Efficient and Secure Delegation of Group Exponentiation to a Single Server

Bren Cavallo[1], Giovanni Di Crescenzo[2], Delaram Kahrobaei[3], and Vladimir Shpilrain[3]

[1] Graduate Center, City University of New York
bcavallo@gc.cuny.edu
[2] Applied Communication Sciences
gdicrescenzo@appcomsci.com
[3] City University of New York
DKahrobaei@gc.cuny.edu
shpil@groups.sci.ccny.cuny.edu *

Abstract. We consider the problem of delegating computation of group operations from a computationally weaker client holding an input and a description of a function, to a *single* computationally stronger server holding a description of the same function. Solutions need to satisfy natural correctness, security, privacy and efficiency requirements. We obtain delegated computation protocols for the following functions, defined for an *arbitrary* commutative group:

1. Group inverses, with security and privacy holding against any computationally unrestricted malicious server.
2. Group exponentiation, with security and privacy holding against any computationally unrestricted "partially honest" server.
3. Group exponentiation, with security and privacy holding against any polynomial-time malicious server, under a pseudorandom generation assumption, and security holding with constant probability.

1 Introduction

Efficient implementation of cryptographic protocols on RFID tags is a challenging research area, due to their limited power, storage and computational resources. Among all types of cryptographic protocols, public-key, or asymmetric, protocols are especially hard to deal with, because of their computational demands, typically higher than their symmetric, or private-key, counterparts. An active research direction addressing these challenges consists of modifying known cryptographic protocols into a lightweight version that can be executed by computationally weaker devices, such as RFID tags. Another (perhaps dual,

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

in some sense) research direction addresses these challenges by delegated (also called outsourced, or server-aided) computation of cryptographic primitives. In this latter direction, a computationally weaker client, holding an input and a description of a function, wants to delegate much of the computation to one or more computationally stronger servers holding a description of the same function. Solutions typically come with the following requirements: correctness (i.e., if the client and servers are honest, the client obtains the output of the function evaluated on its input), security (i.e., servers cannot convince the client of a false computation output), privacy (i.e., servers cannot learn any information about the client's input), and efficiency (i.e., the client's running time is much smaller than the servers' running time or the time to non-interactively compute the function). In this paper, we study delegated computation to a *single* server of an operation that is a component of a large number of cryptographic primitives and/or protocols: exponentiation in a (multiplicative) group.

Related Work. Delegating computation has been and continues to be an active topic, with the increased importance of new computation paradigms, such as computing with low-power (including RFID) devices, cloud computing, etc. A number of solutions had been proposed and then broken in follow-up papers.

An elegant solution for a client to provably delegate any polynomial-time circuit to a single (semi-honest) server was given in [5], using garbled circuits [11] and fully-homomorphic encryption. Due to its generality, this solution is asymptotically efficient, but not so in practical settings, such as low-power devices.

Solutions that provably and efficiently delegate variable-base exponentiation were given in [6, 3, 4, 10] under a pseudo-random powers generation assumption, in turn based on the hidden-subset-sum hardness assumption [2, 9], and [8], under the (stronger) subset sum hardness assumption.

In [6], motivated by efficiency for RFID tags, the authors presented a protocol where a client only performs $O(\log^2 \ell)$ multiplications, where ℓ is the bit length of the exponent, to detect, with constant probability, an untrusted response from either two servers, of which at most one is untrusted, or from one server, which is trusted on average inputs. A different solution with two servers and similar features, but improved constants for client running time, was later given in [3], who explicitly posed the open problem of provable and efficient delegation of modular exponentiation to a single, untrusted, server. Batch exponentiation was first studied in [7] and later improved in [8], where a client can delegate t exponentiations to servers, by performing $O(t + \ell)$ modular multiplications, assuming the hardness of the subset sum problem. In [10] the authors present protocols to delegate variable-exponent, variable-base exponentiation to a single, untrusted, server. In [4], the authors provide efficient and secure solutions, where the base is known to the server (thus, not addressing our privacy requirement).

Our Contributions. In this paper we first of all provide *rigorous definitions* for the requirements of correctness, privacy, security and efficiency for delegated computation protocols in the single, malicious, server model. We then construct three protocols that provably satisfy these requirements while delegating, to a *single* malicious server, computations over *any* arbitrary commutative group G.

Although our main objective is to delegate group exponentiation, our first protocol is a simple protocol for the delegation of group inverses, where both security and privacy hold against a computationally unrestricted, malicious, server. We were surprised to notice that the need for such a protocol has been overlooked in the literature, even though almost all previous solutions (as well as ours) for delegated exponentiation do require inverse computations from the client, and even given the fact that in practical implementations, non-delegated inverse computations are typically as expensive as 0.25 times a non-delegated exponentiation. Our protocol for inverses works for any (even non-commutative) group, only requires the client to perform 3 group multiplications, and satisfies both security (with negligible detection error) and privacy *even in the presence of a computationally unrestricted adversary corrupting the server.*

Our second protocol is for the delegation of group exponentiation to a single, "partially-honest", server. This protocol only requires the client to perform $2m + 4$ group multiplications, where m can be any super-logarithmic and sublinear function in $\ell = \log |G|$, and satisfies both security (in a one-wayness sense, with negligible detection error) and privacy *even against a computationally unrestricted adversary corrupting the server.* Proving the security of this protocol also requires establishing a new lemma of independent interest about the number of distinct group products. No previous protocol was proved to achieve these strong security or privacy guarantees, even against a trusted server.

