Privately Outsourcing Exponentiation to a Single Server: Cryptanalysis and Optimal Constructions

Céline Chevalier[1], Fabien Laguillaumie[2], and Damien Vergnaud[3(✉)]

[1] CRED (U. Panthéon–Assas Paris II), Paris, France
[2] LIP (UCBL, U. Lyon, CNRS, ENS Lyon, INRIA), Lyon, France
[3] DI/ENS (ENS, CNRS, INRIA, PSL), Paris, France
damien.vergnaud@ens.fr

Abstract. We address the problem of speeding up group computations in cryptography using a single untrusted computational resource. We analyze the security of an efficient protocol for securely outsourcing multi-exponentiations proposed at ESORICS 2014. We show that this scheme does not achieve the claimed security guarantees and we present practical polynomial-time attacks on the delegation protocol which allow the untrusted helper to recover part (or the whole) of the device secret inputs. We then provide simple constructions for outsourcing group exponentiations in different settings (*e.g.* public/secret, fixed/variable bases and public/secret exponents). Finally, we prove that our attacks on the ESORICS 2014 protocol are unavoidable if one wants to use a single untrusted computational resource and to limit the computational cost of the limited device to a constant number of (generic) group operations. In particular, we show that our constructions are actually optimal in terms of operations in the underlying group.

1 Introduction

The problem of "outsourcing" computation has been considered in various settings since many years. The proliferation of mobile devices provides yet another venue in which a computationally weak device would like to be able to outsource a costly operation to a third party helper. Such devices do not usually have the computational or memory resources to perform complex cryptographic operations and it is natural to outsource these operations to some helper. However, in this scenario, this helper can, potentially, be operated by a malicious adversary and we usually need to ensure that it does not learn anything about what it is actually computing. The wild and successful deployment of cloud storage services make users outsource their data, for a personal or commercial purpose. These users actually have to trust their storage providers concerning the availability of their data, and indeed outages happen regularly. Cryptographic primitives are needed to convince customers that their platforms are reliable. Among such primitives, provable data possessions [1] and proofs of retrievability [16] allow

I. Askoxylakis et al. (Eds.): ESORICS 2016, Part I, LNCS 9878, pp. 261–278, 2016.
DOI: 10.1007/978-3-319-45744-4_13

the storage cloud to prove that a file uploaded by a client has not been deteriorated or that it can be entirely retrieved. The computation needed on the verification side by the client are highly *"exponentiation-consuming"*. Indeed, the core operation of these cryptosystems is group exponentiation, i.e., computing u^a from a group element u and an exponent a. The main goal of this paper is to analyze new and existing protocols outsourcing group exponentiation to a *single* untrusted helper.

Prior work. In 2005, Hohenberger and Lysyanskaya [14] provided a formal security definition for *securely* outsourcing computations from a computationally limited device to untrusted helpers and they presented two practical schemes. Their first scheme shows how to securely outsource group exponentiations to two, possibly dishonest, servers that are physically separated (and do not communicate). Their protocol achieves security as long as one of them is honest. In 2012, Chen, Li, Ma, Tang and Lou [8] presented a nice efficiency improvement to the protocol from [14], but the security of their scheme also relies on the assumption that the two servers cannot communicate.

Since this separation of the two servers is actually a strong assumption hard to be met in practice, at ESORICS 2014 [27], Wang, Wu, Wong, Qin, Chow, Liu and Tan proposed a protocol to outsource group exponentiations to a *single* untrusted server. Their generic algorithm is very efficient and allows to outsource multi-exponentiations with fixed or variable exponent and bases (that can be public or secret).

Contributions of the paper. Our contributions are both theoretical and practical. Our first result is a practical attack on the protocol for outsourcing multi-exponentiation proposed by Wang *et al.* [27]. Our attack allows to recover secret information in polynomial time using lattice reduction. It shows that their solution is completely insecure. We later show in Theorem 2 that what they expected to achieve (namely, to limit the computational cost of the limited device to a constant number of (generic) group operations) is actually theoretically impossible.

Our second contribution is the proposal of a taxonomy of exponentiation delegation protocols and the associated simple yet precise and formal models of protocols that allow a client $\mathcal{C}$ (or delegator) who wants to compute a multi-exponentiation (which is a computation of the form $\prod_{i=1}^{n} {u_i}^{a_i}$ for group elements u_i's and exponents a_i's) to delegate an intermediate exponentiation to a more powerful server $\mathcal{S}$ (or delegatee). The client's contribution in the computation is then only few multiplications of group elements and arithmetic operations modulo the underlying group order. We consider in this work only prime-order groups. Our taxonomy covers all the practical situations: the group elements can be secret or public, variable or fixed, the exponents can be secret or public, and the result of the multi-exponentiation can also be either public or secret. As an example, a BLS digital signature [4] is a group element $\sigma = h(m)^a$, where m is the signed message, h a hash function, and a the secret key. The signature computation can be delegated with our protocol for a public group element (the hashed value of the message), a secret exponent (the secret key), and a

public output (the signature). During an ElGamal decryption of a ciphertext $(c_1, c_2) = (g^r, m \cdot y^r)$ (where m is the plaintext and $y = g^a$ is the public key), one may want to securely delegate the computation of ${c_1}^a$ (to recover m as $c_2/{c_1}^a$). Such an exponentiation can be delegated with our protocol for known group element (c_1), secret exponent (a) and secret result (${c_1}^a$, in order to keep the plaintext m secret). We propose a delegation protocol for each of the previously mentioned scenarios. The latency of sending messages back and forth has been shown to often be the dominating factor in the running time of cryptographic protocols. Indeed, round complexity has been the subject of a great deal of research in cryptography. We thus focus on the problem of constructing one-round delegation protocols; i.e., where we authorize the client to call only once the server $\mathcal{S}$, and give him access to some pre-computations (consisting of pairs of the form (k, g^k)). We then consider their complexity, in terms of group operations needed by the client to eventually get the desired result securely.

Our third and main contribution is the computation of lower bounds on the number of group operations needed on the delegator's side to securely compute his exponentiation when it has access to a helper server. To give these lower bounds, we analyze the security of delegation protocols in the generic group model which considers that algorithms do not exploit any properties of the encodings of group elements. This model is usually used to rule out classes of attacks by an adversary trying to break a cryptographic assumption. We use it only to prove our lower bounds but we do not assume that an adversary against our protocols is limited to generic operations in the underlying group. As mentioned above, these lower bounds tell us that our protocols are optimal in terms of operations in the underlying group (in other words, they cannot be significantly improved).

A summary of our results for outsourcing protocols for single exponentiation is given in Table 1. For the ease of reference, all our results are collected in Table 2 given on page 16.

2 Preliminaries

Exponentiation Delegation: Definitions. The (multi-)exponentiations are computed in a group $\mathbb{G}$ whose description is provided by an algorithm $\mathsf{GroupGen}$, which takes as input a security parameter λ. It provides a set $params$ which contains the group description, its prime[1] order, say p, and one or many generators. Let n be an integer, we denote by $\mathbf{a}$ (*resp.* $\mathbf{u}$) a vector of n exponents $a_i \in \mathbb{Z}_p$ (*resp.* group elements $u_i \in \mathbb{G}$). The aim of the protocols that follow is to compute $\prod_{i=1}^{n} u_i^{a_i}$, denoted as $\mathbf{u}^{\mathbf{a}}$.

We consider a delegation of an exponentiation as a 2-party protocol between a client $\mathcal{C}$ and a server $\mathcal{S}$. We denote as $(y_{\mathcal{C}}, y_{\mathcal{S}}, tr) \leftarrow (\mathcal{C}(1^\lambda, params, (\mathbf{a}, \mathbf{u})), \mathcal{S}(1^\lambda))$ the protocol at the end of which $\mathcal{C}$ knows $y_{\mathcal{C}}$ and $\mathcal{S}$ learns $y_{\mathcal{S}}$ (usually an empty string). The string tr is the transcript of the interaction. In all our protocols, the

[1] In this paper, following prior works, we consider only prime order groups, but most of our results can be generalized to composite order groups.

Table 1. Results Summary: ℓ is the number of available pairs (k, g^k), $p = |\mathbb{G}|$ and Prot. means "Protocol" (p. 12). The given complexities are the number of operations in $\mathbb{G}$.

