An Efficient Small Modulus Test and Its Applications to Delegated Computation Without Preprocessing

Matluba Khodjaeva[1(✉)] and Giovanni di Crescenzo[2]

[1] CUNY John Jay College of Criminal Justice, New York, NY, USA
mkhodjaeva@jjay.cuny.edu
[2] Peraton Labs Inc., Basking Ridge, NJ, USA
gdicrescenzo@peratonlabs.com

Abstract. Delegation of operations used in cryptographic schemes from a computationally weaker client to a computationally stronger server has been advocated to expand the applicability of cryptosystems to computing with resource-constrained devices. Classical results for the verification of integer and polynomial products are based on a test due to Pippenger, Yao and Kaminski which verifies these operations modulo a small prime. In this paper we describe and prove an efficient small integer modulus test and show its application to single-server delegated computation of operations of interest in cryptosystems. In particular, we show single-server delegated computation protocols, *without any preprocessing*, for the following operations:

1. modular multiplication of two public group values,
2. modular inverse of a public group value,
3. modular inverse of a private group value, and
4. exponentiation of a public base to a small public exponent in the RSA group.

Our protocols satisfy result correctness, input privacy (unless the input is public), result security and client efficiency. Previous work satisfied only a subset of these properties, or required preprocessing, or satisfied lower client efficiency.

Keywords: Small Modulus Test · Applied Cryptography · Secure Delegation · Group Theory

1 Introduction

Server-aided cryptography (starting with, e.g., [1,11,22]) addresses the problem of resource-constrained clients, such as IoT devices, delegating or outsourcing cryptographic computations to computationally more powerful servers. Currently, this area is seeing a renewed interest because of the increasing popularity of various computing trends (i.e., computing over IoT devices' data, cloud/edge/fog computing, etc.), and the need to efficiently implement cryptographic schemes and their sometimes relatively expensive operations on them.

© The Author(s), under exclusive license to Springer Nature Switzerland AG 2024
M. Manulis et al. (Eds.): SecITC 2023, LNCS 14534, pp. 157–177, 2024.
https://doi.org/10.1007/978-3-031-52947-4_12

The ubiquitous deployment of resource-constraint devices makes the security designer's life harder, in that the task of guaranteeing the security of these devices becomes less and less manageable. The natural approach of running a preprocessing phase where cryptographic keys and credentials are stored on these devices and then allowing them to participate in state-of-the-art cryptography protocols based on this stored information, may not always succeed, since sometimes these devices are deployed in use cases where physical security (specifically, confidentiality and/or integrity) of any stored secret keys or data cannot be guaranteed.

This motivated the problem studied in this paper: is it possible for a resource-constrained client to efficiently, privately and securely delegate to a server the computation of operations used in currently applied cryptography schemes, *without need for a preprocessing phase*? A solution to this problem needs to make computation for the client more efficient than in a non-delegated computation, but also needs to withstand server's attacks in learning any new information about the input to the computation (when input privacy is desired), or in disrupting the computation and fooling the client into accepting an incorrect computation result. All of the above needs to be achieved *without* a preprocessing phase storing data on the client's memory.

More generally, we require a solution to the delegation of a function F to be a 2-party protocol between client C and server S, where C and S have a brief message exchange (typically, a message from C to S followed by one from S to C; see Fig. 1), and where the following requirements are satisfied (see also Appendix A for more formal definitions):

1. δ_c*-result correctness*: if C and S honestly run the protocol, at the end of the protocol C returns $F(x)$ with some high probability δ_c;
2. ϵ_p*-input privacy*: except for some small probability ϵ_p, no new information about input x is revealed to S;
3. ϵ_s*-result security*: S should not be able, except possibly with some small probability ϵ_s, to convince C to return a result different than $F(x)$ at the end of the protocol; and
4. (t_F, t_S, t_C, cc, mc)*-efficiency*:
   - *client runtime efficiency*: C's runtime, denoted as t_C, should be significantly smaller than the runtime, denoted as t_F, of computing $F(x)$ without delegation;
   - small S's runtime t_S (i.e., a small constant times t_F);
   - small online phase communication complexity cc (i.e., ideally a small constant times input and output sizes);
   - small number of online phase messages mc (i.e., ideally, ≤ 2).

