Research Article

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

Delegating a Product of Group Exponentiations with Application to Signature Schemes (Submission to Special NutMiC 2019 Issue of JMC)

https://doi.org/10.1515/jmc-2019-0036
Received Aug 05, 2019; accepted Aug 27, 2020

Abstract: Many public-key cryptosystems and, more generally, cryptographic protocols, use group exponentiations as important primitive operations. To expand the applicability of these solutions to computationally weaker devices, it has been advocated that a computationally weaker client (i.e., capable of performing a relatively small number of modular multiplications) delegates such primitive operations to a computationally stronger server. Important requirements for such delegation protocols include privacy of the client's input exponent and security of the client's output, in the sense of detecting, except for very small probability, any malicious server's attempt to convince the client of an incorrect exponentiation result. Only recently, efficient protocols for the delegation of a fixed-based exponentiation, over cyclic and RSA-type groups with certain properties, have been presented and proved to satisfy both requirements.
In this paper we show that a product of *many* fixed-base exponentiations, over a cyclic groups with certain properties, can be privately and securely delegated by keeping the client's online number of modular multiplications only slightly larger than in the delegation of a *single* exponentiation. We use this result to show the *first* delegations of entire cryptographic schemes: the well-known digital signature schemes by El-Gamal, Schnorr and Okamoto, over the q-order subgroup in $\mathbb{Z}_p$, for p, q primes, as well as their variants based on elliptic curves. Previous efficient delegation results were limited to the delegation of single algorithms within cryptographic schemes.

Keywords: Secure Delegation, Modular Exponentiations, Discrete Logarithms, Cryptography, Group Theory, Elliptic Curves

2020 Mathematics Subject Classification: 11T71, 94A60

1 Introduction

Delegation of cryptographic operations is an active research direction addressing the problem of computationally weaker clients delegating the most expensive cryptographic computations to computationally powerful servers. Recently, this area is seeing an increased interest because of shifts in modern computation paradigms

***Corresponding Author: Giovanni Di Crescenzo:** Perspecta Labs Inc. Basking Ridge, NJ, United States of America; Email: gdicrescenzo@perspectalabs.com
Matluba Khodjaeva: CUNY John Jay College of Criminal Justice. New York, NY, United States of America; Email: mkhodjaeva@jjay.cuny.edu
Delaram Kahrobaei: University of York. Heslington, York, United Kingdom; Email: delaram.kahrobaei@york.ac.uk
Vladimir Shpilrain: City University of New York. New York, NY, United States of America; Email: shpil@groups.sci.ccny.cuny.edu

towards cloud computing, edge computing, large-scale computations over big data, and computations with low-power devices, such as RFIDs.

The first formal model for delegation of cryptographic operations (also called outsourcing of cryptographic operations, server-aided cryptography, and server-assisted cryptography) was introduced in [31], where the authors especially studied the case of modular exponentiation, as this operation is a cornerstone of so many cryptographic schemes and protocols. In this model, we have a client, with an input x, who delegates to one or more servers the computation of a function F on the client's input, and the main desired requirements are:

1. *privacy*: only minimal or no information about x should be revealed to the server(s);
2. *security*: the server(s) should not be able, except possibly with very small probability, to convince the client to accept a result different than $F(x)$; and
3. *efficiency*: the client's computation time should be much smaller than computing $F(x)$ without delegating the computation.

Moreover, in all previous work in the area, the computational weakness of the client is really only restricted to the online phase and relatively expensive offline computation can be performed and stored on the client's device, say, at client deployment time. For instance, towards a delegated computation of modular exponentiation, it seems reasonable to assume that even computationally weaker devices are or will soon be able to perform a moderate number of less expensive operations like modular multiplications (see, e.g., recent advances in [1], showing how to practically implement group multiplication, for a specific group, and a related public-key cryptosystem, using RFID tags). This computation gap between servers and clients is significant in that with currently recommended parameter settings an unoptimized single exponentiation may require, on average, more than 2000 multiplications. Examples of computationally weaker devices provided in [31] include RFID tags, embedded devices, and, more generally, low-resource devices, but we note that the delegation problem is actually meaningful in any application domain where clients with server access would rather reduce their computation workload.

In [31], the authors studied delegation of modular exponentiation to 2 servers of which at most one was malicious, and to 1 server, who was honest on almost all inputs. Recently, we solved the open problems of private, secure and efficient delegation to a single, possibly malicious, server, of a single fixed-based exponentiation in cyclic groups [21] and a single fixed-exponent exponentiation in RSA-type groups [22]. We showed delegation for these exponentiation functions over a more general class of groups in [19], however with slightly worse efficiency or security properties than in [21, 22]. Previously, in [13], we had also showed private, secure and efficient delegation to a single, possibly malicious, server, of group inverses, without need for an offline phase.

Our Contributions. In this paper we show that a product of *many* fixed-base exponentiations, over a cyclic group with certain properties, can be privately and securely delegated to a single, possibly malicious, server, by keeping the client's online number of modular multiplications only slightly larger than that for delegating a *single* exponentiation. Our result holds for a general class of cyclic groups with efficient operation and inverse, and efficiently verifiable proofs of membership. Although not all cyclic groups are known to satisfy these properties, we show that this class includes cyclic groups often used in cryptography (i.e., the prime order multiplicative subgroup of $\mathbb{Z}_p$, for primes p of a special form, and the analogue additive group based on elliptic curves). Fixing the first of these two groups for efficiency evaluation, a product of m exponentiations in $\mathbb{Z}_p$ with σ-bit exponents can be delegated by a client that only uses less than $2\lambda+m+4$ modular multiplications in the online phase, if the probability of not detecting an incorrect result is bounded by $2^{-\lambda}$. This improves upon non-delegated computation, which would require up to $2m\sigma + m - 1$ modular multiplications, when exponentiation is performed via a square and multiply algorithm, as well as upon direct and repeated use of the delegation of a single exponentiation from [21], where the client would use up to $2m\lambda + 4m$ modular multiplication in the online phase.

We use this result to delegate the *first* cryptographic schemes: the well-known digital signature schemes by El-Gamal [27], Schnorr [41] and Okamoto [39] over the prime-order subgroup in $\mathbb{Z}_p$, as well as their variants based on elliptic curves. Previously, only primitive operations like group exponentiations or inverses were

delegated, and no complete cryptosystem was delegated to a single, possibly malicious, server. As an example of the efficiency achieved here, Okamoto's scheme normally requiring a verification of 3 exponentiations with 2048-bit exponents can now be delegated by a client that only uses 1 exponentiation to a random and short (e.g., 128-bit) exponent.

In the process, we formally define delegation of digital signature schemes, including changes to both the participant and the security model. First, we enrich the participant model of signature schemes, including a signer and a verifier, with a server who can assist both. Thus, both signer and verifier will be thought of as clients when interacting with the delegation server. Next, we generalize the standard unforgeability requirement to also hold in the presence of 2 additional classes of attacks introduced by the delegated computation paradigm: (1) eavesdropping of the communication between the server and the two clients (i.e., signer and the verifier), as well as (2) querying the server oracles, possibly done by the adversary after being able to perform a signer or verifier impersonation. We also provide a conversion theorem showing that a non-delegated signature scheme can be converted into a delegated signature scheme using a suitable delegation protocol for a desired primitive operation (e.g., single exponentiation, product of exponentiations, etc.). Establishing this result calls for the use of an alternative, and not weaker, simulation-based, definition of privacy in delegation protocols (many previous works targeted an indistinguishability-based definitions of privacy but can be seen to satisfy this definition as well).

Related Work. The first formal model for secure delegation protocols was presented in [31]. There, a secure delegation protocol is formally defined as essentially a secure function evaluation [45] of the client's function delegated to the server. Follow-up models from [29] and [13, 21] define separate requirements of correctness, (input) privacy and (result) security. There, the privacy requirement is defined in the sense of the adversary's indistinguishability of two different inputs from the client, even after corrupting the server; and the security requirement is defined in the sense of the adversary's inability to convince the client of an incorrect function output, even after corrupting the server. In our paper, we use a simulation-based definition of input privacy, which can be shown to imply the indistinguishability-based definition in [13, 21].

We can partition all other (single-server) secure delegation protocols we are aware of in 3 main classes, depending on whether they delegate (a) exponentiation in a specific group; (b) other specific computations (e.g., linear algebra); or (c) an arbitrary polynomial-time function.

With respect to (a), protocols were proposed for a single exponentiation in specific groups related to discrete logarithm or factoring problems (see, e.g., [14, 21, 31] and references therein). These protocols delegate exponentiation in settings where the client is assumed to be powerful enough to run a moderate number of group multiplications, but not enough to evaluate the delegated exponentiation function. There are also many protocols in the literature for the delegation of a single exponentiation, not targeting or achieving all our requirements (see, e.g., [15, 24, 34, 43]). In our model, protocols for the delegation of exponentiation in general groups were proposed in [13, 19], and protocols to delegate multiple exponentiations in specific groups were proposed in [20].

With respect to (b), protocols for linear algebra and/or scientific computation were proposed in, e.g., [2, 3, 8, 28, 35]. These protocols delegate various linear algebra operations in settings where the client is assumed to be powerful enough to run other linear algebra operations of lower time complexity, but not enough to evaluate the delegated linear algebra function.

With respect to (c), [29] proposed a protocol using garbled circuits [45] and fully homomorphic encryption [30]. This protocol delegates functions in settings where the client is powerful enough to run encryption and decryption algorithms of a fully homomorphic encryption scheme, but not enough to homomorphically evaluate a circuit that computes decryption steps in the garbling scheme for the function. Different protocols, not using garbled circuits, were later proposed in [16]. These protocols delegate functions in settings where the client is assumed to be powerful enough to run encryption and decryption algorithms of a fully homomorphic encryption scheme, but not enough to homomorphically evaluate the delegated function.

2 Definitions: Groups with Efficient Membership Proofs

In this section we formally define group notations and definitions that will be used in the rest of the paper.

Group notations and definitions. Let $(G, \star)$ be a group, let σ be its computational security parameter, and let L denote the length of the binary representation of elements in G. Typically, in cryptographic applications we set L as about equal to σ. We also assume that $(G, \star)$ is *cyclic*, has order q, and we fix m of its distinct generators as $g_1, \ldots, g_m$. By $y = g_1^{x_1} \cdots g_m^{x_m} = \prod_{i=1}^{m} g_i^{x_i}$ we denote the *product of m (fixed-base) exponentiations (in G)*. Let $\mathbb{Z}_q = \{0, 1, \ldots, q-1\}$, and let $F_{g_1,\ldots,g_m,q} : (\mathbb{Z}_q \times \ldots \times \mathbb{Z}_q) \rightarrow G$ denote the function that maps to each $(x_1, \ldots, x_m) \in \mathbb{Z}_q \times \ldots \times \mathbb{Z}_q$ the product of m (fixed-base) exponentiations (in G). By $desc(F_{g_1,\ldots,g_m,q})$ we denote a conventional description of the function $F_{g_1,\ldots,g_m,q}$ that includes its semantic meaning as well as generators $g_1, \ldots, g_m$, order q and the efficient algorithms computing multiplication and inverses in G.