Outsourcing Fixed Base Exp.							Outsourcing Var. Base Exp.						
u	a	u^a	#delegations	Lower bound	Achieved Complexity	Optimality	u	a	u^a	# delegations	Lower bound	Achieved Complexity	Optimality
Pub.	Pub.	Pub.					Pub.	Pub.	Pub.				
Pub.	Pub.	Sec.					Pub.	Pub.	Sec.				
Pub.	Sec.	Pub.	1	0	0 (Prot. 2)	✓	Pub.	Sec.	Pub.	s	$\frac{\log p}{s+1}$	$\frac{\log p}{s+1}$ (Prot. 4&5)	✓
Pub.	Sec.	Sec.	1	1	1 (Prot. 3)	✓	Pub.	Sec.	Sec.	s	$\frac{\log p}{s+1}$	$\frac{\log p}{s+1}$ (Prot. 4&5)	✓
Sec.	Pub.	Pub.					Sec.	Pub.	Pub.				
Sec.	Pub.	Sec.	1	1	1 (Prot. 1)	✓	Sec.	Pub.	Sec.	1	$\frac{\log p}{\ell+3}$ *	$\frac{\log p}{\ell}$ (Prot. 8)	✗
										2	≤ 3	3 (Prot. 7)	✗
Sec.	Sec.	Pub.	1	0	0 (Prot. 2)	✓	Sec.	Sec.	Pub.	s	$\frac{\log p}{s+1}$	$\frac{\log p}{s}$ (Prot. 6)	✗
Sec.	Sec.	Sec.	1	1	1 (Prot. 1)	✓	Sec.	Sec.	Sec.	s	$\frac{\log p}{s+1}$	$\frac{\log p}{s}$ (Prot. 6)	✗

server will be very basic, since it will only perform exponentiations whose basis and exponent are sent to him by the client. In [7], Cavallo *et al.* emphasized the need for delegation of group inverses since almost all known protocols for delegated exponentiation do require inverse computations from the client. They presented an efficient and secure protocol for delegating group inverses. However, our protocols do not require such computations and our lower bounds hold even in groups in which inverse computation is efficient (and therefore does not need to be delegated, see Remark 4).

To model the security notions, and to simplify the exposition, we describe by a *computation code* $\boldsymbol{\beta}$ (which is a binary vector of length 4), the scenario of the computation. Indeed, according to the applications, some of the data on which the computations are performed may be either public or secret. In the computation of $\mathbf{u^a}$, the vector of basis $\mathbf{u}$, the vector of exponents $\mathbf{a}$ or the result $\mathbf{u^a}$ may be unknown (and especially to the adversary). The three first entries of the code describe the secrecy of respectively $\mathbf{u}$, $\mathbf{a}$ and $\mathbf{u^a}$: a 0 means that the data is hidden to the adversary, and 1 means that the data is public. The last entry indicates whether the base if fixed (f) or variable (v). For instance, the code 101v means that $\mathbf{u}$ is public, the exponent $\mathbf{a}$ is secret, and the result $\mathbf{u^a}$ is public, while the base is variable. Note that we consider the *whole* vectors (*i.e.,* all of its coordinates) to be either public or private, whereas we could imagine that, for a vector $\mathbf{u}$ of exponents for instance, some of these could be public, and others could be kept secret. The following notions should then be declined according to these scenarios.

- **Correctness.** This requirement means that when the server and the client follow honestly the protocol, the client's output is actually the expected (multi-)exponentiation.
- **One-wayness.** This natural security basically means that an attacker cannot compute any secret data involved during the computation.
- **Privacy.** This indistinguishability-based security notion [7] captures that given two secret inputs (even adversarially chosen), an "honest-but-curious" adversary cannot tell which input was used (with a probability significantly better than that of guessing).

We refer the reader to the paper full version [9] for formal definitions.

Remark 1. *As mentioned in [6, 10, 18], a delegation protocol that does not ensure verifiability may cause severe security problems. Even though our protocols are not verifiable, the computational lower bounds on the efficiency of private outsourcing exponentiation protocols we prove in Sect. 5 readily imply that these bounds also holds for verifiable protocols. In a forthcoming paper, we will show how our methods can be extended to propose verifiable delegation protocols and to improve corresponding efficiency lower bounds.*

Generic Group Model. The generic group model (see [24] for details) is an idealized cryptographic model where algorithms (generally adversaries) do not exploit any properties of the encodings of group elements. They can access group elements only via a random encoding algorithm that encodes group elements as random bit-strings. Proofs in the generic group model provide heuristic evidence of some problem hardness, but they do not necessarily say anything about the difficulty of a specific problem in a concrete group [12].

Computations of pairs (g^k, k). To outsource the computation of an exponentiation in a group $\mathbb{G}$ of prime order p, (pseudo-)random pairs of the form $(g^k, k) \in \mathbb{G} \times \mathbb{Z}_p$ are sometimes used to hide sensitive information to the untrusted server. This looks like a "chicken-and-egg problem" but there exist several techniques to make it possible for a computationally limited device to have such pairs at its disposal, at a low cost. A trivial method is to load its memory with many genuine (generated by a trusted party) random and independent couples. In other settings, a mobile device with limited computing capabilities can precompute "offline" such pairs at low speed and power. If the device can do a little more computation, there exist other preprocessing techniques, that may depend whether the base or the exponent varies.

We only mention here the main technique to produce these pairs. The key ingredient is Boyko, Peinado and Venkatesan generator from [5]: the idea is to store a small number of precomputed pairs (g^{α_i}, α_i), and when a fresh pair is needed, the device outputs a product $g^k = \prod_{i \in S} g^{\alpha_i}$ with $k = \sum_{i \in S} \alpha_i$ for a random set S. It has then been improved by Nguyen, Shparlinski and Stern generator [20], that allows to re-use some α_i in the product. This generator is secure against adaptive adversaries and performs $O(\log \log(p)^2)$ group operations. For some parameters, the generator from [20] is proved to have an output distribution

statistically close to the uniform distribution. Obviously, these generators are of practical interest only if the base g is fixed and used multiple times.

In the sequel we will assume that the delegator may have access to some *(pseudo-)random power generator* $\mathcal{B}(\cdot)$ that at invocation (with no input) outputs a single (pseudo)-random pair $(g^k, k) \in \mathbb{G} \times \mathbb{Z}_p$ where k is uniformly distributed in $\mathbb{Z}_p$ (or statistically close to the uniform distribution). If the generator $\mathcal{B}(\cdot)$ is invoked several times, we assume that the output pairs are independent. In order to evaluate the efficiency of delegation protocols, we consider explicitly the query complexity to the generator $\mathcal{B}(\cdot)$ (depending on the context, this can be interpreted as storage of precomputed values, offline computation or use of the generator from [20] and thus additional multiplications in $\mathbb{G}$).

3 Attack on Wang *et al.*'s Algorithm from ESORICS 2014

Wang *et al.* proposed a generic algorithm to outsource the computation of several multi-exponentiations with variable exponents and variable bases. Their algorithm, called GExp, takes as input a list of tuples $((\{a_{i,j}\}_{1\leq j\leq s}; \{u_{i,j}\}_{1\leq j\leq s}))_{1\leq i\leq r}$ and computes the list of multi-exponentiations $(\prod_{j=1}^{r} u_{i,j}^{a_{i,j}})_{1\leq i\leq s}$. It is claimed that this algorithm is secure in a strong model where the computation is outsourced to a single untrusted server [27, Theorem 1]. We will show that GExp can be broken in polynomial time using lattice reduction if two (simple) exponentiations are outsourced with the *same* exponent, which is the case in the scenario of proof of data possession presented in [27, Section 4]. This means that GExp does not achieve the claimed security.