Our third protocol is for the delegation of group exponentiation to a single, malicious, server. This protocol only requires the client to perform $2m_r + 5$ group multiplications, where m_r is the number of multiplications required by the pseudo-random power generator (recommended to be set to $O((\log \ell)^2)$), and satisfies both security (with constant detection error) and privacy under the hidden-subset-sum assumption, as used in [2, 9]. This protocol improves over previous solutions with a single, untrusted, server *by requiring no inverses and reducing the number of client multiplications*, as detailed in Table 1.

	Ma et. al. [8]	Wang et. al. [10]	Our Third Protocol
Mod Mult	$O(\log k)$	$7m_r + 12 + O(\log \ell)$	$2m_r + 5$
Mod Inv	2	4	0

Table 1. Comparison of our third protocol with previous similar solutions delegating variable-base exponentiation to a single, untrusted, server.

Our protocols are written so to delegate function $F_{G,exp,k}(x) = x^k$ (i.e., fixed-exponent, variable-base exponentiation), but can be reformulated so to delegate function $F_{G,exp,x}(k) = x^k$ (i.e., variable-exponent, fixed-base exponentiation).

2 Definitions and Preliminaries

In this section we formally define delegation protocols, and their correctness, security, privacy and efficiency requirements (building on the approaches in [5] and [6]). We start with some basic notations.

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 algorithm A, taking as input $x_1, x_2, \ldots$ and any necessary random coins, and algorithm B, taking as input $y_1, y_2, \ldots$ and any necessary random coins, where tr is the sequence of messages exchanged by A and B as a result of this execution, and z_A, z_B are A and B's final outputs, respectively.

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 a server's ones, and therefore clients are interested in delegating (or outsourcing) the computation of specific functions to servers. In all our solutions, we consider a single *client*, denoted as C, and a single *server*, denoted as S.

Let σ be a security parameter, expressed in unary notation (i.e., 1^σ), and 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 F is known to both C and S, and input x is known only to C, we define a *delegated protocol for the computation of* F as a 2-party communication protocol between C and S, denoted as $(C(1^\sigma, desc(F), x), S(1^\sigma, desc(F)))$. A *delegated computation of the value* $y = F(x)$ is an execution, using independently chosen random bits for C and S, of the above delegated protocol, and is denoted as

$$(y_C, y_S, tr) \leftarrow (C(1^\sigma, desc(F), x), S(1^\sigma, desc(F)),$$

meaning that at the end of the execution C learns $y_C = y$, S learns y_S (usually an empty string in this paper), and tr is the transcript of the communication exchanged between C and S. (We will often omit $desc(F)$ and 1^σ for brevity of description.) Executions of delegated protocols can happen *sequentially* (each execution starting after the previous one is finished) or *concurrently* (S runs at the same time one execution with each one of many clients). [4]

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 equal to the value obtained by evaluating function F on C's input. A formal definition follows.

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

$$\text{Prob}\left[\,(y_C, y_S, tr) \leftarrow (C(1^\sigma, desc(F), x), S(1^\sigma, desc(F))) : y = F(x)\,\right] \;=\; 1.$$

[4] We assume that the communication link between each client and S is private or not subject to confidentiality, integrity, or replay attacks, and note that such attacks can be separately addressed using known techniques in cryptography and security.

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 and more realistic security requirement, we augment the adversary's power so that the adversary can even choose C's input x, and even take part in a polynomial number of protocol executions, with inputs again chosen by the adversary, before attempting to convince C of an incorrect output.

We also define a natural "partially-honest variant" of this definition, where the adversary can arbitrarily choose the inputs to all protocol executions but can only honestly run the protocols.

For simplicity, we only consider sequential protocol executions, but note that the definition adapts naturally to the case of concurrent protocol executions. A formal definition follows.

Definition 2. Let σ be a security parameter, and F be a function, and let (C, S) be a server-aided protocol for the computation of F.

We say that (C, S) satisfies (t_s, ϵ_s)*-security against a malicious adversary* if for any algorithm A running in time t_s, it holds that

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

for some small ϵ_s, where experiment *SecExp* is detailed below.

We say that (C, S) satisfies (t_s, ϵ_s)*-security against a partially-honest adversary* if for any algorithm A running in time t_s, it holds that

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

for some small ϵ_s, where experiment *phSecExp* is detailed below.

```
$\text{SecExp}_{\text{F,A}}(1^\sigma)$
1. $i = 1$
2. $(a, x_1, aux) \leftarrow A(1^\sigma, desc(F))$
3. while $(a \neq \text{`}attack')$ do
      $(y_i, (a, x_{i+1}, aux), tr_i) \leftarrow (C(x_i), A(aux))$
      $i = i + 1$
4. $x \leftarrow A(aux)$
5. $(y', aux, tr_i) \leftarrow (C(x), A(aux))$
6. if $y' \neq \perp$ and $y' \neq F(x)$ then
      **return:** 1
7. if $y' = \perp$ or $y' = F(x)$ then
      **return:** 0.
```

```
$\text{phSecExp}_{\text{F,A}}(1^\sigma)$
1. $i = 1$
2. $(a, x_1, aux) \leftarrow A(1^\sigma, desc(F))$
3. while $(a \neq \text{`}attack')$ do
      $(a, x_i, aux) \leftarrow A(aux)$
      $(y_i, \cdot, tr_i) \leftarrow (C(x_i), S)$
      $i = i + 1$
4. $x \leftarrow A(aux, tr_1, \ldots, tr_{q(\sigma)})$)
5. $(y', \cdot, tr) \leftarrow (C(x), S)$
6. if $y' \neq \perp$ and $y' \neq F(x)$ then
      **return:** 1
7. if $y' = \perp$ or $y' = F(x)$ then
      **return:** 0.
```

Privacy Requirement. Informally, the most basic privacy requirement would state the following: if C follows the protocol, a malicious adversary corrupting S cannot obtain any information about C's input x from a protocol execution. This is formalized by extending the indistinguishability-based approach used in formal definitions for encryption schemes in the cryptography literature. That is, the adversary can pick two inputs x_0, x_1, then one of these two inputs is chosen at random and used by C in the protocol with the adversary acting as S, and then the adversary tries to guess which input was used by C. To define a stronger and more realistic privacy requirement, we augment the adversary's power so that the adversary can even take part in a polynomial number of protocol executions, where it chooses C's input before attempting to guess C's input in one last execution.

