful number theory definitions and facts in Section 3.1. Our first protocol, where no computation is required in an off-line phase, and the client does not hide its inputs from the server, is described in Section 3.2. Our second protocol, where the client completely hides its inputs from the server, while performing some calculations in an offline phase, is described in Section 3.3.

3.1 Number Theory Definitions and Facts

Let $p = 2q + 1$, for p, q primes. Recall that $\mathbb{Z}_p^* = \{1, \ldots, p - 1\}$ and $\mathbb{Z}_p^*$ has a q-order subgroup which we denote by G_q. Since q is prime, G_q is cyclic, and we let g be a generator of G_q. Consider the function $F_{p,q,g} : \mathbb{Z}_q \longrightarrow G_q$ defined as $F_{p,q,g}(x_i) = y_i$, for $y_i = g^{x_i} \mod p$ for all $i = 1, \ldots, m$. Here, p, q, g are assumed to be part of $desc(F_{p,q,g})$.

An integer $y \in \mathbb{Z}_p^*$ is a *quadratic residue modulo p* if there exists $r \in \mathbb{Z}_p^*$ such that $r^2 = y \mod p$. An integer $y \in \mathbb{Z}_p^*$ is a *quadratic non residue modulo p* if there does not exist an $r \in \mathbb{Z}_p^*$ such that $r^2 = y \mod p$. Euler's theorem (see, e.g., [25]) says that for any odd prime p and any $a \in \mathbb{Z}_p^*$, a is a quadratic residue modulo p if and only if $a^{(p-1)/2} = 1 \mod p$. For the considered type of integers $p = 2q + 1$, for p, q primes, this implies the following

FACT 1. G_q is the set of quadratic residues in $\mathbb{Z}_p^*$.

Fact 1 also implies that $\mathbb{Z}_p^*$ can be partitioned into 2 equal-size sets: the set of quadratic residues modulo p (i.e., G_q) and the set of quadratic non residues modulo p (i.e., $\mathbb{Z}_p^* \setminus G_q$).

3.2 Batch Delegation with No Offline Phase or Input Privacy

Our first protocol for delegating multiple unknown-exponent known-base exponentiations in group $\mathbb{Z}_p^*$ satisfies the following

THEOREM 3.1. Let p, q be large primes such that $p = 2q + 1$, let σ be the computational security parameter associated with the q-order subgroup G_q of $\mathbb{Z}_p^*$, and let λ be a statistical security parameter. There exists (constructively) a client-server protocol (C, S) for batch delegation of the computation of function $F_{p,q,g}$ on m inputs, which satisfies

1. δ_c-correctness, for $\delta_c = 1$;
2. ϵ_s-security, for $\epsilon_s = 2^{-\lambda}$;
3. $(t_F, t_S, t_P, t_C, cc, mc)$-efficiency, where
   - $t_C = 2m + t_{prod,m,exp}(\lambda) + t_{exp}(\sigma)$
   - $t_F = t_{m,exp}(\sigma)$
   - $t_S = t_{m,exp}(\sigma) + m \cdot t_{exp}(\sigma)$
   - $cc = m$ elements in $\mathbb{Z}_q$ + $2m$ elements in $\mathbb{Z}_p^*$
   - $t_P = 0$ and $mc = 2$.

We note the main takeaway from the above theorem: delegation to a single (potentially malicious) server allows C to compute m exponentiations faster than by computing them without delegation, the improvement being a multiplicative factor of about $O(\sigma/\lambda)$. In what follows, we describe protocol (C, S) and prove its properties, as claimed in the theorem.

Informal description of the protocol (C, S). Our protocol's starting point is the idea discussed in the Introduction: C sends the m exponents to S, S sends m (alleged) exponentiations to C, and C checks that they were correctly computed using the small-exponents verification test from [1]. As the latter was only defined and proved correct when the values sent by S are in the group (here being G_q), we modify this idea by requiring that S provides an efficiently verifiable proof that each sent exponentiation is in G_q. The straightforward way, via Euler's theorem, to non-interactively verify that an integer is in G_q involves one exponentiation, which is too expensive for C. In our protocol, based on Fact 1, S sends a square root modulo p of the sent values, and C verifies the exponentiations by just 1 multiplication operation per value, which is efficient enough for C in our delegation model.

Formal description of the protocol (C, S).

Input to S and C: 1^σ, 1^λ, $desc(F_{p,q,g})$

Input to C: $x_1, \ldots, x_m \in \mathbb{Z}_q$

Instructions for C and S:

(1) C sends $x_1, \ldots, x_m$ to S
(2) For $i = 1, \ldots, m$,
$\quad\quad$ S computes $y_i = g^{x_i} \mod p$
$\quad\quad$ S computes $t_i = y_i^{(q+1)/2} \mod p$
$\quad$ S sends $y_1, \ldots, y_m, t_1, \ldots, t_m$ to C
(3) For $i = 1, \ldots, m$,
$\quad\quad$ if t_i or $y_i \notin \mathbb{Z}_p^*$ then C returns: $\perp$ and halts
$\quad\quad$ if $t_i^2 \neq y_i \mod p$ then C returns: $\perp$ and halts
$\quad$ C randomly chooses $s_1, \ldots, s_m \in \{0, 1\}^\lambda$
$\quad$ C computes $x = \sum_{i=1}^m x_i s_i \mod q$
$\quad$ C computes $y = \prod_{i=1}^m y_i^{s_i} \mod p$
$\quad$ if $g^x \neq y \mod p$ then C returns: $\perp$ and halts
$\quad$ C returns: $(y_1, \ldots, y_m)$.

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

- *Round complexity:* the protocol only requires one round, consisting of one message from C to S followed by one message from S to C.
- *Communication complexity:* the protocol requires the transfer of m elements of $\mathbb{Z}_q$ from C to S and $2m$ elements of $\mathbb{Z}_p^*$ from S to C.
- *Runtime complexity:* S performs m fixed-base exponentiations and m variable-base exponentiations to σ-bit exponents. C performs m exponentiations to λ-bit exponents, 1 exponentiation to a σ-bit exponent and $2m$ multiplications. Note that to check if t_i or $y_i \notin \mathbb{Z}_n$, C only needs $O(\sigma)$ time, much less than a multiplication in $\mathbb{Z}_n^*$, and we can ignore it in our calculations.

The *correctness* property follows by showing that if C and S follow the protocol, C always outputs $(y_1, \ldots, y_m)$ such that $y_i = g^{x_i} \mod p$ for all $i = 1, \ldots, m$. First, we observe that C does not halt because of the conditions 't_i or $y_i \notin \mathbb{Z}_p^*$' or '$t_i^2 \neq y_i \mod p$'. This is because for all $i = 1, \ldots, m$, S

computes y_i as $g^{x_i} \mod p$ and thus $y_i \in G_q$ and

$$t_i^2 = \left(y_i^{(q+1)/2} \right)^2 = y_i^{q+1} = g^{x_i(q+1)} \equiv g^{x_i} = y_i \mod p.$$

Then, we observe that C does not halt because of the condition '$g^x \neq y \mod p$', by the correctness of the small-exponent batch verification test.