Description of Wang *et al.*'s protocol. The setting of GExp is the following: $\mathbb{G}$ is a cyclic group of prime order p, and g is a generator. For $1 \leq i \leq r$ and $1 \leq j \leq s$, $a_{i,j}$ are uniform and independent elements of $\mathbb{Z}_p^*$, and $u_{i,j}$ are random elements from $\mathbb{G}$. They assume the $a_{i,j}$'s, the $u_{i,j}$'s and the result are secret (and the $u_{i,j}$ are variable, i.e. $\boldsymbol{\beta} = 000v$ with our notations). The protocol is divided into three steps:

– **Step 1.** The delegator $\mathcal{C}$ generates four random pairs $(\alpha_k, \mu_k)_{1\leq k\leq 4}$ where $\mu_k = g^{\alpha_k}$ (using a pseudo-random power generator). A Υ-bit element χ is randomly picked (for some parameter Υ). Then, for all $1 \leq i \leq r$ and $1 \leq j \leq s$, the elements $b_{i,j}$ are randomly picked in $\mathbb{Z}_p^*$. It sets[2]

$$c_{i,j} = a_{i,j} - b_{i,j}\chi \mod p \tag{1}$$

$$\theta_i = (\alpha_1 \sum_{j=1}^{s} b_{i,j} - \alpha_2) + (\alpha_3 \sum_{j=1}^{s} c_{i,j} - \alpha_4) \mod p. \tag{2}$$

and $w_{i,j} = u_{i,j}/\mu_1$ and $h_{i,j} = u_{i,j}/\mu_3$.

[2] Note that the protocol from [27] can also be described without inversion in the group $\mathbb{G}$ but to help the reader familiar with this paper, we use the same description.

– **Step 2.** The second step consists in invoking the (untrusted) delegatee $\mathcal{S}$ for some exponentiations. To do so, $\mathcal{C}$ generates (using a (pseudo-)random power generator) $r+2$ random pairs $(g^{t_i}, t_i)_{1\leq i\leq r+2}$ and queries (in random order) $\mathcal{S}$ on
  - $(g^{t_i}, \theta_i/t_i)$ to obtain $B_i = g^{\theta_i}$ for all $1 \leq i \leq r$,
  - $(g^{t_{r+1}}, \theta/t_{r+1})$ to obtain $A = g^{\theta}$ with $\theta = t_{r+2} - \sum_{i=1}^{r} \theta_i \mod p$,
  - $\begin{cases} (w_{i,j}, b_{i,j}) \text{ to get } C_{i,j} = (u_{i,j}/\mu_1)^{b_{i,j}} \\ (h_{i,j}, c_{i,j}) \text{ to get } D_{i,j} = (u_{i,j}/\mu_3)^{c_{i,j}} \end{cases}$ for $1 \leq i \leq r$ and $1 \leq j \leq s$.

– **Step 3.** It consists in combining the different values obtained from $\mathcal{S}$ to recover the desired multi-exponentiations. In particular, an exponentiation to the power χ is involved. The protocol to be efficient, needs χ not too large.

Simple attack. Suppose that a delegation of a single exponentiation u^a, for u and a secret, is performed using Wang *et al.*'s protocol. If a is a secret key, an element of the form h^a is likely to be known by the adversary, together with h (one can think of a public key in a scenario of delegation of BLS signatures [4], for instance)). In this case, as the attacker sees an element of the form $c = a - b\chi$ (see Eq. (1)) and knows b (cf. Step 2), he can compute h^c which is equal to $h^a \cdot (h^{\chi})^{-b}$, so that recovering χ can be done by computing the discrete logarithm of $(h^a/h^c)^{b^{-1}}$ in base h. Using a baby-step giant-step algorithm, this can be done in $2^{\Upsilon/2}$ operations, which contradicts [27, Theorem 1].

Main attack. The crucial weakness of this protocol is the use of this *small* element χ which hides the exponents. The authors suggest to take it of bit-size Υ, for $\Upsilon = 64$. We will show that it cannot be that small since it can be recovered in polynomial time if two exponentiations with the *same* exponent are outsourced to the server $\mathcal{S}$. The scenario of our attack is the following: two exponentiations of the form $\mathsf{GExp}((a_{1,1}, \ldots, a_{1,s}); (u_{1,1}, \ldots, u_{1,s}))$ and $\mathsf{GExp}((a_{1,1}, \ldots, a_{1,s}); (u'_{1,1}, \ldots, u'_{1,s}))$ are queried to $\mathcal{S}$. The exponentiations are computed with the *same* exponents. This is typically the case in the first application proposed in [27, Section 4.1] to securely offload Shacham and Waters's proofs of retrievability [23].

For the sake of clarity, it is sufficient to focus on the elements that mask the first exponent $a_{1,1}$. An attacker will obtain (see Step 2) $b_{1,1}$, $b'_{1,1}$, $c_{1,1}$ and $c'_{1,1}$ such that $c_{1,1} = a_{1,1} - b_{1,1}\chi \mod p$ and $c'_{1,1} = a_{1,1} - b'_{1,1}\chi' \mod p$. Subtracting these two equations gives a modular bi-variate linear equation:

$$b_{1,1}X - b'_{1,1}Y + c_{1,1} - c'_{1,1} = 0 \mod p \tag{3}$$

which has χ and χ' as roots, satisfying $\chi \leq X$ and $\chi' \leq Y$, for some X and Y which will be larger that 2^{Υ}, say 2^{64}. We show that it is (heuristically) possible to recover in polynomial time any χ and χ' that are lower than $\sqrt{p}$.

Solving this bi-variate polynomial equation with small modular roots can be done using the well-known Coppersmith technique [11]. Finding small roots of modular bi-variate polynomials was studied in [17], but his method is very general, whereas we consider here only simple linear polynomials. The following

lemma, inspired by Howgrave-Graham's lemma [15] suggests how to construct a particular lattice that will help to recover small modular roots of a linear polynomial in $\mathbb{Z}[x, y]$. We denote as $\|\cdot\|$ the Euclidean norm of polynomials.

Lemma 1. *Let $g(x, y) \in \mathbb{Z}[x, y]$ be a linear polynomial that satisfies*

- $g(x_0, y_0) = 0 \mod p$ *for some* $|x_0| < X$ *and* $|y_0| < Y$,
- $\|g(xX, yY)\| < p/\sqrt{3}$.

Then $g(x_0, y_0) = 0$ holds over the integers.

Let us write a bi-variate linear polynomial as $P(x, y) = x + by + c$, with $b, c \in \mathbb{Z}_p$, which has a root (x_0, y_0) modulo p satisfying $|x_0| < X$ and $|y_0| < Y$. It suffices to divide by $b_{1,1}$ the polynomial from Eq. (3) to make it unary in the first variable. Lemma 1 suggests to find a small-norm polynomial $h(x, y)$ that shares its root with the initial polynomial $P(x, y)$. To do so, we construct the matrix whose rows are formed by the coefficients of the polynomials p, pyY and $P(xX, yY)$ in the basis $(1, X, Y)$. Using the LLL algorithm [19], we can find a small linear combination of these polynomials that will satisfy Lemma 1. Indeed, this matrix has determinant p^2XY and an LLL reduction of the basis of the lattice spanned by the rows of M will output one vector of norm upper bounded by $2^{3/4}(\det(M))^{1/3}$. We expect the second vector to behave as the first, which is confirmed experimentally.

To obtain two polynomials which satisfy Lemma 1, we need the inequality $2^{3/4}(\det(M))^{1/3} < p/\sqrt{3}$, i.e. $XY < 3^{-3/2} \cdot 2^{-9/4}p$. If $g(x, y) = g_0 + g_1x + g_2y$ and $h(x, y) = h_0 + h_1x + h_2y$ are the polynomials corresponding to the shortest vectors output by LLL, we can recover (x_0, y_0) as

$$x_0 = \frac{X(h_0g_2 - g_0h_2)}{g_1h_2 - h_1g_2} \text{ and } y_0 = \frac{Y(h_0g_1 - h_1g_0)}{g_2h_1 - h_2g_1}.$$

As a consequence, this method makes it possible to recover in polynomial time any values χ and χ' that mask the secret value $a_{1,1}$ if they are both below $\sqrt{p}$. The complexity of Nguyen and Stehlé's LLL is quadratic [21], in our case it is $O(d^5 \log(3/2 \log(p))^2)$, with $d = 3$. Then $a_{1,1}$ can be computed as $a_{1,1} = c_{1,1} + b_{1,1}\chi \mod p$. The scheme from [27] is therefore completely insecure.

Remark 2. *One could fix this issue in Wang* et al.*'s protocol by using a larger Υ (such that the value χ is actually uniformly distributed over $\mathbb{Z}_p$). This would make the protocol not more efficient for the delegator than the actual computation of a single exponentiation. However, even this inefficient protocol would not achieve the privacy security notion as explained in the paper's full version [9, §C].*

4 Generic Constructions for Privately Outsourcing Exponentiation

We focus on protocols for outsourcing a single exponentiation $(u, a) \mapsto u^a$. Protocols for outsourcing multi-exponentiations are given in the full version of the

paper [9]. As mentioned in the introduction, round complexity is the main bottleneck in improving the efficiency of secure protocols due to latency, and we consider only 1-round delegation protocols.

