Computing Multiple Exponentiations in Discrete Log and RSA Groups: From Batch Verification to Batch Delegation

Giovanni Di Crescenzo*, Matluba Khodjaeva†, Delaram Kahrobaei‡, Vladimir Shpilrain§
*Vencore Labs. Basking Ridge, NJ, USA. Email: gdicrescenzo@vencorelabs.com
†CUNY Graduate Center. New York, NY, USA. Email: mkhodjaeva@gradcenter.cuny.edu
‡CUNY Graduate Center. New York, NY, USA. Email: dkahrobaei@gc.cuny.edu
§CUNY Graduate Center. New York, NY, USA. Email: shpil@groups.sci.ccny.cuny.edu

***Abstract*—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. 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 various algorithms in cryptosystems, including RSA encryption and Diffie-Hellman key agreement protocols.**

I. 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 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 [12] (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 [12], or an arbitrary polynomial-time computation, starting with [11]), 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., [9] and references therein), or at best bound the probability that a malicious server successfully cheats by a constant (see, e.g., [8] and references therein). 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.

978-1-5386-0683-4/17/$31.00

Batch cryptography: state of the art summary. First discussed in [13], 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. A first set of formal treatment and solutions has emerged from work in [14], [16], [1], [3], [7]. The state-of-the-art batch verifier for group exponentiation, named Small Exponents test in [1], has been studied in [14], [16], [1], [3]. Although being of much interest because of high efficiency gains (it reduces m exponentiations to arbitrary exponents to one exponentiation plus about m exponentiations to much shorter exponents), its early versions came with limitations with respect to the class of applicable groups (i.e., prime-order groups, thus not including RSA), and to the distribution of the exponentiation values to be checked (assumed to already belong to the group, which raises a challenge for often used Discrete Log groups). Variations have been discussed (see, e.g., [3]) that either augment the test to remove the limitation on value distribution at the cost of an expensive membership check, or modify the exponentiation function to one for which the Small Exponents test works. In this paper 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 convincing 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 Diffie-Hellman key agreement, El-Gamal encryption, Cramer-Shoup encryption, RSA encryption or signature verification (with large exponents), 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 [14], [16], [1], [3]).

This protocol does not meets our requirements because of the mentioned limitations on the distribution of the exponentiation values (raising a challenge for our Discrete Log groups), and because it only applies to prime-order groups (raising a challenge for RSA groups). We overcome these deficiencies 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 exponentiations provided by S do not contain low-order elements (here, C only needs 2 additional multiplications per batch element).

II. 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 obtained via the pseudo-random power generator from [4], [17] or made somehow available to the client by its deploying entity.

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 $z_i \in \{\perp, F(x_i)\}$ 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.

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 [22], [16], 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 the group. 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.

III. 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 I 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 III-A. 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 III-B. Our second protocol, where the client completely hides its inputs from the server, and the client performs some calculations in an offline phase, is described in Section III-C.

A. 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 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., $\mathbb{Z}_p^* \setminus G_q$).

B. Batch Delegation with No Offline Phase or Input Privacy

Our first protocol satisfies the following

Theorem 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.

Main ideas 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 us. 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 our delegation model.

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

- *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 $2^{-\lambda}$.

C. Batch Delegation with Input Privacy and Offline Phase

Our second protocol satisfies the following

Theorem 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^σ, 1^λ, $desc(F_{p,q,g})$

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

Offline phase instructions:

1) For $i = 1, \ldots, m$
   randomly choose $u_i \in \mathbb{Z}_q$
   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 randomly chooses $s_1, \ldots, s_m \in \{0,1\}^\lambda$
   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], [17] or by another server and stored on C's device, as described in all papers in the area, starting with [12]). 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 $i = 1, \ldots, m$.

IV. 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 I 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 IV-A. 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 IV-B. 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 IV-C.

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 III-C to achieve privacy, to work for $\mathbb{Z}_n^*$.

A. 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., [18]). 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 $\{vq \,|\, q \text{ is a quadratic residue } \mod c\}$ 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 $\phi(n) = (p-1)(q-1) = 4p_1q_1$. 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 [20] 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.