To prove the *security* property against a malicious S, assume that S sends to C at least one value y_i such that $y_i \neq F_{p,q,g}(x_i)$ for some $i = 1, \ldots, m$. We need to compute an upper bound ϵ_s on the security probability that S convinces C to output such a y_i. We obtain that $\epsilon_s = 2^{-\lambda}$ as a consequence of the following 2 claims:

(1) if C does not halt because of the conditions 't_i or $y_i \notin \mathbb{Z}_p^*$', or '$t_i^2 = y_i \mod p$', then $y_1, \ldots, y_m \in G_q$;
(2) if $y_1, \ldots, y_m \in G_q$, the probability that C does not halt because of the condition '$g^x \neq y \mod p$' is $\leq 2^{-\lambda}$.

Claim 1 directly follows from Fact 1, stating that G_q coincides with the set of quadratic residues in $\mathbb{Z}_p^*$, and thus if at least one y_i is not in G_q, this y_i will either not belong to $\mathbb{Z}_p^*$ or not have a square root modulo p and thus C will reject because of the relevant checks.

Claim 2 directly follows by observing that since we are assuming that at least one value y_i satisfies $y_i \neq F_{p,q,g}(x_i)$, and that $y_1, \ldots, y_m \in G_q$, we can apply the soundness of the small-exponent verification test, saying that C passes the test with probability at most $2^{-\lambda}$.

3.3 Batch Delegation with Input Privacy and Offline Phase

Our second protocol for delegating multiple unknown-exponent known-base exponentiations in group $\mathbb{Z}_p^*$ satisfies the following

THEOREM 3.2. Let p, q be large primes such that $p = 2q + 1$, let σ be the computational security parameter associated with the q-order subgroup G_q of $\mathbb{Z}_p^*$, and let λ be a statistical security parameter. There exists (constructively) a client-server protocol (C, S) for batch delegation of the computation of function $F_{p,q,g}$ on m inputs, which satisfies

1. δ_c-correctness, for $\delta_c = 1$;
2. ϵ_s-security, for $\epsilon_s = 2^{-\lambda}$;
3. ϵ_p-privacy, for $\epsilon_p = 0$;
4. $(t_F, t_S, t_P, t_C, cc, mc)$-efficiency, where
   - $t_F = t_{m,exp}(\sigma)$
   - $t_C = 3m + t_{prod,m,exp}(\lambda) + t_{exp}(\sigma)$
   - $t_S = t_{m,exp}(\sigma) + m \cdot t_{exp}(\sigma)$
   - $t_P = t_{m,exp}(\sigma)$
   - $cc = m$ values in $\mathbb{Z}_q$ + $2m$ values in $\mathbb{Z}_p^*$ and $mc = 2$.

We note the main takeaway from the above theorem: in addition to all properties of our first protocol (including the $O(\sigma/\lambda)$ speedup of client computation with respect to non-delegated computation), our second protocol also satisfies privacy of the exponents, due to the use of exponentiations computed in an offline phase.

Informal description of the protocol (C, S). Our second protocol augments our first protocol via an exponent masking technique to achieve the privacy property, as follows. Instead of directly sending the input exponents $x_1, \ldots, x_m \in \mathbb{Z}_q$ to S, C first masks them with some random values $u_1, \ldots, u_m$ in $\mathbb{Z}_q$, which were precomputed by or provided to C in an offline phase along with associated exponentiations $v_1, \ldots, v_m \in G_q$. Then, the message from C to S only contains random

elements in $\mathbb{Z}_q$ and does not leak any information at all about $x_1, \ldots, x_m$. This masking does not affect correctness of the computation since after S sends the exponentiations $w_1, \ldots, w_m$ for these masked values, C can use these latter exponentiations together with values $v_1, \ldots, v_m$ to obtain exponentiations $y_1, \ldots, y_m$ for the input exponents $x_1, \ldots, x_m$.

Formal description of the protocol (C, S).

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

Input to C: $x_1, \ldots, x_m \in \mathbb{Z}_q$

Offline phase instructions:

(1) For $i = 1, \ldots, m$
   randomly choose $u_i \in \mathbb{Z}_q$ and $s_1, \ldots, s_m \in \{0, 1\}^\lambda$
   set $v_i := g^{u_i} \mod p$ and store (u_i, v_i) on C

Online phase instructions:

(1) C sets $z_i := (x_i - u_i) \mod q$ for $i = 1, \ldots, m$
   C sends $z_1, \ldots, z_m$ to S
(2) For $i = 1, \ldots, m$,
   S computes $w_i := g^{z_i} \mod p$
   S computes $t_i := w_i^{\frac{q+1}{2}} \mod p$
   S sends $w_1, \ldots, w_m, t_1, \ldots, t_m$ to C
(3) For $i = 1, \ldots, m$,
   if t_i or $w_i \notin \mathbb{Z}_p^*$ then C returns: $\perp$ and halts
   if $t_i^2 \neq w_i \mod p$ then C returns: $\perp$ and halts
   C computes $z = \sum_{i=1}^m z_i s_i \mod q$
   C computes $w = \prod_{i=1}^m w_i^{s_i} \mod p$
   if $g^z \neq w \mod p$ then C returns: $\perp$ and halts
   C sets $y_i := w_i \cdot v_i \mod p$ for all $i = 1, \ldots, m$
   C returns: $(y_1, \ldots, y_m)$.

Properties of the protocol (C, S): The *efficiency* properties are verified by protocol inspection. Round and communication complexity are the same as for our first protocol. As for the runtime complexity, we consider both the offline and the online phase. During the offline phase m fixed-base exponentiations with random σ-bit exponents are performed (these can be efficiently approximated, for instance, by C using a pseudo-random power generator like those in [4, 24] or by another server and stored on C's device, as described in all papers in the area, starting with [18]). During the online phase, S performs m fixed-base exponentiations and m variable-base exponentiations to σ-bit exponents, and C performs a product of m exponentiations to λ-bit exponents, 1 exponentiation to a σ-bit exponent and $3m$ multiplications (thus, only m more multiplications than in the first protocol). Note that C's subtractions $\mod q$ and checks of whether t_i or $w_i \notin \mathbb{Z}_p^*$ only need $O(\sigma)$ time (much less than a multiplication in $\mathbb{Z}_n^*$) and thus are, as before, ignored in our calculations.

The *correctness* property follows by showing that if C and S follow the protocol, C always outputs $(y_1, \ldots, y_m)$ such that $y_i = g^{x_i} \mod p$ for all $i = 1, \ldots, m$. Similarly as done for our first protocol, we show that C does not halt because of the conditions 't_i or $w_i \notin \mathbb{Z}_p^*$' or '$t_i^2 \neq w_i \mod p$', or '$g^z \neq w \mod p$'. Then, the correctness of C's output follows by observing that C returns, for $i = 1, \ldots, m$,

$$y_i = w_i \cdot v_i = g^{z_i} \cdot g^{u_i} = g^{x_i - u_i} \cdot g^{u_i} = g^{x_i} \mod p.$$

The *privacy* property of the protocol against a malicious S follows by observing that C's only message to S does not leak any information about the input exponents $x_1, \ldots, x_m$. This message is $(z_1, \ldots, z_m)$ where $z_i = (x_i - u_i) \mod q$, for $i = 1, \ldots, m$, and where $u_1, \ldots, u_m$ are uniformly and independently distributed in $\mathbb{Z}_q$. Intuitively, as subtraction between a group element and a random group element returns a random group element, the $z_1, \ldots, z_m$ are uniformly and independently distributed in $\mathbb{Z}_q$ and independently from $(x_1, \ldots, x_m)$. A bit more formally, by a Shannon theorem argument, any two m-tuples for $(x_1, \ldots, x_m)$ are equally likely to have been used to generate $(z_1, \ldots, z_m)$, each through exactly one $(u_1, \ldots, u_m)$ tuple.