Protocols for fixed base exponentiation are probably folklore (e.g., see [18] for a verifiable variant of the protocol corresponding to the computation code $\boldsymbol{\beta} = 001f$) but remain unpublished (to the best of our knowledge). Protocols for variable base exponentiation seem to be new and are inspired by Gallant, Lambert and Vanstone's decomposition algorithm [13] (see below).

We recall that each case is referred to as its computation code $\boldsymbol{\beta}$ (see Sect. 2). All these protocols are secure in the (indistinguishability) privacy notion defined in [9], in the information-theoretic sense.

Theorem 1 (see [9]). *Let* GroupGen *be a group generator, let λ be a security parameter and let $\mathbb{G}$ be a group of prime order p output by* $\mathsf{GroupGen}(\lambda)$. *Let $(\mathcal{C}, \mathcal{S})$ be one client-server protocol for the delegated computation of the exponentiation u^a described in Protocols 1 – 8 (for the corresponding computation code $\beta \in \{0,1\}^4$ given in their description). The protocol $(\mathcal{C}, \mathcal{S})$ is unconditionally* $(\tau, 0)$-private against an honest-but-curious adversary *for any time τ.*

Tools. In our protocols, we use two classical algorithms. The first one (Algorithm 1) computes the multi-exponentiation $\prod_{i=1}^{t} g_i^{x_i}$, for $g_1, \ldots, g_t \in \mathbb{G}$ and $x_1, \ldots, x_t \in \mathbb{N}$ by using the simultaneous 2^w-ary method introduced by Straus [26]. The minimal cost (which depends on w) is $\ell(1 + o(1))$ multiplications overall, where ℓ denotes the maximal bit-length of the x_i's. The method looks at w bits of each of the exponents for each evaluation stage group multiplication (where w is a small positive integer), (see [2] for details).

Algorithm 1. Multi-Exponentiation by Simultaneous 2^w-ary method

Input: $g_1, \ldots, g_t \in \mathbb{G}$, $x_1, \ldots, x_t \in \mathbb{N}$ with $\ell = \max_{i \in \{1,\ldots,t\}} \lceil \log x_i \rceil$ and $x_j = \sum_{i=0}^{\lfloor \ell/w \rfloor - 1} e_{i,j} 2^{wi} \in \mathbb{N}$ and $e_{i,j} \in \{0, 2^w - 1\}$ for $i \in \{0, \ldots, \lfloor \ell/w \rfloor - 1\}$ and $j \in \{1, \ldots, t\}$
Output: $g_1^{x_1} \cdots g_t^{x_t} \in \mathbb{G}$
  for all non-zero t-tuples $E = (E_1, \ldots, E_t) \in \{0, \ldots, 2^w - 1\}^t$ **do**
    $g_E \leftarrow \prod_{1 \le i \le t} g_i^{E_i}$ ▷ Precomputation stage
  end for
  $h \leftarrow 1_{\mathbb{G}}$
  for i from $\lfloor \ell/w \rfloor - 1$ to 0 **do**
    $h \leftarrow h^{2^w}$
    $E \leftarrow (e_{i,1}, e_{i,2}, \ldots, e_{i,t})$
    $h \leftarrow h \cdot g_E$ ▷ Multiply h by table entry $g_E = \prod_{1 \le k \le t} g_i^{e_{i,k}}$
  end for
  return h

Let p be a prime number and $a \in \mathbb{Z}_p$. Let $s \ge 1$ be an integer and $\boldsymbol{\rho} = (\rho_1, \ldots, \rho_s) \in \mathbb{Z}_p^s$. An *$s$-dimensional decomposition of a with respect to $\boldsymbol{\rho}$* is an s-dimensional vector $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_s) \in \mathbb{Z}_p^s$ such that

$$\langle \boldsymbol{\alpha}, \boldsymbol{\rho} \rangle := \alpha_1 \rho_1 + \cdots + \alpha_s \rho_s = a \bmod p.$$

It is well-known that if the scalars ρ_i for $i \in \{1, \ldots, s\}$ have pairwise differences of absolute value at least $p^{1/s}$, then there exists a polynomial-time algorithm which on inputs a and $\boldsymbol{\rho}$ outputs an s-dimensional decomposition $\boldsymbol{\alpha} \in \mathbb{Z}_p^s$ of a with respect to $\boldsymbol{\rho}$ such that $0 \leq \alpha_i \leq C \cdot p^{1/s}$ for $i \in \{1, \ldots, s\}$ (for some small constant $C > 0$). To find this "small decomposition" of a, the algorithm applies a lattice reduction algorithm (such as the LLL-algorithm) to produce a short basis of the $\mathbb{Z}$-lattice of dimension $s+1$ spanned by the vectors $(p, 0, 0, \ldots, 0)$, $(\rho_1, 1, 0, \ldots, 0)$, $(\rho_2, 0, 1, \ldots, 0)$, $\ldots$, $(\rho_s, 0, 0, \ldots, 1)$ and applies Babai rounding algorithm [3] to find a nearby vector in this lattice from $(a, 0, \ldots, 0)$ (see [25] for details). In the following, we will refer to this second algorithm as the GLV Decomposition Algorithm (GLV-Dec for short) since the method was first introduced by Gallant, Lambert and Vanstone [13] to perform group exponentiations with endomorphism decomposition.

4.1 Constructions for Outsourcing Fixed Base Exponentiation

When the base u is fixed, one can assume that $\mathcal{C}$ can use a pseudo-random power generator for u. As described in Sect. 2, this generator $\mathcal{B}$ is invoked with no input and outputs a single (pseudo)-random pair $(u^k, k) \in \mathbb{G} \times \mathbb{Z}_p$ where k is uniformly distributed in $\mathbb{Z}_p$ (or statistically close to the uniform distribution). If the generator $\mathcal{B}(\cdot)$ is invoked several times, we assume that the output pairs are independent.

Trivial Cases. Obviously, the case $111f$ (everything public) is trivial (simply ask in clear to the delegatee $\mathcal{S}$ the computation of u^a as $\mathcal{S}(u, a)$) and the case $110f$ does not make sense (public inputs and private output), as well as the case $011f$ (secret base) in the prime order setting.

Cases where the Base is Secret ($0{*}{*}f$). If everything is secret (case $000f$), it is easy to delegate the computation of u^a for any exponent a using Protocol 1. The delegator computation amounts to two invocations of the generator $\mathcal{B}$ and one multiplication in $\mathbb{G}$, with only one exponentiation delegated to $\mathcal{S}$.

Even if the exponent is public (case $010f$), Protocol 1 remains the best possible in terms of multiplications in $\mathbb{G}$ (with only one invocation to $\mathcal{S}$) since there is only one multiplication and it is needed to hide the private result.

If the result is public (case $001f$), one can propose the improved Protocol 2, which needs only one invocation of the pseudo-random power generator and no multiplication in $\mathbb{G}$, with only one exponentiation delegated to $\mathcal{S}$.

Cases where the Base is Public ($1{*}{*}f$). If the result is public (case $101f$), Protocol 2 remains the best possible in terms of multiplications in $\mathbb{G}$ (with only one invocation to $\mathcal{S}$) since no multiplication is needed.

If the result is secret (case $100f$), Protocol 3 is the best possible in terms of multiplications in $\mathbb{G}$ since it only needs one invocation of the pseudo-random power generator and one multiplication in $\mathbb{G}$ (needed to hide the private result of the exponentiation), with only one exponentiation delegated to $\mathcal{S}$.

4.2 Constructions for Outsourcing Variable Base Exponentiation

We consider the case when $\mathcal{C}$ wants to delegate the computation of u^a but with a variable u. One cannot assume that $\mathcal{C}$ can use a pseudo-random power generator for u but we can still suppose that it can use a pseudo-random power generator for a fixed generator g that we still call $\mathcal{B}$ with the same properties as before.

Trivial Cases. As above, the case $111v$ (everything public) is trivial (simply ask in clear to the delegatee $\mathcal{S}$ the computation of u^a as $\mathcal{S}(u, a)$) and the case $110v$ does not make sense (public inputs and private output), as well as the case $011v$ (secret base) in the prime order setting.