Our Contribution and Comparison with Previous Work. We show single-server protocols, *without preprocessing*, for the delegation of

1. modular multiplication of two public group values,
2. modular inverse of a public group value,
3. modular inverse of a private group value, and

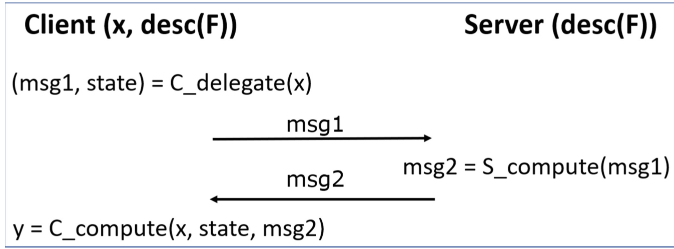

Fig. 1. Delegated computation of $y = F(x)$ without preprocessing

4. exponentiation of a public base to a small public exponent in the RSA group.

All of our protocols satisfy the following 4 properties:

- δ_c-result correctness, for $\delta_c = 1$;
- ϵ_p-input privacy (unless the input is public), for $\epsilon_p = 0$;
- ϵ_s-result security, for $\epsilon_s = 2^{-\lambda}$, where λ is a configurable statistical parameter which in applications can be set, for instance, as $= 50$); and
- client runtime efficiency, with a software implementation that achieves ratio t_F/t_C significantly larger than 1, when $\lambda = 50$ and when the value range of input length σ is consistent with the use of these operations in applied cryptography.

In the case of modular multiplication of two public values, we are not aware of any previous work satisfying these 4 properties without preprocessing. The closest results we know of are: (a) the protocol in [7], which satisfies these 4 properties with preprocessing, and (b) a protocol obtained by a direct adaptation to modular multiplication of the integer multiplication verification test [15,32], which at best achieves client efficiency in an asymptotic sense.

In the case of modular inverses, previous work (see, e.g., [5]) did achieve an efficient delegation protocol without preprocessing, even in the case where the input needs to remain private. The protocols in this paper have improved client efficiency, since the client only performs a few modular reductions with a small modulus, while in [5] the client performed a few modular multiplications.

In the case of modular exponentiation of a public base to a small public exponent in the RSA group [25], we are not aware of any previous work satisfying this set of 4 properties without any preprocessing. The closest results we are aware of are a protocol that satisfies these 4 properties but requires preprocessing [8], and the following protocols without preprocessing for large public exponents: (a) delegation of a batch of exponentiations where the client does compute a single exponentiation [9], and (c) delegation of a single exponentiation which only provably satisfies result correctness and client efficiency [24,28,33]. We note that exponentiation to a small exponent is of much interest since many library implementations of RSA encryption use small exponents, for efficiency reasons.

We show the client runtime efficiency property of our protocols in two ways: with analytical runtime expressions, and by performance measurements for a

software implementation of our protocols as well as some previous protocols (see Tables 1, 2, 3, and 4 for details). Our protocols also perform well with respect to the other targeted efficiency properties; specifically:

- low exchanged communication (a constant number of group values; i.e., $cc = O(1)$ for modular multiplication and inverses, and a logarithmic number; i.e., $cc = O(\log x)$ for modular exponentiation to small exponent x),
- only 1 or 2 exchanged messages (client delegating to server, and server responding; i.e., $mc \leq 2$), and
- low server runtime (only lower order computations in addition to the delegated function).

Our main technical contribution consists of a 2-parameter generalization of Pippenger's probabilistic test [15,32] on efficiently verifying integer equations. Given an integer equation $y = a \cdot b$, this test consisted of checking whether this identity holds modulo a small random prime. We generalize this test in two ways: by using a small random integer instead of a small random prime, and by optimizing the length of this random integer as a function of the desired error probability for the test. We also give a self-contained proof of the lemma proving the effectiveness of this test.