To prove the *security* property against a malicious S, assume that S sends to C at least one value y_i such that $y_i \neq F_{p,q,g}(x_i)$ for some $i = 1, \ldots, m$. We then compute the upper bound $\epsilon_s = 2^{-\lambda}$ on the security probability that S convinces C to output such a y_i, as a consequence of the following 3 claims:

(1) if C does not halt because of the conditions 't_i or $w_i \notin \mathbb{Z}_p^*$', or '$t_i^2 = w_i \mod p$', then $w_1, \ldots, w_m \in G_q$;

(2) if $w_1, \ldots, w_m \in G_q$, the probability that C does not halt because of condition '$g^z \neq w \mod p$' is $\leq 2^{-\lambda}$;

(3) the probability that C returns a tuple $(y_1, \ldots, y_m)$ including a value y_i such that $y_i \neq F_{p,q,g}(x_i)$ is the probability that C does not halt because of condition '$g^z \neq w \mod p$'.

Claims 1 and 2 are proved as done for our first protocol. Claim 3 follows by observing that each y_i is uniquely determined by w_i and v_i, for $i = 1, \ldots, m$.

4 FROM BATCH VERIFICATION TO BATCH DELEGATION OVER RSA TYPE GROUPS

In this section we propose two client-server protocols for secure batch delegation to a single (and possibly malicious) server of exponentiations in the group $\mathbb{Z}_n^*$, where $n = pq$ and p, q are large primes of the form $p = 2p_1 + 1, q = 2q_1 + 1$, for primes p_1, q_1.

The two protocols can be seen as different ways to realize the high-level approach described in Section 1 for RSA type groups relative to large exponents, resulting in protocols with security and incomparable setup, efficiency and privacy properties. Note that RSA encryption or signature verification may already only require a small number of group multiplications in applications where a small public exponent (e.g.: 3) is chosen. However, as noted in [1], there are various application scenarios where one might want to use a larger exponent (for example, when faster decryption or signing is of interest).

We start by providing useful number theory definitions and facts in Section 4.1. Our first protocol, where the client does not hide its inputs from the server, and the client does not have to precalculate in the offline phase, is described in Section 4.2. Our second protocol, where the client completely hides its inputs from the server, and the client performs some calculations in the offline phase, is described in Section 4.3.

When compared with the protocols for discrete logarithm type groups, we note that the group $\mathbb{Z}_n^*$ is not cyclic and thus the verification tests from [1], as written, do not suffice. Thus, we study the reasons why the small exponents verification test does not suffice, and propose remedies via an efficiently verifiable proof from the server. The main novelty here is a technique to prove and efficiently verify that the values sent by the server do not differ by the intended exponentiations by a low-order integer. This requires designing the proof based on properties and a characterization of the specific group chosen. Finally, we adapt the masking of the input exponents, as used in Section 3.3 to achieve privacy, to work for $\mathbb{Z}_n^*$.

4.1 Number Theory Definitions and Facts

Basic definitions. For any prime b and an element $v \in \mathbb{Z}_p^*$, the *Legendre symbol* $(v|b)$ of v mod b is defined as equal to $+1$ if v is a quadratic residue modulo b and -1 otherwise. Using Euler's theorem, the Legendre symbol can be computed by one exponentiation modulo b.

Let $c = ab$, for primes a, b. For any integer $v \in \mathbb{Z}_c^*$, we know that v is a quadratic residue modulo c if and only if $(v|b) = 1$ for any prime b dividing c (see, e.g., [25]). For any $v \in \mathbb{Z}_c^*$, the *Jacobi symbol* $(v|c)$ of v mod c is defined as equal to $(v|a) \cdot (v|b)$. Thus, if $(v|c) = -1$ then v is a quadratic non residue modulo c, as it is a quadratic non residue modulo at least one of the primes dividing c. On the other hand, if $(v|c) = 1$ then no efficient algorithm is known to compute whether v is a quadratic residue or non residue modulo c. For any $v_1, v_2 \in \mathbb{Z}_c^*$, let $\sim_c$ be the relation defined as $v_1 \sim_c v_2 =$ the quadratic residuosity of $v_1 v_2$ mod c. We note that $\sim_c$ is an equivalence relation and the set of quadratic residues modulo c is an equivalence class for this relation; moreover, for any $v \in \mathbb{Z}_c^*$, the set of integers $\{vq \mid q \text{ is a quadratic residue} \mod c\}$ is an equivalence class for this relation. This implies the following characterization: $\mathbb{Z}_c^*$ is divided by relation $\sim_c$ into 4 equal-size equivalence classes, of which one is the class of quadratic residues modulo c, one is the class of quadratic non residues modulo c with Jacobi symbol $+1$, and the remaining 2 classes contain quadratic non-residues modulo c with Jacobi symbol -1.

Useful facts. Let p, q, p_1, q_1 be large, same-length, primes, such that $p = 2p_1 + 1$, $q = 2q_1 + 1$, and let $n = pq$. Let $\mathbb{Z}_n^*$ denote the set of integers coprime with n. Note that the order of $\mathbb{Z}_n^*$ is $\phi(n) = (p-1)(q-1) = 4p_1 q_1$. For any e such that $\gcd(e, \phi(n)) = 1$, define the function $F_{n,e} : \mathbb{Z}_n^* \to \mathbb{Z}_n^*$ as $F_{n,e}(x) = x^e \mod n$. Here, n, e are assumed to be part of $desc(F_{n,e})$. In what follows, we prove two facts that will be critical in proving security of our batch delegation protocols for the function $F_{n,e}$.

Consider the following 3 observations. First, when n has this special form, $-1 \mod n$ is a quadratic non residue modulo n with Jacobi symbol $(-1|n) = (-1|p) \cdot (-1|q) = (-1) \cdot (-1) = +1$. Second, note that by Lemma 1 of [28] every quadratic residue mod n has a square root of Jacobi symbol 1 and a square root of Jacobi symbol -1. Third, note that if r is a square root of y mod n then so is $-r$ mod n. By combining these 3 observations, we obtain that every quadratic residue mod n has 4 square roots modulo n, which can be written as $r_{-1}, r_{+1}, -r_{-1} \mod n, -r_{+1} \mod n$, and such that, as seen by a case analysis, each of them is in a different $\sim_n$ equivalence class. For integer 1, its square roots modulo n can be further characterized as $1, -1, r, -r \mod n$, for some $r \in \mathbb{Z}_n^*$ with Jacobi symbol -1, where again each of these roots is in a different $\sim_n$ equivalence class, and 1 is the root in the class of quadratic residues modulo n. Thus, any fourth root modulo n of 1 is also a square root modulo n of 1, which proves the following

FACT 2. In $\mathbb{Z}_n^*$ there are no elements of order 4.