Cases where the Base is Public ($1**v$). We first consider the case where the variable base u can be made public but not the exponent nor the result (case $100v$). We propose a family of protocols depending on a parameter s that perform the computation of u^a by delegating s exponentiations to a delegator and $\log(p)/(s+1)$ operations in $\mathbb{G}$.

This family of protocols is given in Protocol 5 and the specific case $s = 1$ is Protocol 4. Note that these protocols do not make use of the pseudo-random power generator for g. Unfortunately, the efficiency gain is only a factor s and if the number of delegated exponentiations is constant the delegator still has to perform $O(\log p)$ operations in $\mathbb{G}$.

These protocols are actually optimal in terms of operations in $\mathbb{G}$, as shown in Theorem 2. Obviously, we can also use these protocols if we allow the result u^a to be public (case $101v$) and the optimal result of Theorem 2 show that even in this easier setting, the protocol cannot be improved.

Cases where the Base is Private ($0**v$). We can use this protocol family to construct another delegation protocol for the corresponding cases where the base is kept secret ($000v$ and $001v$). We obtain Protocol 6 that makes two invocations of the pseudo-random generator for g and requires the delegation of one further exponentiation compared to Protocol 5 (and Protocol 4). We do not actually know if these protocols are optimal but the gap is rather tight (see Table 2).

Constructing an outsourcing protocol in these cases with only one exponentiation delegation (or proving it is impossible) is left as an open problem.

We can also use this protocol if we allow the exponent a to be public ($010v$). However, in this case one can improve it with Protocol 7 where the delegator performs only a constant number of group operations in $\mathbb{G}$. In this case, one can also improve it with Protocol 8 where the delegator makes only one call to the delegatee, but at the price of a $O(\log(p))$ number of group operations in $\mathbb{G}$.

Remark 3. *In [7], Cavallo* et al. *presented two other protocols for outsourcing private variable base and public exponent exponentiation. The first one [7, §4, p. 164], recalled in Protocol 9, achieves only the basic security requirement (i.e., in the sense of one-wayness instead of indistinguishability). It relies on a subset-sum in a group and in order to achieve a stronger privacy notion, the delegation scheme actually becomes less efficient for the delegator than performing*

the exponentiation on its own. The second scheme is much more efficient since the delegator computation is constant but it requires a stronger pseudo-random power generator $\mathcal{B}$ that outputs pseudo-random triples of the form (g^r, g^{ar}, r). In particular, this second protocol can only be used for fixed values of the public exponent a.

Protocol 1: $000f$ (and $010f$)

Input: $u \in \mathbb{G}$, $a \in \mathbb{Z}_p$
Output: $u^a \in \mathbb{G}$
  $(u^r, r) \leftarrow \mathcal{B}(\cdot)$ $(u^s, s) \leftarrow \mathcal{B}(\cdot)$
  $t \leftarrow (a - s)/r \bmod p$
  $h \leftarrow \mathcal{S}(u^r, t \bmod p)$
  return $h \cdot u^s$

Protocol 2: $001f$ (and $101f$)

Input: $u \in \mathbb{G}$, $a \in \mathbb{Z}_p$
Output: $u^a \in \mathbb{G}$
  $(u^k, k) \leftarrow \mathcal{B}(\cdot)$
  $h \leftarrow \mathcal{S}(u^k, a/k \bmod p)$
  return h

Protocol 3: $100f$

Input: $u \in \mathbb{G}$, $a \in \mathbb{Z}_p$
Output: $u^a \in \mathbb{G}$
  $(u^k, k) \leftarrow \mathcal{B}(\cdot)$
  $h \leftarrow \mathcal{S}(u, a - k \bmod p)$
  return $h \cdot g^k$

Protocol 4: $100v$ (and $101v$)

Input: $u \in \mathbb{G}$, $a \in \mathbb{Z}_p$
Output: $u^a \in \mathbb{G}$
  $T \leftarrow \lceil \sqrt{p} \rceil$
  $h \leftarrow \mathcal{S}(u, T)$
  $a_0 = a \bmod T$
  $a_1 = a \text{ div } T$ ▷ Euclidean division: $a = a_1 \cdot T + a_0$
  return $u^{a_0} h^{a_1}$ ▷ using Alg. 1

Protocol 5: $100v$ (and $101v$)

Input: $u \in \mathbb{G}$, $a \in \mathbb{Z}_p$
Output: $u^a \in \mathbb{G}$
  $T \leftarrow \lceil p^{1/s+1} \rceil$
  for i from 1 to s **do**
    $h_i \leftarrow \mathcal{S}(u, T^i)$
  end for
  $\mathsf{temp} \leftarrow a$
  for i from s down to 0 **do**
    $a_i = \mathsf{temp} \text{ div } T^i$
    $\mathsf{temp} = \mathsf{temp} - a_i \cdot T^i$
  end for
  ▷ $a = a_s \cdot T^s + \cdots + a_1 T + a_0$
  return $u^{a_0} \prod_{i=1}^{s} h_i^{a_i}$ ▷ using Alg. 1

Protocol 6: $000v$ (and $001v$)

Input: $u \in \mathbb{G}$, $a \in \mathbb{Z}_p$
Output: $u^a \in \mathbb{G}$
  $(g^{k_1}, k_1) \leftarrow \mathcal{B}(\cdot)$; $(g^{k_2}, k_2) \leftarrow \mathcal{B}(\cdot)$
  $v \leftarrow u \cdot g^{k_1}$
  $h_1 \leftarrow v^a$ ▷ delegated with Prot. 5
  $h_2 \leftarrow \mathcal{S}(g, -ak_1 - k_2 \bmod p)$
  return $h_1 \cdot h_2 \cdot g^{k_2}$

Protocol 7: $010v$

Input: $u \in \mathbb{G}$, $a \in \mathbb{Z}_p$
Output: $u^a \in \mathbb{G}$
  $(g^r, r) \leftarrow \mathcal{B}(\cdot)$; $(g^s, s) \leftarrow \mathcal{B}(\cdot)$
  $(g^t, t) \leftarrow \mathcal{B}(\cdot)$
  $k \leftarrow (t - ra)/s \bmod p$
  $h_1 \leftarrow \mathcal{S}(u \cdot g^r, a)$; $h_2 \leftarrow \mathcal{S}(g^s, k)$
  return $h_1 h_2 g^t$

Protocol 8: $010v$

Input: $u \in \mathbb{G}$, $a \in \mathbb{Z}_p$
Output: $u^a \in \mathbb{G}$
  $(g^r, r) \leftarrow \mathcal{B}(\cdot)$
  for i from 1 to s **do**
    $(g^{t_i}, t_i) \leftarrow \mathcal{B}(\cdot)$
  end for
  $(k_0, k_1, \ldots, k_s) \leftarrow$
  GLV-Dec$(1, t_1, \ldots, t_s, -ra \bmod p)$
    ▷ with $k_i \leq p^{1/(s+1)}$
  $h_1 \leftarrow \mathcal{S}(u \cdot g^r, a)$
  $h_2 \leftarrow g^{k_0} (g^{t_1})^{k_1} \ldots (g^{t_s})^{k_s}$ ▷ Alg.1
  return $h_1 h_2$

Protocol 9: $010v$ from [7]

Input: $u \in \mathbb{G}$, $a \in \mathbb{Z}_p$
Output: $u^a \in \mathbb{G}$
  for i from 1 to s **do**
    $g_i \xleftarrow{R} \mathbb{G}$
  end for
  $\mathcal{I} \xleftarrow{R} \mathfrak{P}_m(\{1, \ldots, s\})$ ▷ random subset of cardinal m of $\{1, \ldots, s\}$
  $g_{s+1} \leftarrow u \cdot \prod_{i \in \mathcal{I}} g_i$
  for i from 1 to s **do**
    $h_i \leftarrow \mathcal{S}(g_i, -a)$
  end for
  $h_{s+1} \leftarrow \mathcal{S}(g_{s+1}, a)$
  return $h_{s+1} \cdot \prod_{i \in \mathcal{I}} h_i$

5 Complexity Lower Bound for One-Round Protocols

We focus on studying protocols with minimal interaction, namely the delegator is allowed to delegate the computation of several group exponentiations but it must send all of them to the delegatee in only one communication round. Indeed, interactions over computer networks are usually the most time consuming operations (due to lagging or network congestion) and it is very important to study protocols which require the minimal number of rounds to complete. In the full version [9], we also present complexity lower bounds for multi-round protocols.