By $t_{exp}(\ell)$ we denote a parameter denoting the number of multiplications in G used to compute an exponentiation (in G) of a group value to an arbitrary ℓ-bit exponent. By $t_{m,exp}(\ell)$ we denote a parameter denoting the number of multiplications in G used to compute m exponentiations (in G) of the same group value 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.

We define an *efficiently verifiable membership protocol* for G as a one-message protocol, denoted as the pair (mProve,mVerify) of algorithms, satisfying

1. *completeness*: for any $w \notin G$, mVerify(w,mProve(w))=1;
2. *soundness*: for any $w \notin G$, and any mProve$'$,
   mVerify(w,mProve$'$(w))=0;
3. *efficient verifiability*: the number of multiplications in G, denoted as $t_{mVerify}(\sigma)$, executed by mVerify is $o(t_{exp}(\sigma))$;
4. *efficient provability*: the number of multiplications $t_{mProve}(\sigma)$ in G executed by mProve is not significantly larger than $t_{exp}(\sigma)$.

We say that a group is *efficient* if its description is short (i.e., has length polynomial in σ), its associated operation and the inverse operation are efficient (i.e., they can be executed in time polynomial in σ), and it has an efficiently verifiable membership protocol. Note that for essentially all cyclic groups frequently used in cryptography, the description is short and both the associated operation and inverse operation can be run in time polynomial in σ. The only non-trivial property to establish is whether the group has an efficiently verifiable membership protocol. We now show two examples that are often used in cryptography and that do have efficiently verifiable membership protocols. In the rest of the paper we present our results for any arbitrary efficient cyclic group (using, for notation simplicity, a multiplicative notation for its operation).

Example 1: $(G, \star) = (G_q, \cdot \mod p)$, for large primes p, q such that $p = kq + 1$, where $k \neq q$ is another prime and G_q is the q-order subgroup of $\mathbb{Z}_p^\star$. This group is one of the most recommended for cryptographic schemes like the Diffie-Hellman key exchange protocol [23], El-Gamal encryption [27], Cramer-Shoup encryption [17], DSA etc. It is known that by Sylow's theorem, G_q in this case is the only subgroup of order q in the group $\mathbb{Z}_p^\star$ (i.e. $g^q = 1 \mod p$ if and only if $g \in G_q$). Also, the set of elements of G_q is precisely the set of k-th powers of elements of $\mathbb{Z}_p^\star$. Thus, an efficiently verifiable membership protocol can be built as follows:

1. on input w, mProve computes $r = w^{(q+1)/k} \mod p$ and returns r;
2. on input w, r, mVerify returns 1 if $w = r^k \mod p$ and 0 otherwise.

The completeness and soundness properties of this protocol are easily seen to hold. The efficient provability follows by noting that mProve only performs 1 exponentiation $\mod p$. The efficient verifiability property follows by noting that mVerify requires one exponentiation $\mod p$ to the k-th power. We note that mVerify is very efficient in the case when k is small (e.g., $k = 2$), which is a typical group setting in cryptographic protocols based on discrete logarithms. In the rest of the paper, we assume this specific group when we evaluate the performance of our protocol(s).

Example 2: $(G, +) = (E(\mathbb{F}_p)$, point addition), for a large prime $p > 3$: an elliptic curve E over a field $\mathbb{F}_p$, is the set of pairs $(x, y) \in E(\mathbb{F}_p)$ that satisfy the Weierstrass equation

$$y^2 = x^3 + ax + b \mod p,$$

together with the imaginary point at infinity $\mathcal{O}$, where $a, b \in \mathbb{F}_p$ and $4a^3 + 27b^2 \neq 0 \mod p$. The elliptic curve defined above is denoted by $E(\mathbb{F}_p)$. This group is one of the most recommended for cryptographic schemes like Elliptic-curve Diffie-Hellman key exchange protocol, Elliptic-curve ElGamal encryption, etc. Moreover, many discrete logarithm based cryptographic protocols defined over the set $\mathbb{Z}_p$ in Example 1 can be rewritten as defined over $E(\mathbb{F}_p)$. When those protocols are rewritten using the additive operation for this group instead of modular multiplication over $\mathbb{Z}_p$, the multiplication operation is rewritten as point addition and the exponentiation is rewritten as scalar multiplication in the group $E(\mathbb{F}_p)$, and the textbook "square-and-multiply" algorithm becomes a "double-and-add" algorithm. An efficiently verifiable membership protocol for this group simply consists of verifying the Weierstrass equation, as follows:

1. on input (x, y), mProve does nothing;
2. on input (x, y), mVerify returns 1 if $y^2 = x^3 + ax + b \mod p$ and 0 otherwise.

The completeness, soundness, efficient provability properties of this protocol are easily seen to hold. The efficient verifiability property follows by noting that mVerify performs only 4 multiplications $\mod p$.

3 Definitions: Delegated Protocols

In this section we formally define delegation protocols, and their correctness, security, privacy and efficiency requirements, mainly relying on the definition approach from [21], which in turn builds on those from [29, 31]. One new aspect in our definition, of interest for our results in later sections, is that we use a simulation-based definition of privacy instead of the indistinguishability-based definition in [21].

Basic notations. The expression $y \leftarrow T$ denotes the probabilistic process of randomly and independently choosing y from set T. The expression $y \leftarrow A(x_1, x_2, \ldots)$ denotes the (possibly probabilistic) process of running algorithm A on input $x_1, x_2, \ldots$ and any necessary random coins, and obtaining y as output. The expression $(z_A, z_B, tr) \leftarrow (A(x_1, x_2, \ldots), B(y_1, y_2, \ldots))$ denotes the (possibly probabilistic) process of running an interactive protocol between A, taking as input $x_1, x_2, \ldots$ and any necessary random coins, and B, taking as input $y_1, y_2, \ldots$ and any necessary random coins, where z_A, z_B are A and B's final outputs, respectively, at the end of this protocol's execution, and tr denotes the tuple of messages exchanged between A and B. We denote a distribution D as $D = \{R_1; \ldots; R_n : x\}$, where $R_1, \ldots, R_n$ is a sequence of random processes and x denotes a variable set as a result of their sequential execution.

System scenario, entities, and protocol. We consider a system with a single *client*, denoted as C, and a single *server*, denoted as S. As a client's computational resources are expected to be more limited than a server's ones, C is interested in delegating the computation of specific functions to S. We assume that the communication link between C and S is private or not subject to confidentiality, integrity, or replay attacks, and note that such attacks can be separately addressed using communication security techniques from any applied cryptography textbook (see, e.g., [36]). As in all previous work in the area, we consider a model with an offline phase, where, say, exponentiations to random exponents can be precomputed and made somehow available onto C's device. This model has been justified in several ways, all appealing to different application settings. In the presence of a trusted party, such as a deploying entity setting up C's device, the trusted party can simply perform the precomputed exponentiations and store them on C's device. If no trusted party is available, in the presence of a pre-processing phase where C's device does not have significant computation constraints, C can itself perform the precomputed exponentiations and store them on its own device.

Let σ denote the computational security parameter (i.e., the parameter derived from hardness considerations on the underlying computational problem), and let λ denote the statistical security parameter (i.e., a

parameter such that events with probability $2^{-\lambda}$ are extremely rare). Both parameters are expressed in unary notation (i.e., 1^σ, 1^λ).

Let $F : Dom(F) \rightarrow CoDom(F)$ be a function, where $Dom(F)$ denotes F's domain, $CoDom(F)$ denotes F's co-domain, and $desc(F)$ denotes F's description. Assuming $desc(F)$ is known to both C and S, and input x is known only to C, we define a *client-server protocol for the delegated computation of F* in the presence of an offline phase as a 2-party, 2-phase, communication protocol between C and S, denoted as $(C(1^\sigma, 1^\lambda, desc(F), x), S(1^\sigma, 1^\lambda, desc(F)))$, and consisting of the following steps:

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

As discussed above, Step 1 is executed in an *offline phase*, when the input x to the function F is not yet available. Step 2 is executed in the *online phase*, when the input x to the function F is available to C. At the end of both phases, C learns y_C, intended to be $= y$, and S learns y_S, usually an empty string in this paper. We will often omit $desc(F)$, 1^σ, 1^λ for brevity of description.

Correctness Requirement. Informally, the (natural) correctness requirement states that if both parties follow the protocol, C obtains some output at the end of the protocol, and this output is, with high probability, equal to the value obtained by evaluating function F on C's input. A formal definition follows.

Definition 1. Let σ, λ be the security parameters, let F be a function, and let (C, S) be a client-server protocol for the delegated computation of F. We say that (C, S) satisfies *δ_c-correctness* if for any x in F's domain, it holds that

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

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

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

1. $pp \leftarrow \text{Offline}(desc(F))$
2. $(y_C, y_S, tr) \leftarrow (C(pp, x), S)$
3. if $y_C = F(x)$ then **return:** 1
   else **return:** 0

Security Requirement. Informally, the most basic security requirement would state the following: if C follows the protocol, a malicious adversary corrupting S cannot convince C to obtain, at the end of the protocol, some output y' different from the value y obtained by evaluating function F on C's input x. To define a stronger security requirement, we augment the adversary's power so that the adversary can even choose C's input x, before attempting to convince C of an incorrect output. We also do not restrict the adversary to run in polynomial time. A formal definition follows.

Definition 2. Let σ, λ be the security parameters, let F be a function, and let (C, S) be a client-server protocol for the delegated computation of F. We say that (C, S) satisfies *ϵ_s-security against a malicious adversary* if for any algorithm A, it holds that

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

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

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

1. $pp \leftarrow \text{Offline}(desc(F))$
2. $(x, aux) \leftarrow A(desc(F))$
3. $(y', aux, tr) \leftarrow (C(pp, x), A(aux))$
4. if $y' = \perp$ or $y' = F(x)$ then **return:** 0
   else **return:** 1.

Privacy Requirement. Informally, the privacy requirement should guarantee the following: if C follows the protocol, a malicious adversary corrupting S cannot obtain any information about C's input x from a protocol execution. This is formalized here by extending the simulation-based approach typically used in various formal definitions for cryptographic primitives. That is, there exists an efficient algorithm, called the simulator, that generates a tuple of messages distributed exactly like those in a random execution of the protocol. A formal definition follows.