Now, consider elements $x, y \in \mathbb{Z}_n^*$ and let $\alpha = y/x^e \mod n$. Assume α has order 2; that is, $\alpha^2 = 1 \mod n$. Recall the above characterization that square roots of 1 mod n can be written as $1, -1, r, -r \mod n$, for some $r \in \mathbb{Z}_n^*$ with Jacobi symbol -1, where each of these roots is in a different $\sim_n$ equivalence class, and 1 is the root in the class of quadratic residues modulo n. Then, $\alpha \neq 1$ and thus it is a quadratic non-residue mod n. Because $xy = \alpha x^{e+1} \mod n$, and e is odd, xy is also not a quadratic residue modulo n, which proves the following.

FACT 3. Let $\phi(n) = (p-1)(q-1)$, $x, y \in \mathbb{Z}_n^*$, and let $e \in \mathbb{Z}_{\phi(n)}$ such that $\gcd(e, \phi(n))=1$. If $\alpha = y/x^e$ mod n has order 2 then xy mod n is a quadratic non residue modulo n.

4.2 Batch Delegation with No Offline Phase or Input Privacy

Our first protocol for delegating multiple known-exponent unknown-base exponentiations in group $\mathbb{Z}_n^*$ satisfies the following

THEOREM 4.1. *Let p, q, p_1, q_1 be large, same-length, primes, such that $p = 2p_1 + 1$, $q = 2q_1 + 1$, and let $n = pq$; let σ be the computational security parameter associated with the group $\mathbb{Z}_n^*$ and let λ be a statistical security parameter. There exists (constructively) a client-server protocol (C, S) for batch delegation of the computation of function $F_{n,e}$ on m inputs, which satisfies*

1. *δ_c-correctness, for $\delta_c = 1$;*
2. *ϵ_s-security, for $\epsilon_s = 2^{-\lambda}$;*
3. *$(t_F, t_S, t_P, t_C, cc, mc)$-efficiency, where*
   - $t_C = 2m + 2t_{prod,m,exp}(\lambda) + t_{exp}(\sigma)$
   - $t_F = m \cdot t_{exp}(\sigma)$
   - $t_S = 2m \cdot t_{exp}(\sigma)$
   - $cc = 3m$ *elements in $\mathbb{Z}_n^*$*
   - $t_P = 0$ *and* $mc = 2$.

As with Theorem 3.1, the main takeaway from the above theorem is that delegation to a single (potentially malicious) server allows C to compute m exponentiations faster than by computing them without delegation, the improvement being a multiplicative factor of $O(\sigma/\lambda)$. In what follows, we informally and formally describe our protocol (C, S) and prove its properties, as claimed in the theorem.

Informal description of the protocol (C, S). As in Section 3, our protocol's starting point is the idea discussed in the Introduction: C sends the m exponents to S, S sends m elements to C, and C checks that these elements were correctly computed using the small-exponent verification test from [1]. In Section 3, this idea fell short because this test was only defined and proved sound when the elements sent by S are in the (cyclic) group, and we dealt with this problem by designing an efficiently verifiable proof that each exponentiation sent by S is in the subgroup G_q of $\mathbb{Z}_p^*$. In this section, our group, $\mathbb{Z}_n^*$, is neither cyclic nor it has prime order, and none of the tests in [1] directly applies. In our effort to adapt the small-exponents verification test from [1], the main problem we encounter (also discussed in [3]) is related to low-order group elements that, when used by S in his answer, may significantly reduce the success probability of the test. Accordingly, we design an efficiently verifiable proof that no low-order group elements were used by S in his answer, which we prove to suffice based on (a suitable variant of) the previous analysis for the soundness of the small-exponent verification test, as well as number-theoretic facts established in Section 4.1. Our main novel technical contribution is a careful analysis of the structure of $\mathbb{Z}_n^*$, where $n = pq$, $p = 2p_1 + 1$, $q = 2q_1 + 1$, for p, q, p_1, q_1 primes, and a non-trivial use of a proof of quadratic residuosity as a proof that low-order group elements either do not exist or are not contained among the elements sent by S.

Formal description of the protocol (C, S).

Input to S and C: 1^σ, 1^λ, $desc(F_{n,e})$
Input to C: $x_1, \ldots, x_m \in \mathbb{Z}_n^*$
Instructions for C and S:

(1) C sends $x_1, \ldots, x_m$ to S
(2) For $i = 1, \ldots, m$,
    S computes $y_i = x_i^e \mod n$
    S computes $t_i = x_i^{(e+1)/2} \mod n$

S sends $y_1, \ldots, y_m, t_1, \ldots, t_m$ to C

(3) For $i = 1, \ldots, m$,

    if t_i or $y_i \notin \mathbb{Z}_n$ then C returns: $\perp$ and halts

    if $t_i^2 \neq x_i y_i \mod n$ then C returns: $\perp$ and halts

    C randomly chooses $s_1, \ldots, s_m \in \{1, \ldots, 2^\lambda\}$

    C computes $x = \prod_{i=1}^m x_i^{s_i} \mod n$

    C computes $y = \prod_{i=1}^m y_i^{s_i} \mod n$

    if $x^e \neq y \mod n$ then C returns: $\perp$ and halts

    C returns: $(y_1, \ldots, y_m)$.

Properties of the protocol (C, S): The *efficiency* properties are verified by protocol inspection and are very similar to those of the protocol in Section 3.2. As for the *round* complexity, the protocol only requires one round, consisting of one message from C to S followed by one message from S to C. As for the *communication* complexity, the protocol requires the transfer of m elements of $\mathbb{Z}_n^*$ from C to S and $2m$ elements of $\mathbb{Z}_n^*$ from S to C. As for the *runtime* complexity, S performs $2m$ exponentiations to σ-bit exponents, C performs 2 products of m exponentiations to λ exponents, 1 exponentiation to a σ-bit exponent and $2m$ multiplications to check if $t_i^2 = x_i y_i \mod n$. Note that to check if t_i or $y_i \notin \mathbb{Z}_n$, C only needs $O(\sigma)$ time, much less than a multiplication in $\mathbb{Z}_n^*$, and we essentially ignore it in our calculations.

The *correctness* property follows by showing that if C and S follow the protocol, C always outputs $(y_1, \ldots, y_m)$ such that $y_i = x_i^e \mod n$ for all $i = 1, \ldots, m$. First, we observe that C does not halt because of the conditions 't_i or $y_i \notin \mathbb{Z}_n$' or '$t_i^2 \neq x_i y_i \mod n$'. This is because for all $i = 1, \ldots, m$, S computes y_i as $x_i^e \mod n$ and thus $y_i \in \mathbb{Z}_n^*$, and

$$t_i^2 = \left(x_i^{(e+1)/2}\right)^2 = x_i^{e+1} = x_i(x_i^{e_i}) = x_i y_i \mod n.$$

Then, we observe that C does not halt because of the condition '$x^e \neq y \mod p$', by the correctness of the small-exponent batch verification test over $\mathbb{Z}_n^*$ (see, e.g., [3]).