We also define a natural "partially-honest variant" of this definition, where the adversary chooses the inputs to a polynomial number of protocols but can only honestly run the protocols, and later tries to guess a randomly chosen x used as input by C.

For simplicity, we only consider sequential protocol executions, but note that the definition adapts naturally to the case of concurrent protocol executions. A formal definition follows.

Definition 3. Let σ be a security parameter, and F be a function, and let (C, S) be a server-aided protocol for the computation of F.

We say that (C, S) satisfies (t_p, ϵ_p)*-privacy (in the sense of indistinguishability) against a malicious adversary* if for any algorithm A running in time at most t_p, it holds that

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

for some small ϵ_p, where experiment *PrivExp* is detailed below.

We say that (C, S) satisfies (m, ϵ_p)*-privacy (in the sense of one-wayness) against a partially-honest adversary* if for any algorithm A running in time at most t_p, it holds that

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

for some small ϵ_p, where experiment *phPrivExp* is detailed below.

```
$\text{PrivExp}_{\text{F,A}}(1^\sigma)$
1. $(a, x_1, aux) \leftarrow A(1^\sigma, desc(F))$
2. while $(a \neq \text{`}attack')$ do
     $(y_i, (a, x_{i+1}, aux), \cdot) \leftarrow (C(x_i), A(aux))$
     $i = i + 1$
3. $(x_0, x_1, aux) \leftarrow A(aux)$
4. $b \leftarrow \{0, 1\}$
5. $(y', b, tr) \leftarrow (C(x), A(aux))$
6. if $b = d$ then **return:** 1
7. if $b \neq d$ then **return:** 0.
```

```
$\text{phPrivExp}_{\text{F,A}}(1^\sigma)$
1. $(a, x_1, aux) \leftarrow A(1^\sigma, desc(F))$
2. while $(a \neq \text{`}attack')$ do
     $(a, x_i, aux) \leftarrow A(aux)$
     $(y_i, \cdot, tr_i) \leftarrow (C(x_i), S)$
     $i = i + 1$
3. $x \leftarrow Dom(F)$
4. $(\cdot, x', \cdot) \leftarrow (C(x), A(aux, tr_1, \ldots, tr_{q(\sigma)})$
5. if $F(x') = F(x)$ then **return:** 1
6. if $F(x') \neq F(x)$ then **return:** 0.
```

Efficiency Metrics and Requirements. We say that a server-aided protocol (C,S) for the computation of function F *has efficiency parameters* $(t_F, t_C, t_S))$, if F can be computed using $t_F(\sigma)$ atomic operations, C can be run using $t_C(\sigma)$

atomic operations and S can be run using $t_S(\sigma)$ atomic operations. In our analysis, we only consider group operations as atomic operations (e.g., group multiplications, inverses, and/or exponentiation), and neglect lower-order operations (e.g., equality testing between group elements). Our goal is to design protocols where $t_C(\sigma)$ is smaller than $t_F(\sigma)$ and $t_S(\sigma)$ is not significantly larger than $t_F(\sigma)$, with the following underlying assumptions, that are consistent with the state of the art in cryptographic implementations:

1. group multiplication require significantly less computing resources than group inverses;
2. group multiplication require significantly less computing resources than group exponentiation.

Naturally, we also try to minimize other typical efficiency metrics for cryptographic protocols, such as round complexity and communication complexity.

3 Delegation of Inverses

In this section we present a client-server protocol for delegated computation of group inverses. Our protocol is especially simple, works for any (even not commutative) group and for any computationally unrestricted adversary, and will be used as a subprotocol in our two protocols for delegated computation of modular exponentiation.

Notations and formal theorem statement. Let $(G, *)$ be a group, where the group operation $*$ is also referred as *multiplication*, and 1 denotes G's *identity* element. For any $a \in G$, let $b = a^{-1}$ denote the *inverse* of a; i.e., the value b such that $a * b = 1$. Let $F_{G,inv} : G \rightarrow G$ denote the function that maps every $a \in G$ to its inverse a^{-1}. Formally, we show the following

Theorem 1. There exists (constructively) a client-server protocol (C, S) for delegated computation of function $F_{G,inv}$ which satisfies

1. correctness;
2. (t_s, ϵ_s)-security (in the sense of indistinguishability) against any malicious adversary, for $t_s = \infty$ and ϵ_s negligible in σ;
3. (t_p, ϵ_p)-privacy (in the sense of indistinguishability) against any malicious adversary, for $t_p = \infty$ and ϵ_p negligible in σ;
4. efficiency with parameters (t_F, t_C, t_S), where

- t_F and t_S are $= 1$ inversion in G;
- t_C is $= 3$ multiplications in G.

We remark that Theorem 1 satisfies very strong versions of the security and privacy requirements (i.e., the adversary can arbitrarily deviate from S' program and is not even restricted to run in polynomial time), and of the efficiency requirement (t_C only requires running 3 multiplications in G). In what follows, we describe the protocol satisfying Theorem 1 and its properties.