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 *privacy (in the sense of simulation) against a malicious adversary* if there exists an efficient algorithm Sim such that for any efficient adversary A and any input x to C, the following two distributions are equal:

$$D_{sim} = \{tr \leftarrow Sim(desc(F), 1^{\sigma}, 1^{\lambda}) : tr\}$$
$$D_{prot} = \{pp \leftarrow \text{Offline}(desc(F)); (y_C, y_A, tr_x) \leftarrow (C(pp, x), A(aux)) : tr_x\}$$

Efficiency Metrics and Requirements. In our analysis, we only consider the most expensive group operations as atomic operations, such as group multiplications and/or exponentiation, and neglect lower-order operations, such as equality testing, additions and subtractions between group elements.

Let (C, S) be a client-server protocol for the delegated computation of function F with computational security parameter σ and statistical correctness parameter λ. We say that (C, S) has *efficiency parameters* $(t_F, t_P, t_C, t_S, sc, cc, mc)$, if

1. F can be computed (without delegation) using $t_F(\sigma, \lambda)$ atomic operations;
2. the offline phase can be run using $t_P(\sigma, \lambda)$ atomic operations;
3. C can be run in the online phase using $t_C(\sigma, \lambda)$ atomic operations;
4. S can be run using $t_S(\sigma, \lambda)$ atomic operations;
5. at the end of the offline phase, data with storage complexity sc is stored on C's device;
6. C and S exchange a total of at most mc messages; and
7. C and S exchange messages of total length at most cc.

While we naturally try to minimize all these protocol efficiency metrics, our main goal is to design protocols where

1. $t_C(\sigma, \lambda) << t_F(\sigma, \lambda)$, and
2. $t_S(\sigma, \lambda)$ is not significantly larger than $t_F(\sigma, \lambda)$,

based on the underlying assumption, consistent with the state of the art in cryptographic implementations, for essentially all group types, that group multiplication requires significantly less computing resources than group exponentiation.

4 Delegating a Product of Exponentiations

In this section we present our protocol for delegation of a product of (fixed-base) exponentiations in a large class of groups used in cryptographic protocols, which provably satisfies correctness, simulation-based privacy, security with exponentially small probability, and various desirable efficiency properties. Most notably, the client's online complexity is dominated by a single exponentiation to a significantly smaller exponent.

We first formally state our result, then describe the protocol, and finally prove its correctness, security, privacy and efficiency properties.

Formal theorem statement. We obtain the following

Theorem 4. Let $(G, \star)$ be an efficient cyclic group, let σ be its computational security parameter, and let λ be a statistical security parameter. There exists (constructively) a client-server protocol (C, S) for delegating the computation of function $F_{g_1,\ldots,g_m,q} : (\mathbb{Z}_q \times \ldots \times \mathbb{Z}_q) \to G$, which satisfies

1. δ_c-correctness, for $\delta_c = 1$;
2. ϵ_s-security, for $\epsilon_s \leq \frac{1}{2^\lambda}$;
3. simulation-based privacy;
4. efficiency with parameters $(t_F, t_S, t_P, t_C, sc, cc, mc)$, where
   - t_F is $= t_{prod,m,exp}(\sigma)$;
   - t_S is $= 2\, t_{prod,m,exp}(\sigma) + 2\, t_{mProve}(\sigma)$;
   - t_P is $= 2\, t_{prod,m,exp}(\sigma)$, with random exponents from $\mathbb{Z}_q$;
   - t_C is $\leq t_{exp}(\lambda) + 2\, t_{mVerify}(\sigma) + 2$ multiplications in G and m multiplications in $\mathbb{Z}_q$;
   - $sc = 2$ elements in G and $2m$ elements of $\mathbb{Z}_q$
   - $cc = 4$ elements in G and $2m$ elements of $\mathbb{Z}_q$
   - $mc = 2$.

The main takeaway from Theorem 4 is that C delegates the computation of product of multiple (i.e. m) exponentiations with a σ-bit exponents to S while C only performs an exponentiation with a λ-bit exponent, 2 group membership verifications in G, 2 multiplications in G and m modular multiplications in $\mathbb{Z}_q$. In other words, C's online complexity is only slightly larger than that in a delegation protocol for a *single* exponentiation, as in the protocol from [21]. In fact, our protocol can be seen as a non-trivial extension of the single exponentiation protocol in [21], based on two main ideas: (1) using a single small-coefficient linear verification test on the entire product of m exponentiations, instead of an independent test on each of the m exponentiations in the product; and (2) carefully redistributing the computation of products of exponentiation from the client to the server and the offline phase. This results in savings of about a multiplicative factor of m in the client's online complexity over a direct use of that protocol to delegate a single exponentiation. Using the group in Example 1 from Section 2 for a concrete comparison, the client performs $2\lambda + m + 4$ multiplications, while in a direct use of the protocol in [21] that bound would be $2m\lambda + 4m$, and in non-delegated computation one can perform up to $2m\sigma + m - 1$ multiplications. Using current typical settings in applied cryptography (i.e., $\sigma = 2048$, and $\lambda = 128$), and assuming m ranging from 2 to 128, we see that in our protocol the client's online multiplications are smaller by 2-3 orders of magnitude than non-delegated computation and 1-2 orders of magnitude with respect to a direct use of the delegation of a single exponentiation from [21].

Also remarkable are the running time of S, who only performs 2 products of m exponentiations and 2 group membership proof generations in G. In other words, S's complexity is only about 4 times as that in a non-delegated computation of the same function.

Even in the offline phase, only 2 products of fixed-base exponentiations with random exponents are needed by the client to later compute a product of m fixed-base exponentiations. Finally, the protocol only requires 2 messages, which is clearly minimal in this model, only requires the communication of 4 elements in G and $2m$ elements in $\mathbb{Z}_q$, and only requires that $2m$ elements in $\mathbb{Z}_q$ and 2 group values are stored on C's device at the end of the offline phase.

In what follows we prove Theorem 4 by describing our delegation protocol and proving its properties. The group membership test is realized via the assumed efficiently verifiable group membership protocol. While we do not know of such a protocol for any arbitrary cyclic group, we showed in Section 2 that two groups commonly used in cryptography have one.

Informal description of protocol (C, S)**.** Our starting point is the protocol for private, secure and efficient delegation of fixed-base exponentiation in cyclic groups in [21], also reviewed in Appendix A. There, one main idea consists of a probabilistic verification equation which is verifiable using a much smaller number of modular multiplications (i.e., up to 2λ, instead of 2σ, multiplications). Specifically, in that protocol, C injects an additional random value $b \in \{1, \ldots, 2^\lambda\}$ in one of the inputs on which S is asked to computed the value of the exponentiation function $F_{g,q}$, so to satisfy the following properties: (a) if S returns correct computations of $F_{g,q}$, then C can correctly compute y with a single group multiplication; (b) if S returns

incorrect computations of $F_{g,q}$, then S either does not meet some deterministic verification equation or can only meet C's probabilistic verification equation for at most one possible value of random value b; (d) C can check whether the probabilistic verification equation is satisfied with an exponentiation to the (shorter) exponent b; and (d) C's messages hide the values of the random element as well as C's input to the function. By choosing a large enough domain for b (e.g., setting $\lambda \geq 128$), the protocol achieves a very small security probability (i.e., $2^{-\lambda}$). As this domain is much smaller than the group, this results in a considerable efficiency gain on C's running time.

Towards the design of our protocol proving Theorem 4, a first natural approach is that the client delegates each of the m exponentiations in the product using the delegation protocol for fixed-base exponentiation over cyclic groups in [21], and finally the client computes the product of the obtained m exponentiations. Note that this approach would satisfy correctness, privacy and security requirements. However, when it comes to performance, it is undesirable as it multiplies by a factor of about m both the number of multiplications by the client and the size of the client's storage, and therefore we would gradually lose the computation benefit from the delegation as m gets larger. In the protocol presented here, we target an additive overhead of m, instead of a multiplicative one, and achieve this with the following two main modifications.

First, we view the probabilistic test of [21] as a small-coefficient linear test over a group value's exponent. Then, by using the linear homomorphism properties of the exponentiation function, we can define a single small-coefficient linear test on the entire product of m exponentiations, instead of an independent test on each of the m exponentiations in the product. Thus, a single random coefficient value b is used in the protocol, instead of m random and independent values, so that C only performs a single exponentiation to the small exponent b to run the probabilistic verification equation in the resulting small-coefficient linear test.

Second, no products of exponentiations are performed by the client. When these are needed in the protocol, they are carefully redistributed to the computationally more powerful server or to the offline phase, where more computational power is available. Specifically, our protocol involves 4 products of m exponentiations, of which 2 are performed in the offline phase and 2 are computed by the server. Finally, the computation of these products is set up so that by the homomorphism properties of the exponentiation function, analogue group membership verifications and probabilistic verification test can be performed as in the original protocol, although on products of exponentiations instead of single exponentiations.

Formal description of protocol (C, S)**.** Let G be an efficient cyclic group, and let (mProve, mVerify) denote its efficiently verifiable membership protocol.

Input to C and S: 1^σ, 1^λ, $desc(F_{g_1,\ldots,g_m,q})$

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

Offline phase instructions:

1. Randomly choose $u_{i,j} \in \mathbb{Z}_q$, for $i = 1, \ldots, m$ and $j = 0, 1$
2. Set $v_j = \prod_{i=1}^{m} g_i^{u_{i,j}}$ and store $(u_{1,j}, \ldots, u_{m,j}, v_j)$ on C for $j = 0, 1$

Online phase instructions:

1. C randomly chooses $b \in \{1, \ldots, 2^\lambda\}$
   C sets $z_{i,0} := (x_i - u_{i,0}) \mod q$, $z_{i,1} := (b \cdot x_i + u_{i,1}) \mod q$ for $i = 1, \ldots, m$
   C sends $(z_{i,0}, z_{i,1})$ to S for $i = 1, \ldots, m$
2. S computes $w_j := \prod_{i=1}^{m} g_i^{z_{i,j}}$ and $\pi_j :=$ mProve(w_j), for $j = 0, 1$
   S sends w_0, w_1, π_0, π_1 to C
3. C computes $y := w_0 \star v_0$
   C checks that
   $w_1 = y^b \star v_1$, also called the 'probabilistic test'
   mVerify$(w_0, \pi_0) =$ mVerify$(w_1, \pi_1) = 1$,
   also called the 'membership test'
   if any one of these tests is not satisfied then

C **returns:** $\perp$ and the protocol halts
C **returns:** y

Properties of 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.
- *Storage complexity:* at the end of the offline phase, tuples $(u_{1,j}, \ldots, u_{m,j}, v_j)$, for $j = 0, 1$, are stored on C's device, resulting in a storage complexity of $2m$ values in $\mathbb{Z}_q$ and 2 group elements.
- *Communication complexity:* the protocol requires the transfer of 2 elements in G and 2 proofs of group membership from S to C, and $2m$ elements in $\mathbb{Z}_q$ from C to S.
- *Runtime complexity:* During the offline phase, 2 product of m exponentiations in bases $g_1, \ldots, g_m$ and with random σ-bit exponents are performed. This product of m exponentiations can be evaluated using any of the cited literature algorithms for a product of m exponentiations (e.g., the algorithm in [40]). During the online phase, S computes 2 products of m exponentiations to σ-bit exponents in G and 2 group membership proofs; and C verifies 2 group membership proofs and computes 2 multiplications in G, m modular multiplications in $\mathbb{Z}_q$, and 1 exponentiation in G to a random exponent that is $\leq 2^{\lambda}$ and thus much smaller than 2^{σ}.