To prove the *security* property against a malicious S, assume that S sends to C at least one value y_i such that $y_i \neq F_{n,e}(x)$ for some $i \in \{1, \ldots, m\}$. We need to compute an upper bound ϵ_s on the security probability that S convinces C to output such a y_i. For all $i = 1, \ldots, m$, define $\alpha_i = y_i/(x_i)^e$ mod n. We obtain that $\epsilon_s = 2^{-\lambda}$, as a consequence of the following 5 claims:

(1) for all $i = 1, \ldots, m$, if $y_i \notin \mathbb{Z}_n$ then C outputs $\perp$
(2) for all $i = 1, \ldots, m$, if $y_i \in \mathbb{Z}_n \setminus \mathbb{Z}_n^*$ then C outputs $\perp$ because of the condition '$x^e \neq y \mod n$'
(3) if $y_1, \ldots, y_m \in \mathbb{Z}_n^*$, and for at least one value $j \in \{1, \ldots, m\}$, α_j has order > 4, C does not halt because of the condition '$x^e \neq y \mod n$' with probability at most $2^{-\lambda}$
(4) if $y_1, \ldots, y_m \in \mathbb{Z}_n^*$ and $\alpha_1, \ldots, \alpha_m$ have order ≤ 4, then either $\alpha_1 = \cdots = \alpha_m = 1$ or C halts because of the condition '$t_i^2 = x_i y_i \mod n$';
(5) if all of C's checks in the protocol are satified, then, except with probability $2^{-\lambda}$, it holds that $\alpha_i = 1$, and thus $y_i = (x_i)^e \mod n$, for all $i = 1, \ldots, m$.

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

Claim 2 can be proved as follows: if $y_i \in \mathbb{Z}_n \setminus \mathbb{Z}_n^*$ for some $i \in \{1, \ldots, m\}$ then y_i is a multiple of p or q over the integers. Wlog assume that y_i is a multiple of p i.e. $y_i = p * l$ for some $l \in \mathbb{Z}_n$. For any $s_i \in \{1, \ldots, 2^\lambda\}$, we have that

$$y_i^{s_i} = (p * l)^{s_i} = p * (p^{s_i-1} * l^{s_i})$$

which, together with the fact that n is a multiple of p, implies that

$$y_i^{s_i} \mod n = p * (p^{s_i-1} * l^{s_i}) - kpq = p * (p^{s_i-1} * l^{s_i} - kq),$$

for some integer k, from which we have that $y_i^{s_i} \mod n$ is also a multiple of p, thus showing that $\{y_i^{s_i} \mod n\} \in \mathbb{Z}_n \setminus \mathbb{Z}_n^*$. Finally, observe that $y = \prod_{j=1}^m y_j^{s_j} \mod n$ is also a multiple of p since $y_i^{s_i}$ is multiple of p for some $i \in \{1, \cdots, m\}$ which implies that $\{\prod_{j=1}^m y_j^{s_j} \mod n\} \in \mathbb{Z}_n \setminus \mathbb{Z}_n^*$. But on the other hand $x_j \in \mathbb{Z}_n^*$ for all $j \in \{1, \ldots, m\}$ (i.e. x_j is not multiple of p or q) then, by group closure, $x^e = \prod_{j=1}^m (x_j^{s_j})^e \mod n$ is not multiple of p or q (i.e. $x^e \in \mathbb{Z}_n^*$). Therefore the last check $x^e = y$ mod n is not satisfied and C halts the protocol, from which the claim follows.

Claim 3 follows by the soundness of the small-exponent batch verification test over $\mathbb{Z}_n^*$, as follows. First, note that the condition $x^e = y \mod n$ can be rewritten as $\prod_{i=1}^m \alpha_i^{s_i} = 1 \mod n$, using the definition of α_i and the following two facts:

(1) $x^e = (\prod_{i=1}^m (x_i)^{s_i})^e = \prod_{i=1}^m ((x_i)^e)^{s_i} \mod n$,
(2) $y = \prod_{i=1}^m (y_i)^{s_i} \mod n$.

Now assume, without loss of generality, that the $j \in \{1, \ldots, m\}$ such that α_j has order > 4 is actually $j = 1$. Recall that by Lagrange's theorem in group theory, the order of any element in $\mathbb{Z}_n^*$ is a divisor of $\phi(n) = (p-1)(q-1) = 4p_1 q_1$, for primes p_1, q_1. Therefore, the order of α_1 is at least $\min(p_1, q_1)$. Now, assume that the condition $\prod_{i=1}^m \alpha_i^{s_i} = 1 \mod n$ is satisfied for a particular tuple $(s_1, \ldots, s_m)$; this also means that

$$\alpha_1^{s_1} = \prod_{i=2}^m \alpha_i^{-s_i} \mod n. \tag{1}$$

We observe that for any $s_2, \ldots, s_m$, since α_1 has order at least $\min(p_1, q_1)$, which is much larger than 2^λ, there is at most one $s_1 \in \{1, \ldots, 2^\lambda\}$ such that Equation 1 holds. Thus, for fixed $s_2, \ldots, s_m$, the probability that this equality holds is at most $2^{-\lambda}$ since C chooses s_1 uniformly from $\{1, \ldots, 2^\lambda\}$. Hence, the probability that equality (1) holds is at most $2^{-\lambda}$ even when all of $s_2, \ldots, s_m$ are chosen independently and uniformly from $\{1, \ldots, 2^\lambda\}$, and the claim follows.

Claim 4 follows by combining Fact 2 and Fact 3 from Section 4.1. First, Fact 2 proves that when n has the special form we are considering, there are no integers of order 4 in $\mathbb{Z}_n^*$. Second, Fact 3 proves that if α_j has order 2, then $x_j y_j \mod n$ is not a quadratic residue mod n, which makes C halt because the check '$t_i^2 = x_i y_i \mod n$' cannot be satisfied. Thus, we obtain that either $\alpha_1 = \cdots = \alpha_m = 1$ or C halts because of the condition '$t_i^2 = x_i y_i \mod n$', from which the claim follows.

Claim 5 follows by combining claims 1-4. $\qquad\square$

4.3 Batch Delegation with Input Privacy and Offline Phase

Our second protocol for delegating multiple known-exponent unknown-base exponentiations in group $\mathbb{Z}_n^*$ satisfies the following

THEOREM 4.2. Let p, q, p_1, q_1 be large, same-length, primes, such that $p = 2p_1 + 1$, $q = 2q_1 + 1$; let $n = pq$, let σ be the computational security parameter associated with the group $\mathbb{Z}_n^*$, and let λ be a statistical security parameter. There exists (constructively) a client-server protocol (C, S) for batch delegation of the computation of function $F_{n,e}$ on m inputs, which satisfies

1. δ_c-correctness, for $\delta_c = 1$;
2. ϵ_s-security, for $\epsilon_s = 2^{-\lambda}$;
3. ϵ_p-privacy, for $\epsilon_p = 0$;
4. $(t_F, t_S, t_P, t_C, cc, mc)$-efficiency, where
   - $t_C = 4m + 2 \cdot t_{prod,m,exp}(\lambda) + t_{exp}(\sigma)$
   - $t_F = m \cdot t_{exp}(\sigma)$
   - $t_P = m \cdot (t_{inv}(\sigma) + t_{exp}(\sigma))$

- $t_S = 2m \cdot t_{exp}(\sigma)$
- $cc = 3m$ values in $\mathbb{Z}_n^*$, and $mc = 2$.