Description of protocol (C, S). Informally speaking, the protocol claimed in Theorem 1 for delegated computation of F_{inv} goes as follows. On input $x \in G$,

C uses the group operation to mask x with a random group element, and sends the masked value to S. The latter inverts the masked element and sends it to C. Finally, C uses the group operation to check that the received value is a valid inverse for the masked value and to derive an inverse for its input x. A formal description follows.

Input to S: 1^σ, $desc(F_{G,inv})$

Input to C: 1^σ, $desc(F_{G,inv})$, $x \in G$

Protocol instructions:

1. C randomly chooses $c \in G$, computes $d = x * c$ and sends d to S;
2. S computes $e = d^{-1}$ and sends e to C;
3. C checks whether $d * e = 1$;
   if no, C returns failure symbol $\perp$;
   if yes, C computes $y = c * e$ and returns: y.

Properties of protocol (C,S). The efficiency properties are verified by protocol inspection: C runs at most 3 multiplications in G, and S runs the inversion operation in G once. With respect to round complexity, the protocol only requires one message from C to S, followed by one message from S to C. With respect to communication complexity, the protocol only requires the transfer of one group element in each of the 2 messages.

The correctness properties follows by observing that if C and S follow the protocol, C's check $d * e = 1$ is satisfied and C's output y satisfies $y = c * e = c * d^{-1} = c * c^{-1} * x^{-1} = x^{-1}$.

The security property follows by combining the following two observations: (1) on a single execution of (C, S), C's verification in step 3 forces the adversary to send a honestly computed value e in step 2; and (2) seeing multiple executions of (C, S) does not help the adversary, even when C's inputs in these executions are chosen by the adversary. Observation (1) follows by the fact that there is a single value e satisfying C's check "$e * d = 1$" and it is $e = d^{-1}$; that is, the same value that an honest S sends. Observation (2) follows from the privacy property, discussed below.

The privacy property follows by combining the following two observations: (1) on a single execution of (C, S), the message d sent by C does not leak any information about x; and (2) seeing multiple executions of (C, S) does not help the adversary, even when C's inputs in these executions are chosen by the adversary. Both observations are consequences of the fact that in each execution of (C, S), the value d is uniformly distributed in G and independent from all previous executions.

This concludes the proof of Theorem 1.

4 Delegation of Exponentiation in the Presence of a Partially-Honest Adversary

In this section we present a client-server protocol for delegated computation of group exponentiation, in the model where the adversary corrupting the server is partially honest and polynomial-time bounded. Our protocol works for any commutative group and does not rely on any additional complexity assumptions.

Notations and formal theorem statement. Let $(G, *)$ be a commutative group, let $\ell = \lceil \log |G| \rceil$ and let $b = a^k$ denote the *exponentiation* of a to the k-th power; i.e., the value $b \in G$ such that $a * \cdots * a = b$, where the multiplication operation $*$ is applied $k-1$ times. Let $k > 0$ be an integer (assumed, for simplicity, smaller than G's order), and let $F_{G,exp,k} : G \rightarrow G$ denote the function that maps every $a \in G$ to the exponentiation of a to the k-th power. Formally, we show the following

Theorem 2. Let m be a function super-logarithmic and sub-linear in ℓ. There exists (constructively) a client-server protocol (C, S) for delegated computation of function $F_{G,exp,k}$ which satisfies

1. correctness;
2. (t_s, ϵ_s)-security (in the sense of indistinguishability) against any partially-honest adversary, for $t_s = \infty$ and ϵ_s negligible in ℓ;
3. (t_p, ϵ_p)-privacy (in the sense of one-wayness) against any partially-honest adversary, for $t_s = \infty$ and $\epsilon_s = 2^{-m}+$ a quantity negligible in ℓ;
4. efficiency with parameters (t_F, t_C, t_S), where

- t_F is $= 1$ exponentiation in G;
- t_S is $= m + 1$ exponentiations and 1 inversion in G;
- t_C is $\leq 2m + 4$ multiplications in G.

We remark that Theorem 2 only considers privacy in the sense of one-wayness and partially-honest adversaries for security and privacy. In this model, it does satisfy a strong version of the security and privacy requirements, in that the adversary is not restricted to run in polynomial time. The parameter m can be set as the output of a function of ℓ, that is: (1) super-logarithmic, so to obtain an ϵ_s negligible in ℓ, and (2) sub-linear, so to obtain a t_C sub-linear in ℓ. In the rest of this section, we describe the protocol satisfying Theorem 2 and its properties.

Informal description of protocol (C, S). Informally speaking, the protocol claimed in Theorem 2 for delegated computation of $F_{G,exp,k}$ is based on the following ideas. Direct attempts to produce a protocol for group exponentiation as the natural extension of the protocol for group inverses underlying Theorem 1 fail for efficiency reasons: a small number of multiplications in G do not seem to suffice for C to derive an exponentiation for input value x from an exponentiation for a masked value produced by S. To deal with this problem, we require C to do the following: first, C asks S for a number of exponentiations of random group elements; then, C produces a masked value for x by combining it with a random subset of the previously used random group elements; finally, C obtains an exponentiation for the masked value from S and divides it by the (now known) exponentiations of the random group elements in the subset to obtain the exponentiation of its own value x. Division is delegated to S by using a few group multiplications and the inverse delegation protocol from Section 3. A formal description follows.

Formal description of protocol (C, S)**.** Let (C_{inv}, S_{inv}) denote the protocol satisfying Theorem 1 for delegated computation of inverses in group G. That is, on input a value in G to be inverted, C_{inv} returns a group value d to be sent to S_{inv}; and, on input d, S_{inv} returns a group value e to be sent to C_{inv}.