By "lower bounds", we mean that the number of calls to the delegatee oracle $\mathcal{S}$ and to the pseudo-random power generator $\mathcal{B}$ are fixed, and that we consider the number of group operations. Concerning the first part of Table 2, the bounds come from the protocols given in Sect. 4, since at least one call to the group oracle is mandatory when the result is private (the delegator $\mathcal{C}$ needs to do at least one computation after having received a public result from the delegatee oracle $\mathcal{S}$). The cases $101v$ and $100v$ are then dealt with in Theorem 2. For all these cases, the protocols proposed in Sect. 4 are thus actually optimal. As for Case $010v$, the lower bound for a unique call to $\mathcal{S}$ is proven in Theorem 3, whereas Protocol 7 gives a (constant) upper bound in case we allow a second call to $\mathcal{S}$. Finally, the lower bounds for Cases $001v$ and $000v$ come from the equivalent bounds for Cases $101v$ and $100v$, since the variable base is furthermore assumed to be secret.

In what follows, and as mentioned above, we use the generic group model to prove these lower bounds. We model the different operations as follows:

- The group oracle $\mathcal{G}$ takes as inputs two encodings $\sigma_1 = \sigma(h_1)$ and $\sigma_2 = \sigma(h_2)$ and outputs the encoding σ_3 such $\sigma_3 = \sigma(h_1 h_2)$ (see [24]).
- The pseudo-random power generator $\mathcal{B}$ outputs pairs $(t, \sigma(g^t))$ where the scalar t is picked uniformly at random in $\mathbb{Z}_p$ (independently for all queries).
- The delegatee oracle $\mathcal{S}$ takes as inputs an encoding $\sigma_0 = \sigma(h)$ and a scalar x and outputs the encoding $\sigma'_0 = \sigma(h^x)$ (*i.e.* $\sigma^{-1}(\sigma'_0) = \sigma^{-1}(\sigma_0)^x$).

In order to prove our complexity lower bounds, we make use of the following simple lemma (whose proof is provided in the paper full version [9]):

Lemma 2. *Let* GroupGen *be a group generator, let* $\mathbb{G}$ *be a group of prime order* p *output by* GroupGen *and let* $\mathcal{A}$ *be a generic algorithm in* $\mathbb{G}$*. If* $\mathcal{A}$ *is given as inputs encodings* $\sigma(g_1), \ldots, \sigma(g_n)$ *of groups elements* $g_1, \ldots, g_n \in \mathbb{G}$ *(for* $n \in \mathbb{N}$*) and outputs the encoding* $\sigma(h)$ *of a group element* $h \in \mathbb{G}$ *in time* τ*, then there exists positive integers* $\alpha_1, \ldots, \alpha_n$ *such that* $h = g_1^{\alpha_1} \ldots g_n^{\alpha_n}$ *and* $\max(\alpha_1, \ldots, \alpha_n) \leq 2^\tau$*.*

The following theorems assert that for the cases $101v$ and $100v$, the protocols proposed in Sect. 4 are actually optimal in terms of calls to $\mathcal{S}$ and $\mathcal{G}$.

Theorem 2. *Let* GroupGen *be a group generator and let* $(\mathcal{C}, \mathcal{S})$ *be one client-server protocol for the delegated computation of the exponentiation* u^a *for the corresponding computation code* $\boldsymbol{\beta} = 101v$*. We assume that the delegator* $\mathcal{C}$ *is a generic group algorithm that uses*

– $c\log(p) + O(1)$ *generic group operations (for all groups* $\mathbb{G}$ *of primer order* p *output by* $\mathsf{GroupGen}(\lambda)$*) for some constant* c*,*
– $\ell = O(1)$ *queries to the (private) pseudo-random power generator* $\mathcal{B}$
– *and only* 1 *delegated exponentiation to the delegatee* $\mathcal{S}$

If $c < 1/2$*, then* $(\mathcal{C}, \mathcal{S})$ *is not private: there exists an algorithm running in polynomial-time such that* $\Pr[bit \leftarrow \mathbf{Exp}_{priv}(\mathcal{A}) : bit = 1] \geq 1 - \lambda^{O(1)}$.

Proof. For the ease of exposition, we present a proof for the simple case $s = 1$ where the delegator $\mathcal{C}$ outsources only one exponentiation to the delegatee $\mathcal{S}$. The complete proof is given in the full version of the paper [9]. We assume that $\mathcal{C}$ gets as input two encodings $\sigma(u)$, $\sigma(g)$ of *two* group elements u and g and one scalar a in $\mathbb{Z}_p$ and outputs the encoding $\sigma(u^a)$ of the group element u^a by making q queries to the group oracle $\mathcal{G}$, ℓ queries to the (private) pseudo-random power generator $\mathcal{B}$ and 1 query to $\mathcal{S}$.

We assume that $q = c\log p + O(1)$ with $c < 1/2$ and we prove that it is not possible for $\mathcal{C}$ to compute $\sigma(u^a)$ in such a way that the delegatee $\mathcal{S}$ learns no information on a. More precisely, we construct a polynomial-time adversary $\mathcal{A}$ for the privacy security notion. The adversary chooses a group element u and two scalars $(a_0, a_1) \in \mathbb{Z}_p^2$. For the sake of simplicity, we assume that the adversary picks $(a_0, a_1) \in \mathbb{Z}_p^2$ uniformly at random among the scalars of bit-length $\log(p)$ and u uniformly at random in $\mathbb{G}$. The challenger picks uniformly at random a bit $b \in \{0, 1\}$ and sets $a = a_b$. The delegator runs the delegation protocol with inputs u and a and delegates one exponentiation to the adversary acting as the delegatee. The adversary has to guess the bit b.

Let us denote $(t_1, \sigma(g^{t_1}))$, $(t_2, \sigma(g^{t_2}))$, $\ldots$, $(t_\ell, \sigma(g^{t_\ell}))$ the pairs obtained from the pseudo-random power generator $\mathcal{B}$ by the delegator $\mathcal{C}$. Since $\mathcal{B}$ takes no inputs and outputs independent pairs, we can assume without loss of generality that the delegator $\mathcal{C}$ makes the ℓ queries to $\mathcal{B}$ in a first phase of the delegation protocol. We denote $(\sigma(h), x)$ the unique pair encoding of group element/scalar made by $\mathcal{C}$ to the delegatee $\mathcal{S}$ (which is executed by the adversary $\mathcal{A}$ in an "honest-but-curious" way). Using generic group operations, $\mathcal{C}$ can only construct the corresponding group elements such that:

$$h = u^{\alpha'} \cdot g^{\kappa'} \cdot g^{t_1\gamma'_1} \cdots g^{t_\ell\gamma'_\ell} \tag{4}$$

for some scalars $(\alpha', \kappa', \gamma'_1, \ldots, \gamma'_\ell)$. We denote $k = h^x$ the response of $\mathcal{S}$. Eventually, the delegator $\mathcal{C}$ outputs the encoding $\sigma(u^a)$ of the group element u^a. Again, using generic group operations, it can only construct it as

$$u^a = u^\alpha g^\kappa \cdot g^{t_1\gamma_1} \cdots g^{t_\ell\gamma_\ell} k^\delta h^\varepsilon \tag{5}$$

for some scalars $(\alpha, \kappa, \gamma_1, \ldots, \gamma_\ell, \delta, \varepsilon)$. If we assume that $q = c\log n + O(1)$ (and in particular $q = o(\sqrt{p})$), the delegator $\mathcal{C}$ is not able to compute the discrete logarithm of u in base g. This means that necessarily the exponent of g in Eq. (5) cancels out. Recall that $k = h^x$, h being constructed as in Eq. (4). Thus, taking only the discrete logarithms of powers of u in base u of this equation, we obtain

$$a = \alpha + \varepsilon\alpha' + \delta\alpha' x \mod p \tag{6}$$

We denote τ_1 the number of group operations performed by $\mathcal{C}$ in the computation of h described in Eq. (4) and τ_2 the number of operations in the computation of u^a described in Eq. (5). By assumption, $\tau_1 + \tau_2 \leq c \log p + O(1)$. Furthermore, since $\mathcal{C}$ only used generic group operations, we have (by Lemma 2) $\alpha' \leq 2^{\tau_1}$, $\alpha \leq 2^{\tau_2}$, $\delta \leq 2^{\tau_2}$ and $\varepsilon \leq 2^{\tau_2}$. If we note $\rho_1 = \alpha + \varepsilon\alpha'$ and $\rho_2 = \delta\alpha'$, Eq. (6) becomes $a = \rho_1 + x\rho_2 \mod p$, where x is known to the adversary, $\rho_2 = \delta\alpha' \leq 2^{\tau_1} 2^{\tau_2} = 2^{\tau_1+\tau_2} \leq p^{c+o(1)}$ and $\rho_1 = \alpha + \varepsilon\alpha' \leq 2^{\tau_1} + 2^{\tau_1} 2^{\tau_2} \leq p^{c+o(1)}$.