More Related Work. Almost all past work showing proved guarantees in delegation of operations used in cryptography protocols (starting with, e.g., [14] for exponentiation and [30] for pairings), made critical use of preprocessing, as follows. The delegation protocol was divided into an offline phase and an online phase, and the client was assumed to have time resource constraints only in the online phase. While this assumption may be reasonable in many practical scenarios, it may also not be so in scenarios where we cannot guarantee the integrity and/or confidentiality of the data stored on the client's memory at the offline phase or even cached across multiple protocol executions. Thus, delegation without preprocessing of operations used in cryptography schemes, although much harder to achieve, seems to be an important capability to have for applications with resource-constrained devices.

We are only aware of the following few exceptions (i.e., single-server delegation protocols proved to satisfy the above 4 properties, without requiring any preprocessing): a protocol to delegate an inverse in a group (see, e.g., [5]), and a recent protocol to delegate any single pairing computation with public inputs [19]. With respect to batch computations, we are aware of the following solutions proved to satisfy the above 4 properties, without preprocessing: 2 protocols to delegate a batch of public-base, public-exponent, exponentiations in prime-order or RSA groups, where the client does perform a single exponentiation computation [9], and protocols to delegate a batch of public-input pairings, where the client does perform one or some pairing computations [10,23,30]. We note that in these latter protocols the client does cache some values across the batch delegation, and stress that if cached for a long time, the confidentiality and integrity of these values is also at risk (similarly as discussed for values stored during any preprocessing).

In the case of no-preprocessing delegation of a single large-exponent exponentiation modulo a composite integer, previous attempts satisfied result correctness and client efficiency [24,28,33] but were later showed to satisfy neither input privacy nor result security [4,24]. Similarly for the case of no-preprocessing delegation of a single pairing: the scheme in [13] satisfies result correctness, input privacy and client efficiency but does not target result security, and protocol 1 in [20] satisfies result correctness and client efficiency but was showed to satisfy neither input privacy nor result security in [19]. Some literature papers achieved delegation without preprocessing in the presence of 2 or more non-colluding servers; see, for instance, [29] for the delegation of pairings.

There is also much other work on delegation for operations in a different domain than what studied here, for which we refer to reader to the survey in [27] for other operations beyond cryptography and the survey in [2] for computation of arbitrary functions, with clients more powerful than considered here.

Preliminary Definitions. Let $(G, \cdot)$ denote a group, where we refer to operation $\cdot$ as multiplication, and let 1 denote G's identity element. For any $a \in G$, let $b = a^{-1}$ denote the multiplicative inverse of a; i.e., the value b such that $a \cdot b = 1$. We consider the following functions:

- $F_{mul} : G \times G \to G$, mapping any $a, b \in G$ to their multiplication $a \cdot b$.
- $F_{inv} : G \to G$, mapping any value $x \in G$ to its multiplicative inverse x^{-1}.
- $F_{exp,c} : G \times \{0,1\}^c \to G$, mapping any values $x \in G$ and any c-bit exponent e to the exponentiation x^e.

In the rest of the paper, we will consider these functions over the group $(\mathbb{Z}_m^*, \cdot \bmod m)$, for an arbitrary integer m. In particular, when m is a positive integer of one of the following two forms: (1) m is a prime; (2) m is the product of two same-length primes p, q. Note that these definitions capture groups where the discrete logarithm problem or RSA/factoring problems are conjectured to be hard.

For *asymptotic efficiency* evaluation of our protocols, we will use the following definitions:

- $a(\ell)$: runtime for modular addition/subtraction of ℓ-bit values
- $m(\ell)$: runtime for modular multiplication of ℓ-bit values
- $d(\ell)$: runtime for modular inversion of an ℓ-bit value
- $m_r(\ell)$: runtime for modular reduction to an ℓ-bit modulus
- $p(\ell)$: runtime for a random generation of an ℓ-bit prime number
- $i(\ell)$: runtime for a random generation of an ℓ-bit integer
- $\eta_1 = \lceil \lambda + \log_2 \lambda + \log_2(\pi(2\sigma)) \rceil$, where $\pi(z)$ is the number of primes $\leq z$
- $\eta_2 = \lceil \lambda + \log_2 \sigma \rceil$.