Let m be a value obtained by applying a super-logarithmic and sublinear function to ℓ, or, more practically speaking, a value such that 2^{-m} is a s sufficiently small probability and m is sufficiently smaller than ℓ. We do not further specify m to allow for security/efficiency trade-off analysis.

Input to S: 1^σ, $desc(F_{G,exp,k})$, 1^m

Input to C: 1^σ, $desc(F_{G,exp,k})$, $x \in G$, 1^m

Protocol instructions:

```
1. $C$ randomly chooses $u_1, \ldots, u_m \in G$, and sends them to $S$
2. $S$ computes $v_i = u_i^k$ and sends $v_i$ to $C$, for $i = 1, \ldots, m$;
3. $C$ randomly chooses a subset $U$ of $\{1, \ldots, m\}$;
   $C$ computes $z = x * \prod_{i \in U} u_i$ and $p = \prod_{i \in U} v_i$;
   $C$ runs $C_{inv}$ on input $p$, thus obtaining $d$
   $C$ sends $z, d$ to $S$;
4. $S$ computes $w = z^k$;
   $S$ runs $S_{inv}$ on input $d$, thus obtaining $e$;
   $S$ sends $w, e$ to $C$
5. $C$ runs $C_{inv}$ on input $p, d, e$ to compute $p^{-1}$;
   if this execution of $(C_{inv}, S_{inv})$ returned $\perp$ as output
       $C$ **returns:** $\perp$ and the protocol halts;
   $C$ computes $y = w * p^{-1}$ and **returns:** $y$
```

Properties of protocol (C,S). The efficiency properties are verified by protocol inspection: C runs $\leq 2m + 4$ multiplications in G, and S runs $m + 1$ exponentiations and 1 inversion in G. Thus, if m is sub-linear in the size of elements in G, C's running time improves by a factor of about ℓ/m over the non-delegated computation of an exponentiation in G. With respect to round complexity, the protocol only requires two rounds, each round being one message from C to S followed by one message from S to C. With respect to communication complexity, the protocol requires the transfer of $2m + 6$ group elements.

The correctness properties follows by observing that if C and S follow the protocol, C's output y is not $\perp$, by the correctness of (C_{inv}, S_{inv}), and satisfies

$$y = w * p^{-1} = z^k * \left(\prod_{i \in S} v_i\right)^{-1} = \left(x * \prod_{i \in S} u_i\right)^k * \left(\prod_{i \in S} u_i^k\right)^{-1} = x^k,$$

which implies that $y = F_{G,exp,k}(x)$ for each $x \in G$.

The security property follows by combining the following two observations: (1) in each execution of (C, S), if S follows the protocol, then the equality $y = F_{G,exp,k}(x)$ holds for each $x \in G$; (2) seeing multiple executions of (C, S) does not help the adversary violate the equality $y = F_{G,exp,k}(x)$ in a future execution, even when C's inputs in these executions are chosen by the adversary. Both observations (1) and (2) are based on the fact that correctness property holds for any $x \in G$. Moreover, observation (2) is based on the fact that a partially-honest adversary is defined to follow the protocol, even when maliciously choosing the input x for it.

We now show that the privacy property is satisfied. First, for each $x \in G$, let n_z be the number of z values that C can compute in step 3 of the protocol as the product of a random subset from the $u_1, \ldots, u_m$ values computed in step 1. Note that C can choose at most 2^m subsets U in step 3 and therefore it holds that $n_z \leq 2^m$. Then we show the following two facts: (1) when input x is randomly chosen from G, the probability that an adversary playing as S can compute x' such that $F(x') = F(x)$ at the end of a single execution of $C(, S)$, is $1/n_z$; (2) if m is super-logarithmic in ℓ, except with negligible (in ℓ) probability, it holds that $n_z = 2^m$.

To see that fact (1) holds, note that the adversary, playing as S, receives the following information from C: the m-tuple $(u_1, \ldots, u_m)$, the value d as part of the execution transcript of subprotocol (C_{inv}, S_{inv}) on input $p = \prod_{i \in U} v_i$, and the value z which directly involves x. Then we make the following observations:

1. the values $u_1, \ldots, u_m$ received by the adversary at step 1 are randomly chosen in G and thus do not leak any information about x; and
2. even conditioned on $u_1, \ldots, u_m$ and $v_1, \ldots, v_m$, because of what proved in Theorem 1, the execution of subprotocol (C_{inv}, S_{inv}) does not leak any information about p, and thus about x, to the adversary.

Given the above two observations, the only protocol value that may leak any information about x to S is z. In fact, each possible z determines exactly one possible x value, specifically $x = z * (\prod_{i \in U} u_i)^{-1}$, as the value used by C to compute z. Thus, we obtain that for each z sent by C in step 3, and conditioned on $u_1, \ldots, u_m$, $v_1, \ldots, v_m$ and the communication transcript of (C_{inv}, S_{inv}), the number of possible x that might have been used to compute z is n_z. When x is randomly chosen, this implies fact (1).

Fact (2) is proved in Appendix A.

Facts (1) and (2) imply that the probability of A guessing x' such that $F(x') = F(x)$ is $\leq 2^{-m}$ plus a negligible (in ℓ) amount.

This concludes the proof of Theorem 2.

A Protocol Extension. We can achieve a stronger privacy notion, in the sense of indistinguishability instead of one-wayness, by assuming the hardness of the subset sum problem in groups. This however imposes one further lower bound on the number m, due to ensuring that the subset sub problem is hard, which decreases the efficiency of the protocol.