The adversary $\mathcal{A}$ can then try to decompose a_0 and a_1 as $a_i = \rho_{i,1} + x\rho_{i,2} \mod p$, with $\rho_{i,1}, \rho_{i,2} \leq p^{c+o(1)}$. For $a_b = a$, the decomposition algorithm provided in the paper full version [9] (which generalizes the main attack on Wang *et al.*'s protocol) will recover $\rho_{b,1}$ and $\rho_{b,2}$ in polynomial time. However, for a given x and a random a_{1-b} of bit-length $\log(p)$, there is only a negligible probability that such a decomposition exists (less than $p^{c+o(1)} \times p^{c+o(1)} = p^{2c+o(1)} = o(p)$ scalars can be written in this way). Thus, the adversary can simply run the decomposition algorithm mentioned above on (a_0, x) on one hand and on (a_1, x) on the other hand and returns the bit b for which the algorithm returns a "small decomposition" on input (a_b, x). By the previous analysis, its advantage is noticeable.

Remark 4. *It is worth mentioning that even in (generic) groups where division is significantly less expensive than multiplication (such as elliptic curves or class groups of imaginary quadratic number fields), this lower bound (as well as the following ones) still holds (see the paper full version [9] for details).*

Protocol 7 shows that it is possible to delegate a secret base, public exponent exponentiation with only a constant number of operations if the delegator can delegate at least two exponentiations. Theorem 3 asserts that if the delegator is only allowed to delegate one exponentiation then Protocol 8 is almost optimal in this setting. More precisely, we show that the delegator has to perform at least $O(\log(p))$ group operations if it delegates only one exponentiation and makes at most a constant number of queries to the pseudo-random power generator $\mathcal{B}$. Due to lack of space, the proof is provided in the full version of the paper [9].

Theorem 3. *Let* GroupGen *be a group generator and let* $(\mathcal{C}, \mathcal{S})$ *be one client-server protocol for the delegated computation of one exponentiation for the computation code* $\boldsymbol{\beta} = 010v$*. We assume that the delegator* $\mathcal{C}$ *is a generic group algorithm that uses*

- $c \log(p) + O(1)$ *generic group operations (for groups* $\mathbb{G}$ *of order* p *output by* $\mathsf{GroupGen}(\lambda)$*),*
- $\ell = O(1)$ *queries to the (private) pseudo-random power generator* $\mathcal{B}$
- *and only* 1 *delegated exponentiation to the delegatee* $\mathcal{S}$

If the constant c *satisfies* $c < 1/(\ell+2)$*, then* $(\mathcal{C}, \mathcal{S})$ *is not private: there exists an algorithm running in time* $O(p^{c/2+o(1)})$ *s.t.* $\Pr[bit \leftarrow \mathbf{Exp}_{priv}(\mathcal{A}) : bit = 1] = 1$.

Table 2. Outsourcing protocols for single exponentiation

Constructions for Outsourcing Fixed Base Exponentiation (with a pseudo-random power generator of pairs (k, u^k) available)									
Code	u	a	u^a	Secure protocol*	Complexity	Complexity Lower Bound (for $\mathcal{G}$)			
						Resources	Lower Bound	Proof	Optimality
111f	Public	Public	Public	Trivial					
110f	Public	Public	Secret	Non-sense					
101f	Public	Secret	Public	Protocol 2	$1\ \mathcal{S} + 1\ \mathcal{B}$	$(1\ \mathcal{S}, \ell\ \mathcal{B})$	$0\ \mathcal{G}$	From Protocol 2	✓
100f	Public	Secret	Secret	Protocol 3	$1\ \mathcal{S} + 1\ \mathcal{G} + 1\ \mathcal{B}$	$(1\ \mathcal{S}, \ell\ \mathcal{B})$	$1\ \mathcal{G}$	From Protocol 3	✓
011f	Secret	Public	Public	Non-sense†					
010f	Secret	Public	Secret	Protocol 1	$1\ \mathcal{S} + 1\ \mathcal{G} + 2\ \mathcal{B}$	$(1\ \mathcal{S}, \ell\ \mathcal{B})$	$1\ \mathcal{G}$	From Protocol 1	✓
001f	Secret	Secret	Public	Protocol 2	$1\ \mathcal{S} + 1\ \mathcal{B}$	$(1\ \mathcal{S}, \ell\ \mathcal{B})$	$0\ \mathcal{G}$	From Protocol 2	✓
000f	Secret	Secret	Secret	Protocol 1	$1\ \mathcal{S} + 1\ \mathcal{G} + 2\ \mathcal{B}$	$(1\ \mathcal{S}, \ell\ \mathcal{B})$	$1\ \mathcal{G}$	From Case 010f	✓
Constructions for Outsourcing Variable Base Exponentiation (with a pseudo-random power generator of pairs (k, g^k) available)									
Code	u	a	u^a	Secure protocol *	Complexity	Complexity Lower Bound (for $\mathcal{G}$)			
						Resources	Lower Bound	Proof	Optimality
111v	Public	Public	Public	Trivial					
110v	Public	Public	Secret	Non-sense					
101v	Public	Secret	Public	Protocol 4	$1\ \mathcal{S} + L_p/2\ \mathcal{G}$	$(1\ \mathcal{S}, \ell\ \mathcal{B})$	$L_p/2\ \mathcal{G}$	Theorem 2, Section 5	✓
				Protocol 5	$s\ \mathcal{S} + L_p/(s+1)\ \mathcal{G}$	$(s\ \mathcal{S}, \ell\ \mathcal{B})$	$L_p/(s+1)\ \mathcal{G}$	Theorem 2, Section 5	✓
100v	Public	Secret	Secret	Protocol 4	$1\ \mathcal{S} + L_p/2\ \mathcal{G}$	$(1\ \mathcal{S}, \ell\ \mathcal{B})$	$L_p/2\ \mathcal{G}$	Theorem 2, Section 5	✓
				Protocol 5	$s\ \mathcal{S} + L_p/(s+1)\ \mathcal{G}$	$(s\ \mathcal{S}, \ell\ \mathcal{B})$	$L_p/(s+1)\ \mathcal{G}$	Theorem 2, Section 5	✓
011v	Secret	Public	Public	Non-sense†					
010v	Secret	Public	Secret	Protocol 8	$1\ \mathcal{S} + L_p/\ell\ \mathcal{G} + \ell\ \mathcal{B}$	$(1\ \mathcal{S}, \ell\ \mathcal{B})$	$L_p/(\ell+3)\ \mathcal{G}$	Theorem 3, Section 5	✗
				Protocol 7	$2\ \mathcal{S} + 3\ \mathcal{G} + 3\ \mathcal{B}$	$(2\ \mathcal{S}, \ell\ \mathcal{B})$	$t^{\ddagger}\ \mathcal{G}$	From Protocol 7	✗
001v	Secret	Secret	Public	/		$(1\ \mathcal{S}, \ell\ \mathcal{B})$	$L_p/2\ \mathcal{G}$	From Case 101v	/
				6 (using 4)	$2\ \mathcal{S} + L_p/2\ \mathcal{G} + 2\ \mathcal{B}$	$(2\ \mathcal{S}, \ell\ \mathcal{B})$	$L_p/3\ \mathcal{G}$	From Case 101v	✗
				6 (using 5)	$s\ \mathcal{S} + L_p/s\ \mathcal{G} + 2\ \mathcal{B}$	$(s\ \mathcal{S}, \ell\ \mathcal{B})$	$L_p/(s+1)\ \mathcal{G}$	From Case 101v	✗
000v	Secret	Secret	Secret	/		$(1\ \mathcal{S}, \ell\ \mathcal{B})$	$L_p/2\ \mathcal{G}$	From Case 100v	/
				6 (using 4)	$2\ \mathcal{S} + L_p/2\ \mathcal{G} + 2\ \mathcal{B}$	$(2\ \mathcal{S}, \ell\ \mathcal{B})$	$L_p/3\ \mathcal{G}$	From Case 100v	✗
				6 (using 5)	$s\ \mathcal{S} + L_p/s\ \mathcal{G} + 2\ \mathcal{B}$	$(s\ \mathcal{S}, \ell\ \mathcal{B})$	$L_p/(s+1)\ \mathcal{G}$	From Case 100v	✗

Notations: $\ell = O(1)$ and $L_p = \log(p)$. * refers to the protocols page 12 † Prime order setting. ‡ With $t \in \{0, 1, 2, 3\}$.