Protocol (C, S) is obtained by applying protocol techniques from Section 3.3 and a variant of the exponent masking technique from Section 3.3 to the protocol for $\mathbb{Z}_n^*$ in Section 4.2. Specifically, the protocol techniques for delegating multiple unknown-exponent known-base exponentiations over $\mathbb{Z}_p^*$ from Section 3.2 are adapted to delegating multiple known-exponent unknown-base exponentiations over $\mathbb{Z}_n^*$. This requires, for each input element, the computation in the offline phase of an inverse element in $\mathbb{Z}_n^*$ and, the computation of 2 additional multiplications in the online phase.

Formal description of protocol (C, S).

Input to S: 1^σ, 1^λ, $desc(F_{n,e})$

Input to C: 1^σ, 1^λ, $desc(F_{n,e})$, $x_1, \ldots, x_n \in \mathbb{Z}_n^*$,

Offline phase instructions:

(1) For $i = 1, \ldots, m$
    C randomly chooses $u_i \in \mathbb{Z}_n^*$ and $s_i \in \{1, \ldots, 2^\lambda\}$
    C sets $u_i' = u_i^{-1} \mod n$ and $v_i = (u_i')^e \mod n$

Online phase instructions:

(1) For $i = 1, \ldots, m$
    C sets $z_i := (x_i \cdot u_i) \mod n$
    C sends $z_1, \ldots, z_m$ to S
(2) For $i = 1, \ldots, m$,
    S computes $w_i = z_i^e \mod n$
    S computes $t_i = z_i^{(e+1)/2} \mod n$
    S sends $w_1, \ldots, w_m, t_1, \ldots, t_m$ to C
(3) For $i = 1, \ldots, m$,
    if t_i or $w_i \notin \mathbb{Z}_n$ then C returns: $\perp$ and halts
    if $t_i^2 \neq z_i \cdot w_i \mod n$ then C returns: $\perp$ and halts
    C computes $z = \prod_{i=1}^m z_i^{s_i} \mod n$
    C computes $w = \prod_{i=1}^m w_i^{s_i} \mod n$
    if $z^e \neq w \mod n$ then C returns: $\perp$ and halts
    C sets $y_i := w_i \cdot v_i \mod n$ for all $i = 1, \ldots, m$
    C returns: $(y_1, \ldots, y_m)$.

The proof of the properties of (C, S) are obtained as a direct combination of proofs for Theorem 4.1 and Theorem 3.2.

Properties of the protocol (C, S): The *efficiency* properties are verified by protocol inspection. Round and communication complexity are the same as for our previous protocols. As for the runtime complexity, we consider both the offline and the online phase. During the offline phase C performs m inversions of random σ-bit base, m exponentiations with fixed exponent and m inverses. During the online phase, S performs $2m$ exponentiations to σ-bit exponents, and C performs 2 products of m exponentiations to λ-bit exponents, 1 exponentiation to a σ-bit exponent and m multiplications for masking x_i (i.e. $z_i = (x_i \cdot u_i) \mod n$), $2m$ multiplications to check if $t_i^2 = x_i y_i$ mod n and m multiplications to calculate y_i (i.e. $y_i = w_i \cdot v_i \mod n$, thus total $4m$ multiplications (only $2m$ more multiplications than in the previous protocol).

The *correctness* property follows by showing that if C and S follow the protocol, C always outputs $(y_1, \ldots, y_n)$ such that $y_i = x_i^e \mod n$ for all $i = 1, \ldots, n$. Similarly as done for our previous protocol,

we show that C does not halt because of the conditions 't_i or $w_i \notin \mathbb{Z}_n$' or '$t_i^2 \neq z_i \cdot w_i \mod n$', or '$z^e \neq w \mod n$'. Then, the correctness of C's output follows by observing that, for $i = 1, \ldots, n$,

$$y_i = w_i \cdot v_i = z_i^e \cdot (u_i')^e = (x_i \cdot u_i)^e \cdot (u_i^{-1})^e = x_i^e.$$

The *privacy* property of the protocol against a malicious S follows by observing that C's only message to S does not leak any information about x_i for all $i = 1, \ldots, n$. This message is $(z_1, \ldots, z_n)$ where $z_i = x_i * u_i \mod n$, for $u_1, \ldots, u_n$ uniformly and independently distributed in $\mathbb{Z}_n^*$, and thus the $z_1, \ldots, z_n$ values are uniformly distributed in $\mathbb{Z}_n^*$ and independent from $x_1, \ldots, x_n$.

To prove the *security* property against a malicious S, we need to compute an upper bound ϵ_s on the security probability that S convinces C to output $(y_1, \ldots, y_m)$, where at least one value y_i satisfies $y_i \neq F_{n,e}(x_i)$ for some $i \in \{1, \ldots, m\}$. For all $i = 1, \ldots, m$, define $\alpha_i = w_i/(z_i)^e \mod n$. We obtain that $\epsilon_s = 2^{-\lambda}$, as a consequence of the following 6 claims:

(1) for all $i = 1, \ldots, m$, if t_i or $w_i \notin \mathbb{Z}_n$ then C outputs $\perp$

(2) for all $i = 1, \ldots, m$, if $w_i \in \mathbb{Z}_n \setminus \mathbb{Z}_n^*$ then C outputs $\perp$ because of the condition '$z^e \neq w \mod n$'

(3) if $w_1, \ldots, w_m \in \mathbb{Z}_n^*$, and for at least one value $j \in \{1, \ldots, m\}$, α_j has order > 4, C does not halt because of the condition '$z^e \neq w \mod n$' with probability at most $2^{-\lambda}$

(4) if $w_1, \ldots, w_m \in \mathbb{Z}_n^*$ and $\alpha_1, \ldots, \alpha_m$ have order ≤ 4, then either $\alpha_1 = \cdots = \alpha_m = 1$ or C halts because of the condition '$t_i^2 = z_i w_i \mod n$';

(5) if all of C's checks in the protocol are satisfied, then, except with probability $2^{-\lambda}$, it holds that $\alpha_i = 1$, and thus $w_i = (z_i)^e \mod n$, for all $i = 1, \ldots, m$;

(6) for all $i = 1, \ldots, m$, if $w_i = (z_i)^e \mod n$ then $y_i = (x_i)^e \mod n$.

Claim 1-5 are proved as done for our protocol in Section 4.2, and Claim 6 directly follows by inspection of C's instructions. □

5 PERFORMANCE ANALYSIS

In Theorems 3.1, 3.2, 4.1, and 4.2 we have expressed the performance of our protocols in terms of group multiplication operations, and parameterized by functions $t_{exp}, t_{m,exp}, t_{prod,m,exp}$. Among the different ways to express these functions in terms of group multiplication operations, here we pick the following two: a basic setting (using definitions and textbook algorithms), and an improved setting (using state-of-the-art algorithms). As a basic setting, one can obtain:

- $t_{exp}(\ell) = 2\ell$ using the square-and-multiply algorithm for modular exponentiation,
- $t_{m,exp}(\ell) = m \cdot t_{exp}(\ell)$, by definition,
- $t_{prod,m,exp}(\ell) = m \cdot t_{exp}(\ell) + m - 1$, by definition.