For *practical runtime* evaluation, we have produced a software implementation, in Python 3.8 using the gmpy2 package, of our protocols on a macOS Big Sur Version 11.4 laptop with a 3.2 GHz Apple M1 processor with 8 cores (4 performance cores and 4 efficiency cores at 1/10th of the power) and 16 GB RAM.

2 An Identity Verification Test Modulo Small Integers

We show a 2-parameter generalization of Pippenger's probabilistic test [15,32]. Given an integer equation $y = a \cdot b$, this test consisted of randomly choosing a small prime q, and checking whether $y \bmod q = (a \bmod q) \cdot (b \bmod q)$. We generalize this test in two ways: by using a small random integer s instead of the small random prime q, and by setting the length of s as a function of the desired error probability for the test. We now show the key lemma proving the effectiveness of this test (stated in terms of zero testing since verifying the integer equation $y = a \cdot b$ is equivalent to verifying that $y - a \cdot b = 0$).

Lemma 1. Let λ, σ be integers such that $\lambda \geq 2$ and $7 \leq \sigma \leq 10^8$. Also, let N_η be the set of positive integers $\leq 2^\eta$ and > 1. For any integer x such that $1 \leq x \leq 2^\sigma$, if $\eta = \lceil \lambda + \log_2 \sigma \rceil$, it holds that

$$\text{Prob}\left[\, q \leftarrow N_\eta \,:\, x = 0 \mod q \,\right] \leq 2^{-\lambda}.$$

We start the proof of Lemma 1 by providing some definitions and facts. Our goal is to compute an upper bound on the probability, denoted as $p_{\eta,\lambda,x}$, in the lemma statement; i.e., the probability that after randomly choosing an integer $q \leq 2^\eta$ and > 1, it holds that $x = 0 \mod q$. To compute an upper bound on $p_{\eta,\lambda,x}$, we first elaborate on known bounds on the product of prime numbers.

Theorem 18 in [26] states that for any integer $u < 10^8$, the product of all prime integers $\leq u$ is $> e^t$, for $t = u - 2.05282\sqrt{u}$, which is $> 2^u$ for all integers $u \geq 49$. By direct calculation, one can see that the product of all prime integers $\leq u$ is $> 2^u$ for all $7 \leq u \leq 49$. This implies the following

Fact 1. For any integer u such that $7 \leq u \leq 10^8$, the product of all prime integers $\leq u$ is $> 2^u$.

We now need a result almost identical to Corollary 1 in [18].

Fact 2. For any integers σ, x such that $7 \leq \sigma \leq 10^8$ and $x < 2^\sigma$, the number of positive integers that divide x is $\leq \sigma - 1$.

To show why Fact 2 holds, we see that by assuming that there are $b > \sigma - 1$ distinct positive integers $q_1, \ldots, q_b$ which divide x, one reaches the contradiction

$$\begin{aligned} 2^\sigma &> x \\ &\geq \text{lcm}(q_1, \ldots, q_b) \\ &\geq \text{lcm}(\text{smallest } b \text{ positive integers}) \\ &\geq \text{product of all primes} \leq b \\ &> 2^\sigma \end{aligned}$$

where the last inequality follows from Fact 1, after setting $u = \sigma$.

Using the above facts, we can now compute the desired upper bound on probability $p_{\eta,\lambda,x}$ as follows.

$$p_{\eta,\lambda,x} \le \frac{\text{number of positive integers dividing } x}{\text{number of integers } \le 2^{\eta} \text{ and } > 1}$$
$$\le \frac{\sigma - 1}{2^{\eta} - 1} \le \frac{\sigma}{2^{\eta}} \le \frac{\sigma}{2^{\lambda + \log_2 \sigma}} \le \frac{1}{2^{\lambda}},$$

where the first inequality follows by the definition of $p_{\eta,\lambda,x}$, the second inequality follows from Fact 2, the fourth inequality follows from the definition of η in the lemma statement, and the third and fifth inequalities follow by algebraic simplifications. □