The *correctness* property follows by showing that if C and S follow the protocol, C always output $y = \prod_{i=1}^{m} g_i^{x_i}$. We show that the 2 tests performed by C are always passed. The membership test is always passed since w_j is computed by S as $\prod_{i=1}^{m} g_i^{z_i}$, for $j = 0, 1$, and $g_1, \ldots, g_m$ are generators of group G; the probabilistic test is always passed since

$$w_1 = \prod_{i=1}^{m} g_i^{z_{i,1}} = \prod_{i=1}^{m} g_i^{bx_i+u_{i,1}} = \left(\prod_{i=1}^{m} g_i^{x_i}\right)^{b} \star \prod_{i=1}^{m} g_i^{u_{i,1}} = y^b v_1.$$

This implies that C never returns $\perp$, and thus returns y. To see that this returned value y is the correct output, note that

$$y = w_0 \star v_0 = \prod_{i=1}^{m} g_i^{z_{i,0}} \star \prod_{i=1}^{m} g_i^{u_{i,0}} = \prod_{i=1}^{m} g_i^{x_i-u_{i,0}} \star \prod_{i=1}^{m} g_i^{u_{i,0}} = \prod_{i=1}^{m} g_i^{x_i}.$$

The *privacy* property of the protocol against any arbitrary malicious S is proved by showing an efficient simulator Sim such that for any input $x_1, \ldots, x_m \in \mathbb{Z}_q$ to C, the following two distributions are equal: the distribution D_{prot} of the messages in a random execution of (C, S) where C uses $x_1, \ldots, x_m$ as input; and the distribution D_{sim} output by Sim. First, we observe that C's only message to S does not depend on values $x_1, \ldots, x_m$. Specifically, this message can be written as $(z_{1,0}, \ldots, z_{m,0}, z_{1,1}, \ldots, z_{m,1})$ where $z_{i,0} = (x_i - u_{i,0}) \mod q$, $z_{i,1} = (bx_i + u_{i,1}) \mod q$, and $z_{i,0}$ and $z_{i,1}$ are uniformly and independently distributed in $\mathbb{Z}_q$, as so are $u_{i,0}$ and $u_{i,1}$ for all $i = 1, \ldots, m$. Thus, a simulator Sim can be defined as the algorithm that, on input 1^{σ}, 1^{λ}, $desc(F_{g_1,\ldots,g_m,q})$, runs the following instructions:

1. generate a tuple $mes_1 = (z'_{1,0}, \ldots, z'_{m,0}, z'_{1,1}, \ldots, z'_{m,1})$ of random and independent values in $\mathbb{Z}_q$
2. generate message $mes_2 = (w'_0, w'_1, \pi'_0, \pi'_1)$ by running the same instructions run by S on input 1^{σ}, 1^{λ}, $desc(F_{g_1,\ldots,g_m,q})$ and $(z'_{1,0}, \ldots, z'_{m,0}, z'_{1,1}, \ldots, z'_{m,1})$
3. return: (mes_1, mes_2)

We obtain that distribution D_{Sim} and distribution D_{prot} are identical since in both distributions the first message contains $2m$ random and independent values in $\mathbb{Z}_q$ and the second message is computed using the same algorithm starting from first message.

To prove the *security* property against any malicious S we need to compute an upper bound ϵ_s on the security probability that S convinces C to output a y such that $y \neq F_{g_1,\ldots,g_m,q}(x_1, \ldots, x_m)$. We start by defining the following events with respect to a random execution of (C, S) where C uses x as input:

- $e_{y,\neq}$, defined as 'C outputs y such that $y \neq F_{g_1,\ldots,g_m,q}(x_1, \ldots, x_m)$'

- $e_{w,\neq}$, defined as 'In its message to C, S sends a pair $(w'_0, w'_1) \neq (w_0, w_1)$'
- $e_{\perp}$, defined as 'C outputs $\perp$'

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

Fact 4.1. *If event $e_{y,\neq}$ happens then event $e_{w,\neq} \wedge (\neg\, e_{\perp})$ happens.*

With respect to a random execution of (C, S) where C uses $x_1, \ldots, x_m$ as input, we now define the following events:

- $e_{1,b}$, defined as '$\exists$ exactly one b such that the pair (w'_0, w'_1) sent by S to C satisfies $w'_1 = (w'_0 \star v_0)^b \star v_1$'
- $e_{>1,b}$, defined as '$\exists$ more than one b such that the pair (w'_0, w'_1) sent by S to C satisfies $w'_1 = (w'_0 \star v_0)^b \star v_1$'.

By definition, events $e_{1,b}, e_{>1,b}$ are each other's complement event.

Now, let $i \in \{1, \ldots, m\}$. We observe that no information is leaked by $z_{i,0}, z_{i,1}$ about x_i as: (a) for any $x_i \in \mathbb{Z}_q$, there is exactly one $u_{i,0}$ corresponding to $z_{i,0}$; that is, $u_{i,0} = x_i - z_{i,0} \mod q$; (b) for any $x_i \in \mathbb{Z}_q$, for any $b \in \{1, \ldots, 2^\lambda\}$ chosen by C, there is exactly one $u_{i,1}$ corresponding to $z_{i,1}$; that is, $u_{i,1} = z_{i,1} - bx_i \mod q$ for all $i = 1, \ldots, m$. This implies that, since $u_{i,0}, u_{i,1}$ are uniformly and independently distributed in $\mathbb{Z}_q$, the distribution of tuple $(x_1, \ldots, x_m)$ input to C is independent from the distribution of tuple $((z_{1,0}, z_{1,1}), \ldots, (z_{m,0}, z_{m,1}))$ sent by C to S. Furthermore, by essentially the same proof, protocol (C, S) satisfies the following property: for any $x_1, \ldots, x_m \in \mathbb{Z}_q$, the value of $b \in \{1, \ldots, 2^\lambda\}$ chosen by C is independent from tuple $((z_{1,0}, z_{1,1}), \ldots, (z_{m,0}, z_{m,1}))$. This implies that all possible values for b in $\{1, \ldots, 2^\lambda\}$ are still equally likely even when conditioning over message $((z_{1,0}, z_{1,1}), \ldots, (z_{m,0}, z_{m,1}))$ from C to S. Then, if event $e_{1,b}$ is true, the probability that the pair (w'_0, w'_1) sent by S to C satisfies the probabilistic test, is equal to 1 divided by the number 2^λ of values of b that are still equally likely even when conditioning over message $(z_{1,0}, \ldots, z_{m,0}, z_{1,1}, \ldots, z_{m,1})$. We obtain the following

Fact 4.2. $\text{Prob}\left[\, e_{w,\neq} \wedge (\neg\, e_{\perp}) \,|\, e_{1,b} \,\right] \leq 1/2^\lambda$

We now show the main technical claim, saying that if S is malicious then it cannot produce in step 2 of the protocol a pair of values (w'_0, w'_1) different than the required pair (w_0, w_1), satisfying both of C's tests for two distinct values $b_1, b_2 \in \{1, \ldots, 2^\lambda\}$. Since S can be malicious, in step 2 it can send arbitrary values to C. In particular, S can send a pair (w'_0, w'_1) different than the pair (w_0, w_1) required by the protocol, where $w_j = \Pi_{i=1}^m g^{z_{ij}}$, for $j = 0, 1$. Since C uses π_0, π_1 to check in step 3 that the two group elements belong to the group, we can assume that $w'_0, w'_1 \in G$. Moreover, since G is cyclic, and we assumed that each g_i is generator of G, for $i = 1, \ldots, m$, we can, without loss of generality, consider generator g_1 and write

$$w'_0 = g_1^u \star w_0 \text{ and } w'_1 = g_1^v \star w_1 \text{ for some } u, v \in \mathbb{Z}_q.$$

Thus, we can also write $y = w'_0 \star v_0 = g_1^u \star w_0 \star v_0 = g_1^u \star \prod_{i=1}^m g_i^{x_i}$. Now, recall that the goal of a malicious S is to pass C's two verification tests and force C's output to be $y \neq \prod_{i=1}^m g_i^{x_i}$, which is true when $u \neq 0 \mod q$. We then consider the following equivalent rewriting of C's probabilistic test, obtained by variable substitutions and simplifications:

$$\begin{aligned}
w'_1 &= y^b \star v_1 \\
g_1^v \star w_1 &= \left(g_1^u \star \prod_{i=1}^m g_i^{x_i} \right)^b \star \prod_{i=1}^m g_i^{u_{i,1}} \\
g_1^v \star \prod_{i=1}^m g_i^{z_{i,1}} &= g_1^{ub} \star \prod_{i=1}^m g_i^{bx_i + u_{i,1}} \\
g_1^v \star \prod_{i=1}^m g_i^{bx_i + u_{i,1}} &= g_1^{ub} \star \prod_{i=1}^m g_i^{bx_i + u_{i,1}} \\
g_1^v &= g_1^{ub}
\end{aligned}$$

$$v = ub \mod q.$$

Notice that if $u = 0 \mod q$ then the above calculation implies that $v = 0 \mod q$, and thus S is honest, from which we derive that $\epsilon_s = 0$. Now consider the case S is dishonest, in which case we have that $u \neq 0 \mod q$. We want to show that b is unique in this case. If there exist two distinct b_1 and b_2 such that

$$ub_1 = v \mod q \text{ and } ub_2 = v \mod q$$

then $u(b_1 - b_2) = 0 \mod q$ then $b_1 - b_2 = 0 \mod q$ (i.e $b_1 = b_2$) because $u \neq 0 \mod q$. This shows that b is unique and we obtain the following fact.