B. Batch Delegation with No Offline Phase or Input Privacy

Our first protocol satisfies the following

Theorem 3: 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 $\mathbb{Z}_n^*$, let λ be a statistical security parameter, and let ϵ denote a function negligible in σ. 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, unless S can factor n, for $\epsilon_s = 2^{-\lambda} + \epsilon$;
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 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 III, 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 III, 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 IV-A. 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 $s_1, \ldots, s_m \in \{0,1\}^\lambda$
   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 III-B. 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 $\epsilon_s = 2^{-\lambda} + \epsilon$, 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 $y_i \in \mathbb{Z}_n \setminus \mathbb{Z}_n^*$ with not negligible probability, then S can be used to efficiently factor 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 $2^{-\lambda} + \epsilon$, 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 follows by definition of $\mathbb{Z}_n^*$; specifically, assume, by way of contradiction, that $y_i \notin \mathbb{Z}_n \setminus \mathbb{Z}_n^*$; this implies that $\gcd(y_i, n) > 1$, and since $y_i < n$, it holds that $\gcd(y_i, n) > 1$ is a factor of n.

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 $\Pi_{i=1}^m \alpha_i^{s_i} = 1 \mod n$, using the definition of α_i and the following two facts:

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

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 $\phi(n) = (p-1)(q-1) = 4p_1q_1$, 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 $\Pi_{i=1}^m \alpha_i^{s_i} = 1 \mod n$ is satisfied for a particular tuple $(s_1, \ldots, s_m)$; this also means that

$$\alpha_1^{s_1} = \Pi_{i=2}^m \alpha_i^{-s_i} \mod n. \quad (1)$$

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 $s_1 \in \{0,1\}^\lambda$ 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 $\{0,1\}^\lambda$. 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 $\{0,1\}^\lambda$, and the claim follows.

Claim 4 follows by combining Fact 2 and Fact 3 from Section IV-A. 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. ■

C. Batch Delegation with Input Privacy and Offline Phase

By directly combining the techniques in Section IV-B and the input mixing technique from Section III-C, we obtain a protocol for delegating multiple exponentiations in $\mathbb{Z}_n^*$ that preserves the privacy of the input exponents. The proof of the following theorem is a direct combination of proofs of Theorem 3 and Theorem 2.

Theorem 4: 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 λ be a statistical security parameter, and let ϵ denote a function that is negligible in σ. 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, unless S can factor n, for $\epsilon_s = 2^{-\lambda} + \epsilon$;
3. ϵ_p-privacy, for $\epsilon_p = 0$;
4. $(t_F, t_S, t_P, t_C, cc, mc)$-efficiency, where
   - $t_C = 3m + 2 \cdot t_{prod,m,exp}(\lambda) + t_{exp}(\sigma)$
   - $t_F = m \cdot t_{exp}(\sigma)$
   - $t_P \leq t_S = 2m \cdot t_{exp}(\sigma)$
   - $cc = 3m$ values in $\mathbb{Z}_n^*$, and $mc = 2$.

Protocol (C, S) is obtained by adding a direct variant of the input mixing technique from Section III-C to the protocol for $\mathbb{Z}_n^*$ in Section IV-B, and is omitted.

V. Performance Analysis

The performance of our protocols has been expressed in terms of group multiplications, and parameterized by functions t_{exp}, $t_{m,exp}$, $t_{prod,m,exp}$. As a basic setting of these functions, one can obtain:

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

Using improved algorithms from the literature (we use closed-form estimates in [2] for algorithms in [5], [21], [19], [6], but see also [10], [15] for other algorithms claiming improvements without closed-form evaluations), one can obtain:

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

The table below compares the performance of our Section III protocols under the basic setting of these functions, with a non-delegated computation under both basic and improved settings.

Perf.	$F_{p,q,g}$, **with no delegation**		$F_{p,q,g}$, **with delegation, basic**	
	Basic	**Improved**	**Section III-B**	**Section III-C**
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	$2^{-\lambda}$	$2^{-\lambda}$

The main takeaway from above settings and table is that our protocols in Section III 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 computation, when using the basic (resp., improved) setting of the t functions.

Protocols in Section IV 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 IV-B and $t_C = 2\sigma + 4m\lambda + 5m$ for the protocol in Section IV-C. 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 the following 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.

m	No Deleg.	**Section III**		**Section IV**	
		III-B	**III-C**	**IV-B**	**IV-C**
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

On protocols from Section III, we note an improvement of half to slightly more than 1 order of magnitude, depending on the value of m. On protocols from Section IV, we note similar improvements, although starting from $m \geq 10$.

VI. 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 cryptosystems of major importance, including RSA encryption (with large exponents) and Diffie-Hellman key agreement protocols.

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

References

[1] 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://cr.yp.to/papers/pippenger.ps, 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): 736739, 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] 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.

[10] 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.

[11] 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.

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

[13] A. Fiat, *Batch RSA,* in J. of Cryptology, vol 10, n. 2, pp. 75-88, 1997.

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

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

[16] 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.

[17] 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.

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

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

[20] 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.

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

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