6 Conclusion and Future Work

All our results on (one-round) secure delegation of group exponentation are collected in Table 2. In addition, we also provide protocols and lower-bounds for multi-exponentiations and lower bounds for multi-round delegation of exponentiation protocols in the paper full version [9]. As a future work, understanding the relationship between computational efficiency and memory usage is vital when implementing delegation protocols. In particular, it is interesting to propose efficient delegation protocols and to improve our lower bounds in settings where the memory complexity of the delegator is limited.

Acknowledgments. The authors are supported in part by the French ANR JCJC ROMAnTIC project (ANR-12-JS02-0004), the French ANR EnBid Project (ANR-14-CE28-0003) and by ERC Starting Grant ERC-2013-StG-335086-LATTAC. The authors thank Guillaume Hanrot and Damien Stehlé for helpful discussions, and Olivier Billet for his comments and for pointing out references.

References

1. Ateniese, G., Burns, R.C., Curtmola, R., Herring, J., Kissner, L., Peterson, Z.N.J., Song, D.X.: Provable data possession at untrusted stores. In: Ning et al. [22], pp. 598–609. http://doi.acm.org/10.1145/1315245.1315318
2. Avanzi, R.M.: The complexity of certain multi-exponentiation techniques in cryptography. J. Cryptology **18**(4), 357–373 (2005)
3. Babai, L.: On Lovász' lattice reduction and the nearest lattice point problem. Combinatorica **6**(1), 1–13 (1986)
4. Boneh, D., Lynn, B., Shacham, H.: Short signatures from the Weil pairing. J. Cryptology **17**(4), 297–319 (2004)
5. Boyko, V., Peinado, M., Venkatesan, R.: Speeding up discrete log and factoring based schemes via precomputations. In: Nyberg, K. (ed.) EUROCRYPT 1998. LNCS, vol. 1403, pp. 221–235. Springer, Heidelberg (1998)
6. Canard, S., Devigne, J., Sanders, O.: Delegating a pairing can be both secure and efficient. In: Boureanu, I., Owesarski, P., Vaudenay, S. (eds.) ACNS 2014. LNCS, vol. 8479, pp. 549–565. Springer, Heidelberg (2014)
7. Cavallo, B., Crescenzo, G.D., Kahrobaei, D., Shpilrain, V.: Efficient and secure delegation of group exponentiation to a single server. In: Mangard, S., Schaumont, P. (eds.) Radio Frequency Identification. LNCS, vol. 9440, pp. 156–173. Springer, Heidelberg (2015)
8. Chen, X., Li, J., Ma, J., Tang, Q., Lou, W.: New algorithms for secure outsourcing of modular exponentiations. In: Foresti, S., Yung, M., Martinelli, F. (eds.) ESORICS 2012. LNCS, vol. 7459, pp. 541–556. Springer, Heidelberg (2012)
9. Chevalier, C., Laguillaumie, F., Vergnaud, D.: Privately outsourcing exponentiation to a single server: Cryptanalysis and optimal constructions. Cryptology ePrint Archive, Report 2016/309 (2016). http://eprint.iacr.org/
10. Chevallier-Mames, B., Coron, J.-S., McCullagh, N., Naccache, D., Scott, M.: Secure delegation of elliptic-curve pairing. In: Gollmann, D., Lanet, J.-L., Iguchi-Cartigny, J. (eds.) CARDIS 2010. LNCS, vol. 6035, pp. 24–35. Springer, Heidelberg (2010)

11. Coppersmith, D.: Finding a small root of a univariate modular equation. In: Maurer, U.M. (ed.) EUROCRYPT 1996. LNCS, vol. 1070, pp. 155–165. Springer, Heidelberg (1996)
12. Dent, A.W.: Adapting the weaknesses of the random oracle model to the generic group model. In: Zheng, Y. (ed.) ASIACRYPT 2002. LNCS, vol. 2501, pp. 100–109. Springer, Heidelberg (2002)
13. Gallant, R.P., Lambert, R.J., Vanstone, S.A.: Faster point multiplication on elliptic curves with efficient endomorphisms. In: Kilian, J. (ed.) CRYPTO 2001. LNCS, vol. 2139, pp. 190–200. Springer, Heidelberg (2001)
14. Hohenberger, S., Lysyanskaya, A.: How to securely outsource cryptographic computations. In: Kilian, J. (ed.) TCC 2005. LNCS, vol. 3378, pp. 264–282. Springer, Heidelberg (2005)
15. Howgrave-Graham, N.: Finding small roots of univariate modular equations revisited. In: Darnell, M. (ed.) Crytography and Coding. LNCS, vol. 1355, pp. 131–142. Springer, Heidelberg (1997)
16. Juels, A., Kaliski Jr., B.S.: PORS: proofs of retrievability for large files. In: Ning et al. [22], pp. 584–597. http://doi.acm.org/10.1145/1315245.1315317
17. Jutla, C.S.: On finding small solutions of modular multivariate polynomial equations. In: Nyberg, K. (ed.) EUROCRYPT 1998. LNCS, vol. 1403, pp. 158–170. Springer, Heidelberg (1998)
18. Kiraz, M.S., Uzunkol, O.: Efficient and verifiable algorithms for secure outsourcing of cryptographic computations. Int. J. Inf. Secur, 1–19 (2015, to appear). doi:10.1007/s10207-015-0308-7
19. Lenstra, A.K., Lenstra, H.W.J., Lovász, L.: Factoring polynomials with rational coefficients. Math. Ann. **261**, 515–534 (1982)
20. Nguyen, P., Shparlinski, I., Stern, J.: Distribution of modular sums and the security of server aided exponentiation. Workshop Comp. Number Theor. Crypt. **20**, 1–16 (1999)
21. Nguyen, P.Q., Stehlé, D.: Floating-point LLL revisited. In: Cramer, R. (ed.) EUROCRYPT 2005. LNCS, vol. 3494, pp. 215–233. Springer, Heidelberg (2005)
22. Ning, P., di Vimercati, S.D.C., Syverson, P.F. (eds.): Proceedings of the 2007 ACM Conference on Computer and Communications Security, CCS 2007, Alexandria, Virginia, USA, October 28–31, 2007. ACM (2007)
23. Shacham, H., Waters, B.: Compact proofs of retrievability. In: Pieprzyk, J. (ed.) ASIACRYPT 2008. LNCS, vol. 5350, pp. 90–107. Springer, Heidelberg (2008)
24. Shoup, V.: Lower bounds for discrete logarithms and related problems. In: Fumy, W. (ed.) EUROCRYPT 1997. LNCS, vol. 1233, pp. 256–266. Springer, Heidelberg (1997)
25. Smith, B.: Easy scalar decompositions for efficient scalar multiplication on elliptic curves and genus 2 Jacobians. Contemp. Math. Ser. **637**, 15 (2015)
26. Straus, E.G.: Problems and solutions: Addition chains of vectors. Am. Math. Mon. **71**, 806–808 (1964)
27. Wang, Y., Wu, Q., Wong, D.S., Qin, B., Chow, S.S.M., Liu, Z., Tan, X.: Securely outsourcing exponentiations with single untrusted program for cloud storage. In: Kutyłowski, M., Vaidya, J. (eds.) ESORICS 2014, Part I. LNCS, vol. 8712, pp. 326–343. Springer, Heidelberg (2014)