Fact 4.3. $\text{Prob}\left[e_{>1,b} \right] = 0$

The rest of the proof consists of computing an upper bound ϵ_s on the probability of event $e_{y,\neq}$, using all previously established facts. We obtain the following

$$\begin{aligned}
\text{Prob}\left[e_{y,\neq} \right] &\leq \text{Prob}\left[e_{w,\neq} \wedge (\neg\, e_{\perp}) \right] \\
&= \text{Prob}\left[e_{1,b} \right] \cdot \text{Prob}\left[e_{w,\neq} \wedge (\neg\, e_{\perp}) \,|\, e_{1,b} \right] \\
&\quad + \text{Prob}\left[e_{>1,b} \right] \cdot \text{Prob}\left[e_{w,\neq} \wedge (\neg\, e_{\perp}) \,|\, e_{>1,b} \right] \\
&= \text{Prob}\left[e_{1,b} \right] \cdot \text{Prob}\left[e_{w,\neq} \wedge (\neg\, e_{\perp}) \,|\, e_{1,b} \right] \\
&\leq \text{Prob}\left[e_{1,b} \right] \cdot \frac{1}{2^{\lambda}} \\
&\leq \frac{1}{2^{\lambda}},
\end{aligned}$$

where the first inequality follows from Fact 4.1, the first equality follows from the definition of events $e_{1,b}, e_{>1,b}$ and the conditioning rule, the second equality follows from Fact 4.3, and the second inequality follows from Fact 4.2.

We can finally set $\epsilon_s = 2^{-\lambda}$, which concludes the proof for the security property for (C, S). □

4.1 Performance

A naive algorithm to compute (without delegation) a product of m exponentiations consists of first computing single exponentiations $y_i = g_i^{x_i}$ for $i = 1, \ldots, m$, and then the product $\prod_{i=1}^{m} y_i$. For this algorithm, which we call *nPoExp*, we have that $t_{prod,m,exp}(\ell) = m \cdot t_{exp}(\ell) + m - 1$, which is equal to $2m\sigma + m - 1$, when a single exponentiation is computed using the square-and-multiply algorithm.

Several papers propose faster algorithms to compute single exponentiations (see, e.g., [7, 10, 18, 25, 37, 44]), as well as a product of m exponentiations (see, e.g., [10, 37, 40, 42]). For instance, using the closed-form estimate from [11] of Pippenger's algorithm in [40], one can obtain an algorithm, which we denote as *fPoExp*, satisfying $t_{prod,m,exp}(\ell) \sim \ell(1 + m/(\log m + \log \ell))$.

As yet another comparison method to delegate the computation of a product of m exponentiations to σ-bit exponent, we define protocol *nDelPoExp* in which a client delegates to a server the computation of each of the m exponentiations using Protocol 1 from [21] and then directly computes a product of the m obtained exponentiations.

In Table 1 we show concrete evaluations for our protocol's main performance metric: the client's number t_C of group multiplications in the online phase. In particular, we also compare our protocol with the delegated protocol *nDelPoExp* and the non-delegated algorithms *nPoExp* and *fPoExp*, for computing exponentiation in group $\mathbb{Z}_p^{\star}$, for $p = 2q + 1$, where p, q are primes. First we show the numbers for t_C for varying and arbitrary values of m, while setting $\sigma = 2048$ and $\lambda = 128$, the currently recommended parameter settings for many cryptographic applications. Then, in the last row we show closed-form expressions for t_C with respect to arbitrary m, σ, λ. We observe that our result's improvement is significant in many practical parameter settings.

For small values of m, our result improves by 1-2 orders of magnitude over the non-delegated algorithms and between a constant factor and 1 order of magnitude over the delegation approach based on [21]. For large values of m, our result improves by at least 3 orders of magnitude over the non-delegated algorithms and at least 2 orders of magnitude over the delegation approach based on [21].

Table 1: The number of C's online multiplications in the example group $(\mathbb{Z}_p^\star, \cdot \mod p)$ where $p = 2q + 1$, and p, q are primes.

		$F_{g_1,\ldots,g_m,q}$ No delegation		$F_{g_1,\ldots,g_m,q}$ With delegation	
	m	*nPoExp*	*fPoExp*	*nDelPoExp*	Our result
	2	8,193	2,389	520	262
	5	20,484	2,779	1,300	265
$\sigma = 2048$	10	40,969	3,413	2,600	270
$\lambda = 128$	50	204,849	8,072	13,000	310
	100	409,699	13,426	26,000	360
	1000	4,096,999	99,512	260,000	1,260
	Arbitrary m	$4097m - 1$	$2048(1 + \frac{m}{\log m + 11})$	$260m$	$260 + m$
Arbitrary m, σ, λ		$2m\sigma + m - 1$	$\sigma(1 + \frac{m}{\log m + \log \sigma})$	$2m\lambda + 4m$	$2\lambda + m + 4$

In Table 2 and Table 3 we report performance results measured when running our software implementation of our protocol in Section 4 and of the protocol *nDelPoExp* in [21] for the multiplicative group $(\mathbb{Z}_p^\star, \cdot \mod p)$, for $p = 2q + 1$, where p, q are large primes, and using $\sigma = 2048$ and $\lambda = 128$. Our implementation was carried out on a macOS Catalina Version 10.15.1 laptop with 2.9 GHz Intel Core i9 processor with memory 32 GB 2400 MHz DDR4. The protocols were coded in Python 3.7 using the gmpy2 package. 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, and improvement ratio t_F/t_C, for different values of parameter m (i.e., $m = 2, 4, 7, 10, 50, 100$), and using both

Table 2: Performance of the protocol when $\sigma = 2048$, $\lambda = 128$ and group $G = \mathbb{Z}_p^\star$ where $p = 2q + 1$ where p and q are primes.

$m =$	2		4		7		10		50		100	
	SM	NoSM	SM	NoSM	SM	NoSM	SM	NoSM	SM	NoSM	SM	NoSM
$t_F =$	0.0255	0.0130	0.0562	0.0263	0.0950	0.0463	0.1254	0.0641	0.4911	0.3442	0.5024	0.6498
$t_P =$	0.0458	0.0262	0.1056	0.0530	0.1886	0.0917	0.2563	0.1361	0.9710	0.6718	1.0096	1.3179
$t_C =$	0.0007	0.0005	0.0009	0.0005	0.0008	0.0005	0.0007	0.0005	0.0008	0.0008	0.0005	0.0009
$t_S =$	0.0740	0.0372	0.1311	0.0637	0.2146	0.1084	0.2830	0.1501	0.9980	0.6726	1.0192	1.3210
$\frac{t_F}{t_C} =$	**35.58**	**23.73**	**63.36**	**56.39**	**111.90**	**85.87**	**167.26**	**119.76**	**637.82**	**445.71**	**1025.5**	**687.98**

Table 3: Performance of the protocol using [21] of protocol *nDelPoExp* when $\sigma = 2048$, $\lambda = 128$ and group $G = \mathbb{Z}_p^\star$ where $p = 2q + 1$ where p and q are primes.

$m =$	2		4		7		10		50		100	
	SM	NoSM	SM	NoSM	SM	NoSM	SM	NoSM	SM	NoSM	SM	NoSM
$t_F =$	0.0099	0.0048	0.0195	0.0096	0.0342	0.0165	0.0488	0.0237	0.2430	0.1203	0.5342	0.2365
$t_P =$	0.0196	0.0096	0.0389	0.0192	0.0682	0.0332	0.0975	0.0471	0.4872	0.2389	9.9175	0.4731
$t_C =$	0.0006	0.0004	0.0012	0.0007	0.0022	0.0013	0.0031	0.0018	0.0156	0.0091	0.0346	0.0179
$t_S =$	0.0390	0.0191	0.0773	0.0383	0.1355	0.0663	0.1938	0.0940	0.9673	0.4770	4.5677	0.9444
$\frac{t_F}{t_C} =$	**15.68**	**13.22**	**15.65**	**13.12**	**15.64**	**13.21**	**15.60**	**13.21**	**15.60**	**13.27**	**15.45**	**13.23**

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'). Conclusions from our empirical results in Tables 2 and 3 essentially confirm our conclusions from our analytical results in Table 1.

5 Delegating Signature Schemes

In this section we show private, secure and efficient delegation schemes for well-known (i.e., ElGamal, Schnorr and Okamoto's) signature schemes using the delegation of a product of (fixed-base) exponentiation for cyclic groups from Section 4. We start the presentation by recalling in Section 5.1 the definition of signature schemes in the standard (i.e., non-delegated) model. In Section 5.2 we augment this definition so to additionally take into account eavesdropping and oracle query attacks in the delegated model. Then, in Section 5.3 we present a general result that shows how to convert signature schemes in the non-delegated model into signature schemes in the delegated model by using a suitable delegation protocol. Finally, in Section 5.4 we show delegated ElGamal, Schnorr and Okamoto's signature schemes by simply showing modification to the original algorithms where signers and/or verifiers use the delegated protocol for computing a product of exponentiations. The proof that these modifications result in correct, secure and private delegated signature schemes directly follows from our general result in Section 5.3.

5.1 Definitions: Signature Schemes in the standard model

We now recall the definition of digital signature schemes in the standard (i.e., non-delegated) model.

Notations and algorithm syntax. An *oracle*, denoted as $O(\cdot)$, is a function. An *oracle algorithm*, denoted as $A^{O(\cdot)}$, is an algorithm that during its computation can repeatedly make a query to the oracle and obtain the corresponding oracle's output.

In a signature scheme SS, we consider two types of parties: signers and verifiers, and three algorithms: a key-generation algorithm KG, a signing algorithm Sign, and a verification algorithm Ver, satisfying the following syntax and requirements.

On input a security parameter 1^σ, algorithm KG returns a public key pk and a matching secret key sk. On input a message m of arbitrary length, algorithm Sign returns a signature sig. On input a putative message m', and a putative signature sig', algorithm Ver returns a bit = 1 (resp., 0) to denote that sig' is a valid (resp., not valid) signature of m'.

Requirements: Correctness and Unforgeability. Informally speaking, the correctness requirement states that if both signer and verifier correctly run the algorithms, the verifier can recognize the signer's signature as valid; and the unforgeability requirement states that no efficient algorithm querying the signature oracle can produce a message with a valid signature. Formal definitions follow.

Definition 5. We say that SS=(KG,Sign,Ver) satisfies *δ-correctness* if for any message $m \in \{0,1\}^*$, it holds that

$$\text{Prob}\left[(pk, sk) \leftarrow \text{KG}(1^\sigma); sig \leftarrow \text{Sign}(pk, sk, m) : \text{Ver}(pk, m, sig) = 1\right] \geq \delta,$$

for some δ close to 1.