3 No-Preprocessing Delegation of Group Multiplication

In this section we show the first single-server delegation protocols for group multiplication, without any preprocessing. Formally, we obtain the following

Theorem 1. Let σ be computational security parameter, let m be a σ-bit integer, and let λ be a statistical security parameter. There exist (constructively) a single-server protocol $\mathcal{P}^{mul}$ without preprocessing for delegating computation of function F_{mul} in group $(\mathbb{Z}_m^*, \cdot \mod m)$, satisfying the properties of 1-correctness, $2^{-\lambda}$-security, and (t_F, t_S, t_C, mc, cc)-efficiency, where, for $\eta = \lceil \lambda + \log_2 \sigma \rceil$,

- $t_C = 5$ η-bit-modulus reductions of σ-bit integers

  $+ 2$ η-bit-values multiplications $+ 1$ η-bit-value addition;
- $t_S = 1$ multiplication $+ 1$ division mod m,
- $mc = 1$, $cc = 2$

We also remark that the asymptotic expression of t_C is $O(m_r(\eta) + m(\eta) + a(\eta))$, which improves over non-delegated computation runtime $t_F = m(\sigma)$ of ring multiplication for a large region of the (λ, σ) parameter space, including values of highest practical interest, when using the most recommended algorithms in applied cryptography (i.e., Karatsuba's algorithm, Toom-Cook's algorithm and the grade-school algorithm).

In the rest of this section we show the proof of Theorem 1.

Informal Description of $\mathcal{P}^{mul}$. Our starting point is the delegation protocol, *with* preprocessing, for multiplication modulo primes from [7], here denoted as $\mathcal{P}^{mul}_{pre}$. In this latter protocol, the online input to C and S consists of two integers a, b and a prime modulus p, and its online phase starts with S computing the product $w = a \cdot b$ over the integers and sending to C the decomposition of w modulo p (i.e., the quotient w_0 and the remainder w_1 of the division of w by p). After that, C verifies the equation $a \cdot b = w_0 \cdot p + w_1$ modulo a small random prime q, which was chosen in the offline phase. Protocol $\mathcal{P}^{mul}_{pre}$ uses a verification test which extended a well-known test for probabilistic verification of multiplication over the integers, mentioned by Yao [32] and Kaminski [15], and credited in

both papers to Pippenger (see, e.g., example 2 in [15] for a description of the protocol). The extension consists in a configurable choice of the size of the small prime modulus q, based on the desired error probability for the test.

As the offline of $\mathcal{P}^{mul}_{pre}$ essentially only consists of randomly choosing a small prime and storing it on C's memory, a natural approach to obtain a delegation protocol *without* preprocessing consists of moving this step into C's program in the online phase. We denote the resulting protocol as $\mathcal{P}^{mul}_{opm}$. As detailed in Table 1, when implementing $\mathcal{P}^{mul}_{opm}$ for practical parameter values, C's runtime is significantly slower than a non-delegated computation. After we realized that this is due mainly to the random choice and testing of the small prime, we considered using a version of this probabilistic verification test based on arbitrary small integers as moduli (instead of primes), as in Sect. 2, for which the random modulus choice is very efficient and no primality testing of the modulus is needed. Our Lemma 1 shows that this approach is sound, and the size of this integer is not much different (in fact, slightly smaller) than the size of the prime integer chosen in [7]. As a consequence, in the resulting delegation protocol, denoted as $\mathcal{P}^{mul}$, C's runtime is smaller than non-delegated computation, even if no preprocessing is used.

Formal Description of $\mathcal{P}^{mul}$. Consider the group $(\mathbb{Z}^*_m, \cdot \mod m)$, for some positive integer m. We now formally describe a 1-server protocol $\mathcal{P}^{mul} = (C, S)$ for the delegation of multiplication of public online group values a and b in $\mathbb{Z}^*_m$, where $|a| = |b| = \sigma$, and with statistical parameter λ.