Using improved algorithms from the literature (specifically, we use closed-form estimates in [2] for algorithms in [5, 6, 27, 29], but see also [12, 22] for other algorithms claiming improvements without closed-form evaluations), we can define the following improved setting:

- $t_{exp}(\ell) \sim \ell(1 + 1/(\log \ell))$,
- $t_{m,exp}(\ell) \sim \ell(1 + m/(\log \ell))$,
- $t_{prod,m,exp}(\ell) \sim \ell(1 + m/(\log m + \log \ell))$.

In Table 1 we compare the performance of our Section 3 protocols under the basic setting of these functions, with a non-delegated computation under both basic and improved settings.

The main takeaway from Table 1 is that our protocols in Section 3 reduce C's online computation t_C by a multiplicative factor of about σ/λ (resp., $(\sigma \log \lambda)/(\lambda \log \sigma)$) with respect to non-delegated

Table 1. Performance of our first 2 protocols, parameterized in m, λ, σ

Metric	$F_{p,q,g}$, **with no delegation**		$F_{p,q,g}$, **with delegation, basic**	
	Basic setting	**Improved setting**	**Section 3.2**	**Section 3.3**
t_F	$2m\sigma$	$\sigma(1 + m/\log\sigma)$	$2m\sigma$	$2m\sigma$
t_P	0	0	0	$2m\sigma$
t_C	$2m\sigma$	$\sigma(1 + m/\log\sigma)$	$2\sigma + 2m\lambda + 3m$	$2\sigma + 2m\lambda + 4m$
t_S	0	0	$4m\sigma$	$4m\sigma$
ϵ_p	0	0	None	0
ϵ_s	0	0	$2^{-\lambda}$	$2^{-\lambda}$

Table 2. The number t_C of C's group multiplications in our 4 protocols, when $\sigma = 2048$ and $\lambda = 128$

m	**No Delegation**	**Section 3**		**Section 4**	
		3.2	**3.3**	**4.2**	**4.3**
2	8,192	4,614	4,616	5,128	5,130
10	40,960	6,686	6,696	9,256	9,266
100	409,600	29,996	30,096	55,696	55,796
1K	4,096K	263K	264K	520K	521K
2	2,420	2,398	2,400	2,558	2,560
10	3,909	2,506	2,516	2,758	2,768
100	20,666	3,500	3,600	4,566	4,666
1K	188K	11,906	12,906	19,578	20,578

computation, when using the basic (resp., improved) setting of the t functions. Given currently recommended parameter settings (i.e., $\sigma = 2048$ and $\lambda = 128$), our protocols for $F_{p,q,g}$ reduce C's online computation by essentially 1 order of magnitude. It also makes sense to compare our protocol in Section 3.3 with a repeated execution, once for each exponentiation, of our recent protocol in [15] for a single exponentiation in G_q. The two approaches are incomparable in that our protocol in this paper has better off-line complexity, while the repeated execution of the protocol in [15] would have better client runtime in the case of a small number of modular exponentiations.

Protocols in Section 4 can be used to generate a similar table, the only difference being in $t_C = 2\sigma + 4m\lambda + 4m$ for the protocol in Section 4.2 and $t_C = 2\sigma + 4m\lambda + 5m$ for the protocol in Section 4.3. Thus, the reduction of t_C with respect to non-delegated computation achieved for $F_{n,e}$ is essentially the same (only a factor of 2 smaller) as for $F_{p,q,g}$.

By setting $\sigma = 2048$ and $\lambda = 128$ (currently recommended parameter settings for cryptographic applications), we obtain and capture in Table 2 concrete numbers for t_C (i.e., C's group multiplications in the online phase), with respect to both basic (rows 1-4) and improved (rows 5-8) settings for the t functions. On protocols from Section 3, we note an improvement of half to slightly more than 1 order of magnitude, depending on the value of m. On protocols from Section 4, we note similar improvements, although starting from $m \geq 10$.

We implemented our protocols in Sections 3.2, 3.3, 4.2 and 4.3 for the multiplicative group $(\mathbb{Z}_p^*, \cdot \mod p)$, for $p = 2q + 1$, and p, q are large primes such that $\sigma \in \{1024, 2048\}$ and the security parameter $\lambda = 128$ (i.e. $\epsilon_s = 2^{-128}$). 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. Performance data measured

Table 3. Performance data (in sec.) for protocol in Section 3.2, $\sigma = 2048$

m	2		10		100		1000	
	SM	NoSM	SM	NoSM	SM	NoSM	SM	NoSM
t_F	3.097E-02	7.610E-03	1.555E-01	3.667E-02	1.554E+00	3.732E-01	1.652E+01	3.698E+00
t_C	1.755E-02	4.435E-03	2.537E-02	6.317E-03	1.124E-01	3.128E-02	1.041E+00	2.665E-01
t_S	6.218E-02	1.536E-02	3.108E-01	7.282E-02	3.123E+00	7.397E-01	3.308E+01	7.431E+00
$\frac{t_F}{t_C}$	1.765E+00	1.716E+00	6.132E+00	5.805E+00	1.383E+01	1.193E+01	1.587E+01	1.388E+01

Table 4. Performance data (in sec.) for protocol in Section 3.3, $\sigma = 2048$

m	2		10		100		1000	
	SM	NoSM	SM	NoSM	SM	NoSM	SM	NoSM
t_F	3.596E-02	7.056E-03	1.625E-01	3.616E-02	1.685E+00	3.748E-01	1.611E+01	3.742E+00
t_P	3.976E-02	7.108E-03	1.632E-01	3.639E-02	1.660E+00	3.737E-01	1.602E+01	3.768E+00
t_C	2.075E-02	4.101E-03	2.632E-02	6.314E-03	1.197E-01	3.127E-02	1.010E+00	2.813E-01
t_S	7.464E-02	1.412E-02	3.253E-01	7.240E-02	3.333E+00	7.575E-01	3.217E+01	7.493E+00
$\frac{t_F}{t_C}$	1.732E+00	1.721E+00	6.173E+00	5.727E+00	1.408E+01	1.199E+01	1.595E+01	1.330E+01

Table 5. Performance data (in sec.) for protocol in Section 4.2, $\sigma = 2048$

m	2		10		100		1000	
	SM	NoSM	SM	NoSM	SM	NoSM	SM	NoSM
t_F	3.589E-02	7.245E-03	2.014E-01	3.536E-02	1.918E+00	3.543E-01	1.711E+01	3.542E+00
t_C	2.264E-02	4.651E-03	4.451E-02	8.571E-03	2.491E-01	5.343E-02	2.070E+00	5.030E-01
t_S	7.278E-02	1.462E-02	4.062E-01	7.063E-02	3.834E+00	7.050E-01	3.415E+01	7.106E+00
$\frac{t_F}{t_C}$	1.585E+00	1.558E+00	4.525E+00	4.125E+00	7.701E+00	6.631E+00	8.266E+00	7.042E+00

for all four protocols from Sections 3.2, 3.3, 4.2 and 4.3 is grouped in four tables. In each table we report performance data for one of our protocols, by measuring running times t_F, t_C, t_S, and t_P (if it is applicable in the protocol), and improvement ratio t_F/t_C, for different values of parameter m (i.e., $m = 2, 10, 100, 1000$), and using both an implementation of the modular exponentiation based on the textbook square-and-multiply algorithm (in column labelled 'SM') and the Python built-in function gmpy2.powmod (in column labelled 'NoSM'). Our empirical results essentially confirm the theoretical analysis, showing that our delegation protocols reduce C's online computation by about 1 order of magnitude.