Definition 6. We say that the signature scheme SS=(KG,Sign,Ver) satisfies *existential ϵ-unforgeability under chosen message attack* (briefly, ϵ-cma-EU) if for any efficient oracle algorithm A, it holds that

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

for some ϵ close to 0, where experiment SecExp is detailed below:

$\text{SecExp}_{\text{SS,A}}(1^\sigma)$

1. $(pk, sk) \leftarrow \text{KG}(1^\sigma)$
2. $(m', sig') \leftarrow A^{\text{Sign}(pk,sk,\cdot)}(pk)$
3. Let Q be the set of message queries made by A to oracle $\text{Sign}(pk, sk, \cdot)$
4. if $m' \in Q$ or $\text{Ver}(pk, sk, m', sig') = 0$ then **return:** 0
   else **return:** 1.

5.2 Definitions: Delegated Signature Schemes

Given a (non-delegated) signature scheme SS = (KG,Sign,Ver), as defined in Section 5.1, and a delegation protocol (C, S) for a function F, as defined in Section 3, we now formally define an associated delegated signature scheme dSS.

Notations and algorithm syntax. We consider three parties: a signer, a verifier, and a server, where during their computations the signer and/or the verifier may act as clients interacting with the server. Since in this paper we only use one-round client-server delegation protocols, we first model the server as an oracle that answers client queries, and then model the signer and verifier's interactions with the server as calls to the server oracle.

For each one-round delegation protocol (C, S) for a function F, we define an *(F, C, S)-associated server oracle*, denoted as $S(desc(F), 1^\sigma, 1^\lambda, \cdot)$, as the oracle taking as query input C's message in (C, S) and returning as output the server S's response to this message according to protocol (C, S).

An *oracle signature algorithm*, denoted as Sign^S, is defined as an algorithm with the same syntax as signature algorithm Sign, but with the additional capability of making queries to an oracle S. Analogously, an *oracle verification algorithm*, denoted as Ver^S, is defined as an algorithm with the same syntax as a verification algorithm Ver, but with the additional capability of making queries to an oracle S. An *oracle signature scheme* is defined as a signature scheme $(\text{KG},\text{Sign}^S,\text{Ver}^S)$ where signature and verification algorithms are actually oracle signature and oracle verification algorithms, capable of querying the same oracle S. We then say that an oracle signature scheme $(\text{KG},\text{Sign}^S,\text{Ver}^S)$ is *(F, C, S)-compatible* if oracle signing algorithm Sign^S and oracle verification algorithm Ver^S are semantically equivalent to the signature algorithm Sign and the verification algorithm Ver from SS, in the sense that on the same input, the final output from Sign^S and Ver^s is identical to the output from Sign and Ver, respectively (but in the middle of its computation, Sign^S and Ver^S may also perform queries to S). Finally, we formally define the *(SS,F, C, S)-compatible delegated signature scheme* dSS as the tuple $(S, \text{KG}, \text{Sign}^S, \text{Ver}^S)$, where S is the (F, C, S)-associated server oracle and $(\text{KG},\text{Sign}^S,\text{Ver}^S)$ is the (F, C, S)-compatible oracle signature scheme.

In our formal description of the protocols, we will actually separate algorithms Sign^S and Verify^S into an offline-phase and online-phase version, for the purpose of minimizing the online complexity; however, to reduce notation in the description of the model, in this subsection we keep both offline and online version as a single algorithm.

Requirements: Correctness and Unforgeability. The requirements of correctness and unforgeability for dSS are also obtained by suitably augmenting those for SS. In the case of correctness, the extension is immediate. In the case of unforgeability, we replace the adversary A's oracle $Sign$ with two oracles:

1. an augmented oracle $dSign(pk, sk, \cdot)$ which, on input message m, returns a signature sig as well as the transcript of any query/answer interaction with the server oracle S performed by Sign during the generation of sig;
2. the server oracle $S(desc(F), 1^\sigma, 1^\lambda, \cdot)$, which, on input C's query message $qmes_C$, returns S's response to $qmes_C$ in an execution of protocol (C,S).

Note that by giving the adversary access to oracle $dSign$, we model the adversary's eavesdropping attacks on executions of the delegation protocol between a signer (acting as client) and the server, as well as between

a verifier (acting as client) and the server. Moreover, by giving the adversary access to oracle S, we model the adversary's interaction with the server while colluding with a signer or verifier. Formal definitions of correctness and unforgeability requirements for dSS follow.

Definition 7. Let F be a function, and (C, S) be a delegation protocol for F, and let S be the (F, C, S)-associated server oracle. We say that the (SS,F, C, S)-compatible delegated signature scheme dSS = (S, KG, Sign^S, Ver^S) satisfies δ-*correctness* if for any message $m \in \{0,1\}^\star$, it holds that

$$\text{Prob}\left[(pk, sk) \leftarrow \text{KG}(1^\sigma); sig \leftarrow \text{Sign}^S(pk, sk, m) : \text{Ver}^S(pk, m, sig) = 1\right] \geq \delta,$$

for some δ close to 1.

Definition 8. Let F be a function, and (C, S) be a delegation protocol for F, and let S be the (F, C, S)-associated server oracle. We say that the (SS,F, C, S)-compatible delegated signature scheme dSS = (S, KG, Sign^S, Ver^S) satisfies *existential ϵ-unforgeability under chosen message attack* (briefly, ϵ-cma-EU) if for any efficient oracle algorithm A, it holds that

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

for some ϵ close to 0, where experiment SecExp is detailed below:

$\text{SecExp}_{\text{dSS,A}}(1^\sigma)$
1. $(pk, sk) \leftarrow \text{KG}(1^\sigma)$
2. $(m', sig') \leftarrow A^{\text{dSign}(pk,sk,\cdot),S(\cdot)}(pk)$
3. Let Q be the set of message queries made by A to oracle $\text{dSign}(pk, sk, \cdot)$
4. if $m' \in Q$ or $\text{Ver}^S(pk, sk, m', sig') = 0$ then **return:** 0
   else **return:** 1.

5.3 Delegated Signature Schemes: a general result

We show the relationship between non-delegated signature schemes, delegation protocols and delegated signature schemes in the following theorem.

Theorem 9. Let F be a function, and (C, S) be a delegation protocol for F, and let S be the (F, C, S)-associated server oracle. Also, let SS = (KG,Sign,Ver) be a (non-delegated) signature scheme and let dSS = (S,KG,Sign^S,Ver^S) be the (F, C, S)-compatible delegated signature scheme. If SS satisfies δ-correctness and ϵ-unforgeability under chosen message attack, then dSS satisfies δ'-correctness and ϵ'-unforgeability under chosen message attack, for $\delta' = \delta$ and $\epsilon' = \epsilon$.

The main takeaway from Theorem 9 is to provide a shortcut to provably turn a conventional signature scheme into a delegated signature scheme, as defined in Section 5.2: just design a suitable delegation protocol, as defined in Section 3, for a function F of interest in the computation or verification of a signature. In particular, the delegated signature scheme comes with protection of the original signature scheme against more powerful attacks such as eavesdropping on the delegation protocol messages, and querying the server oracle.

Critical to establish the relationship in the theorem is the delegation protocol's simulation-based privacy property. First of all, we observe that the correctness property of the delegated signature scheme directly follows from the analogue property of the original signature scheme. Then, we show that the unforgeability of the delegated signature scheme follows by the unforgeability of the non-delegated signature scheme and the delegation protocol's simulation-based privacy property. Specifically, assume an adversary A is able to violate the unforgeability of the delegated signature scheme. One can construct an adversary A' that violates the unforgeability of the non-delegated signature scheme, as follows:

1. A' runs algorithm A and processes A's queries as follows
2. When A queries $dSign(pk, sk, \cdot)$ with message m, A' does the following:
   A' queries $Sign(pk, sk, \cdot)$ with message m, thus obtaining signature sig
   A' runs simulator Sim to obtain the transcripts $\{tr\}$ containing
   queries to S and replies from S performed during
   the executions of algorithms Sign^S and Ver^S
   A' simulates the oracle $dSign(pk, sk, \cdot)$'s answer as $(sig, \{tr\})$
3. When A queries $S(\cdot)$ with message $qmes_C$, A' does the following:
   A' runs S on input query message $qmes_C$ thus obtaining answer $qans_S$
   A' simulates the oracle $S(\cdot)$'s answer as $qans_S$

We note that the simulation-based privacy of the delegation protocol for F implies that the the success of A' in breaking SS is the same as the success of A in breaking dSS. The theorem follows.

5.4 Delegating ElGamal, Schnorr and Okamoto's Schemes

In this section we show delegated signature protocols for 3 well-known signature schemes: those by El Gamal [27], Schnorr [41] and Okamoto [39]. In each case, the delegated signature scheme, denoted as dSS, is obtained by combining the non-delegated signature scheme, denoted as SS and reviewed in Appendix B, with the delegation protocol (C, S) for a product of exponentiations in the associated group, described in Section 4, and then applying Theorem 9. In all considered non-delegated signature schemes, the online complexity of the signature generation and verification process is dominated by a product of 2 or 3 fixed-base exponentiations. In the design of each dSS scheme, we replace each of these products with an execution of protocol (C, S) and also carefully split the signature and verification computations between offline and online phases of the two algorithms. For uniformity of presentation and performance evaluation, both our review of the non-delegated signature scheme in Appendix B and the presentation of our delegated signature schemes in the rest of this section use as example group $G = (\mathbb{Z}_p^\star, \cdot \mod p)$, where $p = 2q + 1$, for p, q primes. We obtain that with respect to this group, our delegated schemes improve the online complexity of the signature generation and verification process by a factor between 30 and 50 over the non-delegated version and by a factor between 2 and 3 over the delegation approach built on [21], as detailed in Table 4.

Table 4: Comparison of the number of C's online multiplications in non-delegated and delegated versions of signature schemes in [27, 39, 41], all defined in group $G = (\mathbb{Z}_p^\star, \cdot \mod p)$, where $p = 2q + 1$, and p, q are primes.

Schemes	$F_{g_1,\ldots,g_m,q}$ No delegation		$F_{g_1,\ldots,g_m,q}$ With delegation	
	nPoExp	*fPoExp*	*nDelPoExp*	Our result
ElGamal	8, 198	2, 394	525	267
Schnorr	8, 193	2, 389	520	262
Okamoto	12, 290	2, 521	780	262

Delegated El Gamal Signature Scheme. Let (C, S) be the client-server protocol from Section 4 for the delegation of the function $F_{g_1,g_2,q}$ computing the product of two exponentiations over group G. Also, by (C_{inv}, S_{inv}) we denote the client-server protocol from Section 3 of [13] for the delegation of the function computing an inverse of an element in $\mathbb{Z}_q$. Finally, let H denote a cryptographic hash function. Our delegated version of the signature scheme in [27] goes as follows.