Online Input to C and S: $1^\sigma, 1^\lambda$, integer $m \in \{0,1\}^\sigma$, $a, b \in \mathbb{Z}^*_m$

Online Phase Instructions:

1. S computes $w := a \cdot b$ (i.e., the product, over $\mathbb{Z}$, of integers a and b)
   S computes w_0, w_1 such that $w = w_0 \cdot m + w_1$ (over $\mathbb{Z}$), where $0 \le w_1 < m$
   S sends w_0, w_1 to C
2. C randomly chooses an integer $s < 2^\eta$, where $\eta = \lceil \lambda + \log_2 \sigma \rceil$
   C computes $w'_0 := w_0 \mod s$ and $w'_1 := w_1 \mod s$
   C computes $a' := a \mod s$, $b' := b \mod s$ and $m' := m \mod s$
   If $a' \cdot b' \neq w'_0 \cdot m' + w'_1 \mod s$ then
   C returns: $\perp$ and the protocol halts
   C returns: $y := w_1$

Properties of $\mathcal{P}^{mul}$: The proofs for the result correctness and result security of $\mathcal{P}^{mul}$, the latter using Lemma 1, can be found in Appendix B.

The *efficiency* property follows by protocol inspection. In particular, S computes one multiplication of two σ-bit values over $\mathbb{Z}$, and one reduction of a σ-bit integer modulo m, and C computes five reductions modulo the η-bit integer s of integers of size at most σ and one verification check which requires one addition and two multiplications modulo the η-bit integer s.

In Table 1 we report on the practical efficiency of the scheme, based on our software implementation of the scheme, one main takeaway being that C's runtime t_C is smaller than non-delegated computation t_F (i.e., the delegation

improvement ratio t_F/t_C is > 1) for all values of most interest of parameters λ, σ; specifically, $\lambda = 50$ and $\sigma \in \{1024, 2048, 3072\}$. We also report the ratio t_F/t_C for two related protocols: protocol $\mathcal{P}^{mul}_{pre}$ from [7] with preprocessing, and the protocol $\mathcal{P}^{mul}_{opm}$ which has no preprocessing, where C chooses a prime modulus s in the online phase. The takeaways there are that, for such practical parameter values: (1) delegation would not improve C's runtime in $\mathcal{P}^{mul}_{opm}$; (2) delegation does improve C's runtime in $\mathcal{P}^{mul}$ by a multiplicative factor between 1.7 and 4 depending on the modulus size; and (3) the delegation improvement ratio for $\mathcal{P}^{mul}$, not using preprocessing, is about half the ratio of $\mathcal{P}^{mul}_{pre}$, which does use preprocessing, or larger.

Table 1. Performance results for the delegation of $F_{mul}(a, b) = a \cdot b \mod m$ in $\mathbb{Z}^*_m$, where $|m| = \sigma$, $\lambda = 50$, $t_{Cm}(\eta) \leq 5m_r(\eta) + 2m(\eta) + a(\eta)$ and $t_F = 1.19\text{E-}05$ s.

Protocol/Pre-processing		t_C	t_F/t_C			$\sigma = 3072$	
			$\sigma = 2048$	$\sigma = 3072$	$\sigma = 4096$	t_P	t_C
m is a prime integer							
$\mathcal{P}^{mul}_{pre}$	Yes	$t_{Cm}(\eta_1)$	3.479	4.852	6.578	5.78E-05	2.44E-06
$\mathcal{P}^{mul}_{opm}$	No	$t_{Cm}(\eta_1) + p(\eta_1)$	0.108	0.185	0.314	0	6.43E-05
$\mathcal{P}^{mul}$	No	$t_{Cm}(\eta_2) + i(\eta_2)$	1.694	2.716	3.973	0	4.43E-06
m is the product of 2 same-length primes							
$\mathcal{P}^{mul}_{pre}$	Yes	$t_{Cm}(\eta_1)$	3.538	4.728	6.501	6.46E-05	2.53E-06
$\mathcal{P}^{mul}_{opm}$	No	$t_{Cm}(\eta_1) + p(\eta_1)$	0.122	0.198	0.306	0	6.06E-05
$\mathcal{P}^{mul}$	No	$t_{Cm}(\eta_2) + i(\eta_2)$	1.731	2.701	3.999	0	4.34E-06