6 CONCLUSIONS

We studied the problem of a client efficiently, privately and securely delegating the computation of multiple group exponentiations to a more computationally powerful server (i.e., a cloud server). We presented the first practical and provable solutions to this batch delegation problem for groups commonly used in cryptography, based on discrete logarithm and RSA hardness assumptions. Our results directly solve batch delegation of various algorithms in widely used cryptosystems, including RSA encryption with large exponents and key contribution in Diffie-Hellman key agreement protocols. Open problems specific to our results include designing efficient, secure and private protocols that do not require an off-line phase. Open research directions related to this area include

Table 6. Performance data (in sec.) for protocol in Section 4.3, $\sigma = 2048$

m	2		10		100		1000	
	SM	NoSM	SM	NoSM	SM	NoSM	SM	NoSM
t_F	3.428E-02	7.087E-03	1.702E-01	3.559E-02	1.572E+00	3.542E-01	1.585E+01	3.633E+00
t_P	3.431E-02	7.244E-03	1.706E-01	3.604E-02	1.578E+00	3.601E-01	1.576E+01	3.684E+00
t_C	2.140E-02	4.566E-03	3.753E-02	8.834E-03	2.055E-01	5.458E-02	1.935E+00	5.280E-01
t_S	6.835E-02	1.413E-02	3.404E-01	7.090E-02	3.152E+00	7.095E-01	3.556E+01	7.269E+00
$\frac{t_F}{t_C}$	1.602E+00	1.552E+00	4.535E+00	4.028E+00	7.650E+00	6.489E+00	8.193E+00	6.880E+00

the batch delegation of operations in elliptic curve groups or lattices used in cryptography, or of other cryptographic algorithms and protocols.

REFERENCES

[1] M. Bellare, J. Garay, and T. Rabin, *Fast batch verification for modular exponentiation and digital signatures*, in Proc. of Eurocrypt 98, pp. 236-250, Springer, 1998

[2] D. J. Bernstein, *Pippenger's exponentiation algorithm*, http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.25.786, 2002

[3] C. Boyd and C. Pavlovski, *Attacking and repairing batch verification schemes*, in Proc. of Asiacrypt 00, pp. 58-71, Springer, 2000.

[4] V. Boyko, M. Peinado and R. Venkatesan, *Speeding up discrete log and factoring based schemes via precomputations*, in Proc. of Eurocrypt 98, pp. 221-235, Springer, 1998.

[5] A. Brauer, *On addition chains*, in Bulletin of the American Mathematical Society. 45 (10): 736-739, 1939.

[6] E. Brickell, D. Gordon, K. McCurley, and D. Wilson, *Fast exponentiation with precomputation: algorithms and lower bounds*, in Proc. of Eurocrypt '92, pp. 200-207, Springer, 1992.

[7] J. Camenisch, S. Hohenberger, and M. Pedersen, *Batch verification of short signatures*, in Proc. of Eurocrypt 2007, pp. 246-263, Springer.

[8] B. Cavallo, G. Di Crescenzo, D. Kahrobaei, and V. Shpilrain, *Efficient and secure delegation of group exponentiation to a single server*, in Proc. of RFIDSec 15, pp. 156-173, Springer, 2015.

[9] J. Cheon and D. Lee *Use of sparse and/or complex exponents in batch verification of exponentiations*, in IEEE Transactions on Computers, vol. 55, n. 12, pp 1536-1542, 2006.

[10] J. Cheon and M. Lee *Improved batch verification of signatures using generalized sparse exponents*, in Computer Standards & Interfaces, vol. 40, pp 42-52, Elsevier, 2015.

[11] C. Chevalier, F. Laguillaumie, D. Vergnaud, *Privately Outsourcing Exponentiation to a Single Server: Cryptanalysis and Optimal Constructions*, in Proc. of ESORICS 16, pp. 261-278, Springer, 2016.

[12] B. Cubaleska, A. Rieke and T. Hermann, *Improving and extending the Lim/Lee exponentiation algorithm*, in Proc. of Selected Areas in Cryptography 99, pp.163-174, Springer, 1999.

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

[14] G. Di Crescenzo, M. Khodjaeva, D. Kahrobaei, V. Shpilrain, *Computing multiple exponentiations in discrete log and RSA groups: From batch verification to batch delegation*, in Proc. of IEEE CNS 2017, pp. 531-539, 2017.

[15] G. Di Crescenzo, M. Khodjaeva, D. Kahrobaei, V. Shpilrain, *Practical and Secure Outsourcing of Discrete Log Group Exponentiation to a Single Malicious Server*, in Proc. of ACM CCSW 2017, pp. 17-28, 2017.

[16] A. Fiat, *Batch RSA,* in CRYPTO'89, vol 435, n. 2, pp. 175-185, 1989.

[17] R. Gennaro, C. Gentry, and B. Parno, *Non-interactive verifiable computing: Outsourcing computation to untrusted workers,* in Proc. of Crypto 10, pp. 465-482, Springer, 2010.

[18] S. Hohenberger and A. Lysyanskaya, *How to securely outsource cryptographic computations*, in Proc. of TCC 2015, pp. 264-282, Springer.

[19] F. Hoshino, M. Abe, and T. Kobayashi *Lenient/strict batch verification in several groups*, in ISC'01, pp 81-94, Springer, 2001.

[20] J. Hwang, B. Song, D. Choi, S. Jin, H. Cho, M. Lee, *Simplified small exponent test for batch verification*, in Theoretical Computer Science, vol. 622, pp 48-58, Elsevier, 2017.

[21] C. H. Lim and P. J. Lee, *Security of interactive DSA batch verification*, In: Electronics letters, 30 (19): 1592-1593, IET, 1994.

[22] B. Möller, *Improved techniques for fast exponentiation*, in Proc. of ICISC 02, pp. 298-312, Springer, 2002.

[23] D. Naccache, D. M'Raïhi, S. Vaudenay, and D. Raphaeli, *Can DSA be improved? Complexity trade-offs with the digital signature standard*, in Proc. of Eurocrypt 1994, pp. 77-85, Springer.

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

[25] I. Niven, H. Zuckerman, and H. Montgomery, *An introduction to the theory of numbers*, John Wiley & Sons, 2008.

[26] K. Peng, C. Boyd, and E. Dawson, *Batch Zero-Knowledge Proof and Verification and Its Applications*, in ACM Transactions on Information and System Security, vol. 10, n. 2, pp. 1-28, 2007.

[27] N. Pippenger, *On the Evaluation of Powers and Monomials*, in SIAM J. Comput. 9(2): 230-250, 1980.

[28] J. Van De Graaf and R. Peralta, *A simple and secure way to show the validity of your public key*, in Proc. of CRYPTO 87, pp.128-134, Springer, 1987.

[29] A. Yao, *On the Evaluation of Powers*, in SIAM J. Comput. 5(1): 100-103, 1976.

[30] S. Yen and C. Laih, *Improved digital signature suitable for batch verification*, in IEEE Trans. on Computers, 44 (7) pp. 957-959, IEEE, 1995.