1. *Key generation:* Let g be a generator of the q-order subgroup of group G. Randomly choose $x \in \{0, \ldots, q-1\}$ and set $y := g^x \mod p$. The public key is (p, q, g, y) and the private key is x.

2. *Offline Signing:* on input public key (p, q, g, y) and private key x, choose random $k \in \{0, \ldots, q-1\}$ such that $\gcd(k, q) = 1$ and set $r := (g^k \mod p) \mod q$. Output offline signature (r).
3. *Online Signing:* on input public key (p, q, g, y), private key x, offline signature (r) and a message m, compute $s := k^{-1}(H(m) - xr) \mod q$ and output signature (r, s) if $0 < r < q$ and $0 < s < q$ or $\perp$ otherwise.
4. *Offline Verifying:* on input a public key (p, q, g, y), run the offline phase of the delegation protocol (C, S) with inputs $g_1 = g$ and $g_2 = y$, resulting in offline output pp.
5. *Online Verifying:* on input a public key (p, q, g, y), offline output pp, a message m, and a signature (r, s) with $0 < r < q$ and $0 < s < q$, run the following instructions. First, compute $u = s^{-1} \mod q$. Then, compute $x_1 = H(m)u \mod q$ and $x_2 = -ru \mod q$, query S with inputs $g_1 = g, g_2 = y, x_1$ and x_2, and use S's reply to compute the product π. Finally, check that $\pi = r \mod p$. In a slightly improved version, we observe that the computation of u could also be delegated using protocol (C_{inv}, S_{inv}).

Note that in the scheme the verification algorithm checks whether

$$g^{H(m)s^{-1}} y^{-rs^{-1}} = r \mod p,$$

which is equivalent to the check

$$g^{H(m)} = y^r r^s \mod p$$

in the original ElGamal's scheme. We also note that contrarily to the original scheme, in the above there is a negligible probability (when $r = 0$ or $s = 0$) that Sign does not compute a valid signature.

The delegated Schnorr Signature Scheme. Our delegated version of the signature scheme in [41], described below, uses a cryptographic hash function H and a client-server protocol (C, S) for the delegation of function $F_{g_1,g_2,q}$ computing the product of two exponentiations over group G.

1. *Key generation:* Let g be a generator of the q-order subgroup of group G. Randomly choose $x \in \mathbb{Z}_q$ and set $y := g^x \mod p$. The public key is (G, p, q, g, y) and the private key is x.
2. *Offline Signing:* on input public key (G, p, q, g, y) and private key x, choose random $k \in \mathbb{Z}_q$ and set $I := g^k \mod p$. Output offline signature I.
3. *Online Signing:* on input public key (G, p, q, g, y), private key x, offline signature I and a message m, compute $r := H(I, m)$ and $s := rx + k \mod q$. Output signature (r, s).
4. *Offline Verifying:* on input a public key (G, p, q, g, y), run the offline phase of the delegation protocol (C, S) resulting in offline output pp.
5. *Online Verifying:* on input public key (G, p, q, g, y), a message m, offline output pp and signature (r, s), set $x_1 = s$ and $x_2 = -r \mod q$, query S with inputs $g_1 = g, g_2 = y, x_1$ and x_2, and use S's reply to compute the product π. Finally, check that $H(\pi, m) = r \mod p$.

The Delegated Okamoto Signature Scheme. Our delegated version of the signature scheme in [39], described below, uses a cryptographic hash function H and a client-server protocol (C, S) for the delegation of function $F_{g_1,g_2,g_3,q}$, computing the product of three exponentiations over group G.

1. *Key generation:* Let $p = 2q+1$ where p, q are primes, let g_1 and g_2 be generators for the q-order subgroup of group G, and let t be a sufficiently large integer; e.g., $t \geq 128$. Randomly choose $s_1, s_2 \in \mathbb{Z}_q$ and set $v := g_1^{-s_1} \cdot g_2^{-s_2} \mod p$. The public key is (p, q, g_1, g_2, t, v) and the private key is (s_1, s_2).
2. *Offline Signing:* on input public key (p, q, g_1, g_2, t, v) and private key (s_1, s_2), choose random $r_1, r_2 \in \mathbb{Z}_q$, set $x := g_1^{r_1} \cdot g_2^{r_2} \mod p$ and output offline signature x.
3. *Online Signing:* on input public key (p, q, g_1, g_2, t, v) and private key (s_1, s_2), offline signature x and a message m, compute $e := H(x, m) \in \mathbb{Z}_{2^t}$, followed by (y_1, y_2) such that $y_1 = r_1 + es_1 \mod q$ and $y_2 = r_2 + es_2 \mod q$. Output signature (e, y_1, y_2).
4. *Offline Verifying:* on input a public key (p, q, g_1, g_2, t, v), run the offline phase of the delegation protocol (C, S) resulting in offline output pp.

5. *Online Verifying:* on input a public key (p, q, g_1, g_2, t, v), offline output pp, a message m, and signature (e, y_1, y_2), set $x_1 = y_1$, $x_2 = y_2$, $x_3 = e$, query S with inputs g_1, g_2 and $g_3 = v$, and use S's reply to compute the product π. Finally, check that $H(\pi, m) = e \mod p$.

6 Conclusions

We considered the problem of delegating the computation of a product of group exponentiations to a single, possibly malicious, server. We solved this problem by showing a protocol that provably satisfies formal correctness, privacy, security and efficiency requirements, in a large class of cyclic groups; specifically, cyclic groups whose multiplication and inverse operations can be efficiently computed, and which admit an efficiently verifiable protocol to prove that an element is in the group. The considered class of cyclic groups includes groups often discussed in cryptography literature, such as prime-order subgroups in $\mathbb{Z}_p$ and elliptic curve groups.

As an application, we showed the first private, secure and efficient delegated (to a single, possibly malicious, server) versions of an entire cryptographic scheme. Previous research only achieved these properties for delegation of a single operation in a scheme's algorithm. Specifically, we showed delegated versions of well-known signature schemes whose most expensive computations could be rephrased as products of exponentiations over cyclic groups. This implies that in any delegated model of computation, certain well-known signature schemes can be run with improvements of 1-2 orders of magnitude in the online runtime for the entire signature generation and verification process.

Moreover, we believe that our methods provide hope towards private, secure and efficient delegation of more complex cryptographic protocols to a single, possibly malicious, server.

References

[1] A. Arbit, Y. Livne, Y. Oren, and A. Wool, *Implementing Public-Key Cryptography on Passive RFID Tags is Practical*. In: International Journal of Information Security, 14(1): pp. 85-99, 2015.

[2] M. Atallah, K. Pantazopoulos, J. Rice, and E. Spafford, *Secure Outsourcing of Scientific Computations*. In: Advances in Computers, 54, pp. 215–272, 2002.

[3] M. Atallah and K. Frikken, *Securely Outsourcing Linear Algebra Computations*. In: Proc. of 5th ACM ASIACCS, 2010, pp. 48–59.

[4] P. Barrett, *Implementing the Rivest Shamir and Adleman Public Key Encryption Algorithm on a Standard Digital Signal Processor*. In: Proc. of CRYPTO 1986, LNCS 263, pp. 311–323.

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

[6] M. Bellare, J. Garay, and T. Rabin, *Fast Batch Verification for Modular Exponentiation and Digital Signatures*. In: Proc. of Eurocrypt 1998, pp. 236–250, LNCS, Springer.

[7] A. Brauer, *On Addition Chains*. In: Bulletin of the American Mathematical Society, vol. 45, pp. 736–739, 1939.

[8] D. Benjamin and M. Atallah, *Private and Cheating-Free Outsourcing of Algebraic Computations*. In: Proc. of 6th PST 2008, Springer, pp. 240–245.

[9] V. Boyko, M. Peinado, and R. Venkatesan, *Speeding up Discrete Log and Factoring Based Schemes via Precomputations*. In: Proc. of Eurocrypt 1998, pp. 221–235, LNCS, Springer.

[10] E. Brickell, D. Gordon, Z. Mccurley, and D. Wilson, *Fast Exponentiation with Precomputation*. In: Proc. of Eurocrypt 92, LNCS 658, Springer.

[11] D. Bernstein, *Pippenger's Exponentiation Algorithm*, online source, 2002.

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

[13] B. Cavallo, G. Di Crescenzo, D. Kahrobaei, and V. Shpilrain, *Efficient and Secure Delegation of Group Exponentiation to a Single Server*. In: Proc. of International Workshop on Radio Frequency Identification: Security and Privacy Issues: pp. 156–173, LNCS, Springer.

[14] X. Chen, J. Li, J. Ma, Q. Tang, and W. Lou, *New Algorithms for Secure Outsourcing of Modular Exponentiations*. In: Proc. of ESORICS 2012, pp. 541-556, LNCS, Springer.