5 Delegation of Exponentiation in the Presence of a Malicious Adversary

In this section we present a client-server protocol for delegated computation of group exponentiation, in the model where the adversary corrupting the server can be malicious. Our protocol works for any commutative group, and is based on a pseudo-random generation assumption, which in previous work was instantiated using the hidden-subset-sum assumption.

Notations and formal theorem statement. Let $(G, *)$ be a commutative group, let $\ell = \lceil \log |G| \rceil$, and let $k > 0$ be an integer not larger that G's order.

Let σ be a security parameter. We say that $Rand_{G,k}$ is a *pseudo-random (G, k)-powers generator* if it is a stateful probabilistic polynomial-time algorithm with the following syntax and properties:

1. on input $i = 0$, $Rand_{G,k}$ returns an auxiliary state information aux;
2. on input integer $i > 0$, and auxiliary state information aux, $Rand_{G,k}$ returns a pair (u_i, u_i^k), where $u_i \in G$, and an updated state aux;
3. for any polynomial p, the tuple $\{(u_1, u_1^k), \ldots, (u_{p(\sigma)}, u_{p(\sigma)}^k)\}$, obtained as part of the output of algorithm $Rand_{G,k}$, is computationally indistinguishable from the tuple $\{(z_1, z_1^k), \ldots, (z_{p(\sigma)}, z_{p(\sigma)}^k)\}$, where $z_1, \ldots, z_{p(\sigma)}$ are random and independent elements from G.

A generator with these properties was first designed in [2], then refined in [9], and since then used in a number of works, including previous work in outsourcing modular exponentiation (see, e.g., [6]). We recall that this generator can be designed based on the hidden-subset-sum assumption in groups. Using this same design, the running time of $Rand_{G,k}$ is comparable to about m_r group multiplications, where, based on previously recommended parameter settings, $m_r = O(\log^2 \ell)$ (see, e.g., [6]). The security parameter σ and the group element length ℓ are, in turn, typically set to be the same value.

Formally, we show the following

Theorem 3. Let σ be a security parameter, let k be a positive integer and assume the existence of a pseudo-random (G, k)-powers generator. There exists (constructively) a client-server protocol (C, S) for delegated computation of function $F_{G,exp,k}$ which satisfies

1. correctness;
2. (t_s, ϵ_s)-security against any malicious adversary, for $t_s = \text{poly}(\sigma)$ and $\epsilon_s = 1/2 + \epsilon_0$, where ϵ_0 is negligible in σ;
3. (t_p, ϵ_p)-privacy against any malicious adversary, for $t_p = \text{poly}(\sigma)$ and ϵ_p negligible in σ;;
4. efficiency with parameters (t_F, t_C, t_S), where

- t_F is $= 1$ group exponentiation in G;
- t_S is $= 2$ group exponentiations and 1 group inverse in G;
- t_C is $= 5 + 2 \cdot m_r$ multiplications in G, where m_r denotes the number of multiplications in one execution of $Rand_{G,k}$ with input > 0.

We remark that the result in Theorem 3 does not restrict to partially-honest adversaries, as done in Theorem 2, but holds for malicious adversaries, under the assumption of the existence of procedure $Rand(G, k)$. In the rest of this section, we describe the protocol satisfying Theorem 3, together with its properties.

Informal description of protocol (C, S). One approach to construct the protocol claimed in Theorem 3 for delegated computation of $F_{G,exp,k}$ could be to produce a protocol secure and private against a malicious adversary by building on the protocol secure and private against a partially-honest adversary underlying Theorem 2. Although general conversion techniques are known in the cryptography literature to transform a protocol secure against a honest adversary into one secure against a malicious adversary, these techniques do not perform well with respect to many efficiency metrics, typically because of their generality. Instead, we propose the following approach.

Instead of C delegating to S the computation of a k-th power of a random group element, as done in the protocol from Section 4, C uses the procedure $Rand_{G,k}$ to generate two pairs $(u_0, v_0), (u_1, v_1)$ of random group elements u_0, u_1 and their k-th powers v_0, v_1, respectively. Then, one of these two pairs is used to verify that answers from S are correct, and the other pair is used to mask C's input x and allow C to compute a k-th power of x, using the answers received from S. Again, as before, division is delegated by using one group operation and the inverse delegation protocol from Section 3. The privacy property follows from the fact that the message sent by C to S is computationally indistinguishable from random elements in G with their k-th powers, in turn based on the properties of $Rand_{G,k}$, and thus leaks no information about x. The security property follows from the fact that the message sent by C to S does not reveal which of the two pairs of group elements is used for verification and which is used for computation and therefore any dishonest answer from S will be detected by C with probability at least 1/2. A formal description follows.

Formal description of protocol (C, S). Let (C_{inv}, S_{inv}) denote the protocol satisfying Theorem 1 for delegated computation of inverses in group G. That is, on input a value x in G to be inverted, C_{inv} returns a group value d to be sent to S_{inv}; then, on input d, S_{inv} returns a group value e to be sent to C_{inv}; finally, based on x, d, e, algorithm C_{inv} computes value x^{-1}.

Also, let $Rand_{G,k}$ denote a pseudo-random (G, k)-powers generator. We assume that C computes $aux = Rand_{G,k}(0)$ once and at setup time, before running any delegated computation protocol.

Input to S: 1^σ, $desc(F_{G,exp,k})$

Input to C: 1^σ, $desc(F_{G,exp,k})$, $x \in G$, $aux = Rand_{G,k}(0)$

Protocol instructions:

1. C computes $(u_i, v_i, aux) = Rand_{G,k}(i, aux)$, for $i = 0, 1$;
   C randomly chooses $b \in \{0, 1\}$;
   C sets $z_b = u_b$, $z_{1-b} = x * u_{1-b}$;
   C runs C_{inv} on input v_{1-b}, thus obtaining d;
   C sends z_0, z_1, d to S;
2. S computes $w_i = z_i^k$ for $i = 0, 1$;
   S runs S_{inv} on input d, thus obtaining e;
   S sends w_0, w_1, e to C

```
3. $C$ runs $C_{inv}$ on input $t, d, e$ to compute $v_{1-b}^{-1}$;
   if this execution of $(C_{inv}, S_{inv})$ returned $\perp$ as output
           $C$ **returns:** $\perp$ and the protocol halts;
   if $w_b \neq v_b$ then
           $C$ **returns:** $\perp$ and the protocol halts;
   $C$ computes $y = w_{1-b} * v_{1-b}^{-1}$ and **returns:** $y$
```

Properties of protocol (C,S). The efficiency properties are verified by protocol inspection. With respect to round complexity, the protocol only requires one round, consisting of one message from C to S followed by one message from S to C. With respect to communication complexity, the protocol requires the transfer of 6 group elements. With respect to runtime complexity, S runs 2 exponentiation operations and 1 inversion operation in G, and C runs 2 multiplication operations in G, 1 execution of the inverse delegation protocols (requiring 3 multiplications) and 2 executions of procedure $Rand_{G,k}$.

The correctness properties follows by observing that if C and S follow the protocol, C's equality verification in step 3 will be satisfied, and thus C's output y is $\neq\perp$ and satisfies

$$y = w_{1-b} * v_{1-b}^{-1} = (x * u_{1-b})^k * ((u_{1-b})^k)^{-1} = x^k * (u_{1-b})^k * (u_{1-b})^{-k} = x^k,$$

which implies that $y = F_{G,exp,k}(x)$ for each $x \in G$.

The privacy property follows by combining the following two observations: (1) on a single execution of (C, S), the message z_0, z_1, d sent by C does not leak any information about x; and (2) seeing multiple executions of (C, S) does not help the adversary in obtaining information about the input x in a new execution, even when C's inputs in these executions are chosen by the adversary. To show observation (1), first observe that (u_0, u_1) is computationally indistinguishable from a pair of random group elements, by property 3 of the pseudo-random (G, k)-powers generator. Thus, the same holds for pair (z_0, z_1), since $z_b = u_b$ and $z_{1-b} = x * u_{1-b}$ for some $b \in \{0, 1\}$. Then, the fact that the entire message z_0, z_1, d sent by C does not leak any information about x follows from Theorem 1 and the fact that the value d (and, in fact, the entire transcript of the execution of protocol (C_{inv}, S_{inv})), do not depend on x. Because protocol (C, S) is a one-round protocol, the analysis done to show observation (1) extends across multiple executions of the same protocol, thus showing observation (2).

To see that the security property is satisfied, first consider a single execution of (C, S), where C follows the protocol, and, for any probabilistic polynomial-time adversary corrupting S, consider the values w_0, w_1, e returned by the adversary to C. The value e is associated with an execution of the inverse delegation protocol from Section 3, which is secure against any probabilistic polynomial-time adversary, as shown in Theorem 1. Thus, if the adversary deviates from the protocol in computing an e that allows C to compute v_{1-b}^{-1}, C will detect this fact and return the failure symbol $\perp$. Now, consider values w_0, w_1, and let n_A be the number in $\{0, 1, 2\}$ of $i \in \{0, 1\}$ such that $w_i = z_i^k$. We have two cases: (a) $n_A = 2$, and (b) $n_A \leq 1$. If (a) happens, then C will not return the failure symbol and can compute y as in the last line of protocol step 3, and it will satisfy

$$y = w_{1-b} * v_{1-b}^{-1} = (z_{1-b})^k * v_{1-b}^{-1} = (x * u_{1-b})^k * ((u_{1-b})^k)^{-1} = x^k,$$

On the other hand, if (b) happens, since $v_b = u_b^k$ and, assuming property 3 of the pseudo-random (G, k)-powers generator, S cannot guess random bit b, the verification $w_b = v_b$ will be passed with probability at most 1/2. Seeing multiple executions of (C, S) does not help the adversary increase this probability in the next execution, since no information is leaked to S in any execution, assuming property 3 of the pseudo-random (G, k)-powers generator, as discussed when showing the privacy property.

This concludes the proof of Theorem 3.

A Protocol Extension. We can extend protocol (C, S) to decrease the $\epsilon_s = 1/2$ in the security property by a suitable parallel repetition of it, as follows: first of all, t executions of the protocol are executed in parallel, then, in step 3, C also returns the failure symbol $\perp$ if the value y computed in step 3 is not the same in each parallel execution. The resulting protocol satisfies the security property with $\epsilon_s = 2^{-t}$, and the efficiency property with $t_C = t(5 + 2 \cdot m_r)$ multiplications in G. Thus, only small values for t can be used until the value t_C becomes as large as the number of multiplications in a non-delegated computation of $F_{G,exp,k}$.

6 Conclusions

Towards making public-key cryptography more accessible to RFID tags, we considered the problem of delegating group exponentiation to a single, untrusted, server. We showed protocols that provably satisfy formal correctness, privacy, security and efficiency requirements. With our protocols, we highlighted the importance of delegating the group inverse operation, and the possibility of achieving strong privacy and security properties against computationally unrestricted adversaries, and of further improving client computation time.