[15] C. Chevalier, F. Laguillaumie, and D. Vergnaud, *Privately Outsourcing Exponentiation to a Single Server: Cryptanalysis and Optimal Constructions*. In: Proc. of ESORICS 2016, pp. 261-278, LNCS, Springer.
[16] K. Chung K, Y. Kalai, and S. Vadhan, *Improved Delegation of Computation using Fully Homomorphic Encryption*. In: Proc. of CRYPTOO 2010, pp. 483–501, LNCS 6223, Springer, 2010.
[17] R. Cramer and V. Shoup, *Design and Analysis of Practical Public-Key Encryption Schemes Secure against Adaptive Chosen Ciphertext Attack*. In: SIAM Journal on Computing, 33(1): pp. 167–226, 2003.
[18] B. Cubaleska, A. Rieke, and T. Hermann, *Improving and Extending the Lim/Lee Exponentiation Algorithm*. In: Proc. of SAC 1999, pp. 163–174, LNCS, Springer.
[19] 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, LNCS 10931, Springer.
[20] G. Di Crescenzo, M. Khodjaeva, D. Kahrobaei, and V. Shpilrain, *Computing Multiple Exponentiations in Discrete Log and RSA Groups: From Batch Verification to Batch Delegation*. In: Proc. of 3rd IEEE Workshop on Security and Privacy in the Cloud, IEEE, 2017.
[21] G. Di Crescenzo, M. Khodjaeva, D. Kahrobaei, and V. Shpilrain, *Practical and Secure Outsourcing of Discrete Log Group Exponentiation to a Single Malicious Server*. In: Proc. of 9th ACM Cloud Computing Security Workshop (CCSW), pp. 17-28, 2017.
[22] G. Di Crescenzo, M. Khodjaeva, D. Kahrobaei, and V. Shpilrain, *Secure Delegation to a Single Malicious Server: Exponentiation in RSA-type Groups*. In: Proc. of 5th IEEE Workshop on Security and Privacy in the Cloud, IEEE, 2019.
[23] W. Diffie and M. E. Hellman, *New Directions in Cryptography*. In: IEEE Transactions on Information Theory 22(6): 644-654 (1976).
[24] M. Dijk, D. Clarke, B. Gassend, G. Suh, and S. Devadas, *Speeding Up Exponentiation using an Untrusted Computational Resource*. In: Designs, Codes and Cryptography, 39 (2), pp. 253-273, 2006.
[25] V. Dimitrov, G. Jullien, and W. Miller, *An Algorithm for Modular Exponentiation*. In: Information Processing Letters, 66(3): pp. 155–159, 1998.
[26] Y. Ding, Z. Xu, J. Ye, and K. Choo, *Secure Outsourcing of Modular Exponentiations under Single Untrusted Programme Model*. In: Journal of Computer and System Sciences, vol. 90, C, Academic Press, Inc., pp. 1-13, 2017.
[27] T. El Gamal, *A Public Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms*. In: IEEE Transactions on Information Theory, 31(4): 469-472 (1985).
[28] D. Fiore and R. Gennaro, *Publicly Verifiable Delegation of Large Polynomials and Matrix Computations, with Applications*. In: Proc. of ACM CCS Conference 2012, pp. 501–512.
[29] R. Gennaro, C. Gentry, and B. Parno, *Non-interactive Verifiable Computing: Outsourcing Computation to Untrusted Workers,* in Proc. of CRYPTO 2010, LNCS 6223, pp. 465–482.
[30] C. Gentry, *Fully Homomorphic Encryption using Ideal Lattices*. In: Proc. of ACM STOC 2009, pp. 169–178.
[31] S. Hohenberger and A. Lysyanskaya, *How to Securely Outsource Cryptographic Computations*. In: Proc. of TCC 2005, pp. 264–282, Springer.
[32] M. Jakobsson and S. Wetzel, *Secure Server-Aided Signature Generation*. In: Proc. of PKC 2001, pp. 383-401, LNCS, Springer.
[33] C. Lim and P. Lee, *More Flexible Exponentiation with Precomputation*. In: Proc. of CRYPTO 1994, pp. 95–107, LNCS, Springer.
[34] X. Ma, J. Li, and F. Zhang, *Outsourcing Computation of Modular Exponentiations in Cloud Computing*. In: Cluster Computing (2013) 16:787-796 (also INCoS 2012).
[35] T. Matsumoto, K. Kato and H. Imai, *An Improved Algorithm for Secure Outsourcing of Modular Exponentiations*. In: Proc. of CRYPTO 1988, pp. 497–506, LNCS, Springer.
[36] A. Menezes, P. C. van Oorschot, and S. A. Vanstone, *Handbook of Applied Cryptography*. In: CRC Press, 1996, ISBN 0-8493-8523-7.
[37] B. Möller, *Improved Techniques for Fast Exponentiation*. In: Proceedings of ICISC 2002, pp. 298-312, LNCS 2587, Springer.
[38] P. Nguyen, I. Shparlinski, and J. Stern, *Distribution of Modular Sums and the Security of the Server Aided Exponentiation*. In: Proceedings of Cryptography and Computational Number Theory, pp. 331-342, Springer, 2001.
[39] T. Okamoto *Provably Secure and Practical Identification Schemes and Corresponding Signature Schemes*. In: Proc. of CRYPTO 1992, pp. 31-53, LNCS, Springer.
[40] N. Pippenger, *On the Evaluation of Powers and Monomials*. In: SIAM Journal of Computing, vol. 9, pp. 230-250, 1980.
[41] C. Schnorr, *Efficient Signature Generation by Smart Cards*. In: Journal of Cryptology 4(3), pp. 161–174, 1991.
[42] E. Straus, *Addition Chains of Vectors (problem 5125)*. In: American Mathematical Monthly, vol. 70, (1973), pp. 907-913.
[43] Y. Wang, Q. Wu, D. Wong, B. Qin, S. Chow, Z. Liu, and X. Tao, *Securely Outsourcing Exponentiations with Single Untrusted Program for Cloud Storage*. In: Proc. of ESORICS 2014, pp. 326-343, LNCS, Springer.
[44] A. Yao, *On the Evaluation of Powers*. In: SIAM Journal on Computing 5(1): pp. 100–103, 1976.
[45] A. Yao, *Protocols for Secure Computations*. In: Proc. of 23rd IEEE FOCS, pp. 160–168, 1982.
[46] L. Zhao, M. Zhang, H. Shen, Y. Zhang, and J. Shen, *Privacy-Preserving Outsourcing Schemes of Modular Exponentiations Using Single Untrusted Cloud Server*. In: KSII Transactions on Internet & Information Systems, 11 (2), 2017.

A Delegation of a Single Fixed-Base Exponentiation

Let $(G, \star)$ be a cyclic group having order q, efficient operation, efficiently computable inverses, and an efficiently verifiable membership protocol, denoted as (mProve, mVerify). Let g be a generator for G, and denote as $y = g^x$ a *(fixed-base) exponentiation (in G)*. Let $\mathbb{Z}_q = \{0, 1, \ldots, q-1\}$, and let $F_{g,q} : (\mathbb{Z}_q \times \ldots \times \mathbb{Z}_q) \to G$ denote the function that maps to each $x \in \mathbb{Z}_q$ a fixed-base exponentiations (in G). By $desc(F_{g,q})$ we denote a conventional description of the function $F_{g,q}$ that includes its semantic meaning as well as generator g, order q and the efficient algorithms computing multiplication and inverses in G. The delegation protocol for a single fixed-base exponentiation in G was formally defined in [21] as follows.

Input to S: 1^σ, 1^λ, $desc(F_{g,q})$

Input to C: 1^σ, 1^λ, $desc(F_{g,q})$, $x \in \mathbb{Z}_q$

Offline phase instructions:

1. Randomly choose $u_i \in \mathbb{Z}_q$, for $i = 0, 1$
2. Set $v_i = g^{u_i}$ and store (u_i, v_i) on C, for $i = 0, 1$

Online phase instructions:

1. C randomly chooses $b \in \{1, \ldots, 2^\lambda\}$
   C sets $z_0 := (x - u_0) \mod q$, $z_1 := (b \cdot x + u_1) \mod q$
   C sends z_0, z_1 to S
2. S computes $w_i := g^{z_i}$ and $\pi_i :=$mProve(w_i), for $i = 0, 1$
   S sends w_0, w_1, π_0, π_1 to C
3. If $x = 0$
   C returns: $y = 1$ and the protocol halts
   if mVerify$(w_i, \pi_i) = 0$ for some $i \in \{0, 1\}$, then
   C returns: $\perp$ and the protocol halts
   C computes $y := w_0 \star v_0$
   C checks that
   $y \neq 1$, also called the 'distinctness test'
   $w_1 = y^b \star v_1$, also called the 'probabilistic test'
   mVerify$(w_0, \pi_0) =$ mVerify$(w_1, \pi_1) = 1$,
   also called the 'membership test'
   if any one of these tests is not satisfied then
   C returns: $\perp$ and the protocol halts
   C returns: y

B Non-Delegated Signature Schemes

We review non-delegated signature schemes from [27, 39, 41], all defined using group $G = (\mathbb{Z}_p, \cdot \mod p)$, where $p = 2q+1$, for p, q primes. We note that G has a q-order subgroup with an easily computable generator (e.g., any quadratic residue modulo p different than 1).

B.1 ElGamal Signature Scheme

The signature scheme from [27] can be defined over group G, and using a cryptographic hash function H, as follows.

1. **Key generation:** Randomly choose primes p, q such that $p = 2q + 1$ and a generator g for the q-order subgroup of group $G = (\mathbb{Z}_p, \cdot \mod p)$. Randomly choose $x \in \{0, \ldots, q-1\}$ and set $y := g^x \mod p$. The public key is (p, q, g, y) and the private key is x.
2. **Signing:** on input private key x and a message m, randomly choose $k \in \{1, \ldots, q-1\}$, set $r := g^k \mod p$ and compute $s := k^{-1}(H(m) - xr) \mod q$. If $s = 0$ then start again. Output signature (r, s).
3. **Verifying:** on input a public key (p, q, g, y), a message m, and signature (r, s) with $0 < r < q$ and $0 < s < q$, only accept the signature if

$$g^{H(m)} = y^r r^s \mod p.$$

B.2 The Schnorr Signature Scheme

The signature scheme [41] can be defined over group G, and using a cryptographic hash function H, as follows.

1. **Key generation:** Randomly choose primes p, q such that $p = 2q + 1$ and a generator g for the q-order subgroup of group $G = (\mathbb{Z}_p, \cdot \mod p)$. Randomly choose $x \in \mathbb{Z}_q$ and set $y := g^x \mod p$. The public key is (p, q, g, y) and the private key is x.
2. **Signing:** on input private key x, public key (p, q, g, y) and a message m, randomly choose $k \in \mathbb{Z}_q$ and compute $I := g^k \mod p$, $r := H(I, m)$ and $s := rx + k \mod q$. Output signature (r, s).
3. **Verifying:** on input a public key (p, q, g, y), a message m, and signature (r, s), compute $I := g^s \cdot y^{-r} \mod p$ and only accept the signature if

$$H(I, m) = r.$$

B.3 The Okamoto Signature Scheme

The signature scheme from [39] can be defined over group G, , and using a cryptographic hash function H, as follows.

1. **Key generation:** Randomly choose primes p, q such that $p = 2q + 1$ and two generators g_1, g_2 for the q-order subgroup of group $G = (\mathbb{Z}_p, \cdot \mod p)$. Also, let λ be a statistical parameter; e.g., $\lambda = 128$. Randomly choose $s_1, s_2 \in \mathbb{Z}_q$ and set $v := g_1^{-s_1} \cdot g_2^{-s_2} \mod p$. The public key is $(p, q, g_1, g_2, 1^\lambda, v)$ and the private key is (s_1, s_2).
2. **Signing:** on input private key (s_1, s_2) and a message m, randomly choose $r_1, r_2 \in \mathbb{Z}_q$ and compute $x := g_1^{r_1} \cdot g_2^{r_2} \mod p$, $e := H(x, m) \in \mathbb{Z}_{2^t}$, $y_1 = r_1 + es_1 \mod q$ and $y_2 = r_2 + es_2 \mod q$. Output signature (e, y_1, y_2).
3. **Verifying:** on input a public key (p, q, g_1, g_2, t, v), a message m, and signature (e, y_1, y_2), compute $x := g_1^{y_1} \cdot g_2^{y_2} \cdot v^e \mod p$ and only accept the signature if

$$H(x, m) = e.$$