References

1. P. Barrett, *Implementing the Rivest Shamir and Adleman public key encryption algorithm on a standard digital signal processor*, in: Advances in Cryptology, CRYPTO 1986, LNCS **263** (1986), 311-323.
2. V. Boyko and M. Peinado and R. Venkatesan, *Speeding up discrete log and factoring based schemes via precomputations.* In: Advances in Cryptology, EUROCRYPT'98, pp. 221-235, Springer, 1998.
3. X. Chen and J. Li and J. Ma and Q. Tang and W. Lou, *New algorithms for secure outsourcing of modular exponentiations.* In: Computer Security–ESORICS 2012, pp. 541-556, 2012.
4. 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.

5. R. Gennaro, C. Gentry, and B. Parno, *Non-interactive verifiable computing: Outsourcing computation to untrusted workers,* in: Advances in Cryptology, CRYPTO 2010, Lecture Notes Comp. Sc. **6223** (2010), 465-482.
6. S. Hohenberger and A. Lysyanskaya, *How to securely outsource cryptographic computations.* In: Theory of Cryptography, pp. 264-282, 2005.
7. M. Jakobsson and S. Wetzel, *Secure server-aided signature generation.* In: Public Key Cryptography, pp. 383-401, Springer, 2001.
8. X. Ma and J. Li and F. Zhang, *Outsourcing computation of modular exponentiations in cloud computing.* In: Cluster Computing (2013) 16:787?796 (also INCoS 2012).
9. P. Q. Nguyen and I. E. Shparlinski and J. Stern, *Distribution of modular sums and the security of the server aided exponentiation.* In: Cryptography and Computational Number Theory, pp. 331-342, Springer, 2001.
10. Y. Wang and Q. Wu and D. Wong and B. Qin and S. Chow and Z. Liu and X. Tao, *Securely outsourcing exponentiations with single untrusted program for cloud storage.* In: Computer Security-ESORICS 2014, pp.326-343, Springer, 2014.
11. A. C. Yao, *Protocols for secure computations.* In: Proceedings of the 23rd Annual Symposium on Foundations of Computer Science, pp. 160-168, IEEE Computer Society, 1982.

A Number of Distinct Group Products

We show a proof for fact (2), which was required in Section 4 to complete the proof that the privacy property of protocol (C, S) from that section is satisfied. First, we recall the statement of fact (2): if m is super-logarithmic in σ, except with negligible (in σ) probability, it holds that $n_z = 2^m$, where n_z be the number of z values that C can compute in step 3 of the protocol as the product of a random subset from the $u_1, \ldots, u_m$ values computed in step 1. (Also, recall that $n_z \leq 2^m$.)

Let $X \subset G$ where G is a commutative group. We say that X is *not collision free* (NCF) if there exist distinct subsets $S_1, S_2 \subset X$ such that

$$\prod_{i \in S_1} i = \prod_{j \in S_2} j.$$

Alternatively if all subsets of X have distinct products, we say that X is collision free (CF).

Lemma 1. Let m be sub-linear in $\ell = \log |G|$, then the probability that a random subset $X \subset G$ where $|X| = m$ has a collision is negligible in ℓ.

Proof. Let $X = \{x_1, \cdots, x_m\}$ and $X_i = \{x_1, \cdots, x_i\}$. Then

$$\begin{aligned}\Pr(X \text{ is NCF}) &= \Pr(X \text{ is NCF} \mid X_{m-1} \text{ is CF}) * \Pr(X_{m-1} \text{ is CF}) + \\ &\qquad \Pr(X \text{ is NCF} \mid X_{m-1} \text{ is NCF}) * \Pr(X_{m-1} \text{ is NCF}) \\ &\leq \Pr(X \text{ is NCF} \mid X_{m-1} \text{ is CF}) + \Pr(X_{m-1} \text{ is NCF}) \\ &\leq \sum_{i=1}^{m} \Pr(X_i \text{ is NCF} \mid X_{i-1} \text{ is CF}) \\ &\leq \sum_{i=1}^{m} \frac{3^{i-1}}{|G|} \\ &= \frac{3^m - 1}{2|G|},\end{aligned}$$

which is negligible in ℓ as long as $m = o(\ell)$, and where the probability derivations are explained as follows.

The first equality is obtained by an application of the probability conditioning rule. The first inequality follows by upper bounding $\Pr(X_{m-1} \text{ is CF})$ with 1 and observing that, by definition, $\Pr(X \text{ is NCF} \mid X_{m-1} \text{ is NCF}) = 1$. The second inequality is obtained by iterating the first inequality to the $\Pr(X_{m-1} \text{ is NCF})$ term. The fourth inequality is obtained by a geometric summation calculation. To see how the third inequality is obtained, observe that we compute an upper bound for $\Pr(X_i \text{ is NCF} \mid X_{i-1} \text{ is CF})$ as follows. First note that the only way X_i can have a collision is if $\exists a, b$ that are products of distinct elements of X_i such that

$$ax_i = b.$$

Therefore, x_i must avoid all distinct elements of the form ba^{-1}. Let $y_j = x_j^{-1}$. Then any element of the form ba^{-1} can be written as

$$x_{j_1} \cdots x_{j_k} x_{j_{k+1}}^{-1} \cdots x_{j_l}^{-1}$$

where each element in the above product is distinct and for each x_j that appears, x_j^{-1} does not appear. There are a total of 3^{i-1} such strings and therefore at most 3^{i-1} distinct elements x_i must avoid. We then have that

$$\Pr(X_i \text{ is NCF} \mid X_{i-1} \text{ is CF}) \leq \frac{3^{i-1}}{|G|}.$$

□