4 No-Preprocessing Delegation for Group Inverses

In this section we present single-server protocols for delegated computation of group inverses which have improved client efficiency over previous work. Our protocols build on the multiplication delegation protocol in Sect. 3. Formally, our result is the following

Theorem 2. Let σ be computational security parameter, let m be a σ-bit integer, and let λ be a statistical security parameter. There exist (constructively) two single-server protocols $\mathcal{P}^{inv}_1$, for input scenario 'x public online', and $\mathcal{P}^{inv}_2$, for input scenario 'x private online', for delegating computation of function F_{inv}, in group $(\mathbb{Z}^*_m, \cdot \mod m)$, satisfying the properties of 1-result-correctness, $2^{-\lambda}$-result-security, and (t_F, t_S, t_C, mc, cc)-efficiency, where, for $\eta = \lceil \lambda + \log_2 \sigma \rceil$,

- for $\mathcal{P}^{inv}_1$: $\epsilon_s = 2^{-\lambda}$, $t_F = 1$ inversion,
  $t_C = 5$ η-bit-modulus reductions + 2 η-bit-values multiplications
  + 1 η-bit-value addition,
  $t_S = 1$ inversion + 1 multiplication + 1 division mod m,
  $mc = 1$, $cc = 3$

– for $\mathcal{P}_2^{inv}$ (also satisfying input-privacy): $\epsilon_s = 2^{-\lambda}$, $t_F = 1$ inversion,
  $t_C = 2$ group multiplications + 5 η-bit-modulus reductions
  + 2 η-bit-values multiplications + 1 η-bit-value addition,
  $t_S = 1$ inversion + 1 multiplication + 1 division mod m,
  $mc = 2$, $cc = 4$.

In the rest of this section we prove Theorem 2, by describing the two claimed protocols in the two different input scenarios and their properties. Specifically, we describe delegation of inversion $F_{inv}(x) = x^{-1} \mod m$ in group $(\mathbb{Z}_m^*, \cdot \mod m)$, using a protocol $\mathcal{P}^{mul}$ for delegation of multiplication $F_{mul}(a, b) = a \cdot b \mod m$, such as the protocol from Sect. 3, where inputs a and b are public online.

4.1 The "x Public Input" Scenario

Our first protocol consists of a single message by the server including the inverse value $x^{-1} \mod m$ of the input x and the client delegating the computation of the product $x \cdot x^{-1} \mod m$, using protocol $\mathcal{P}^{mul}$ from Sect. 3, and checking that the result obtained at the end of this protocol execution is equal to 1.

Formal Description of Protocol $\mathcal{P}_1^{inv}$.

Input Scenario: x public online

Online Input to C and S: σ, λ, $desc(F_{inv})$, x

Online Phase Instructions:

1. S computes $w := x^{-1} \mod m$ and sends w to C
2. C and S use protocol $\mathcal{P}^{mul}$, for $a = x$ and $b = w$, and parameters σ, λ, resulting in C obtaining z;
3. If $z \neq 1$ then C **returns** $\perp$ and the protocol halts
4. C **returns** $y := w$ and halts.

Properties of $\mathcal{P}_1^{inv}$: The proofs for the result correctness and result security properties, the latter using Lemma 1, can be found in Appendix C.

The *efficiency* properties follow directly by protocol inspection and the same properties of $\mathcal{P}^{mul}$. In particular, we note that if $\mathcal{P}^{mul}$ consists of a single message from S to C, as the protocol in Sect. 3, then so does $\mathcal{P}_1^{inv}$.

In Table 2 we report on the practical efficiency of the scheme, based on our software implementation of $\mathcal{P}_1^{inv}$, one main takeaway being that C's runtime t_C is smaller than non-delegated computation t_F (i.e., the delegation improvement ratio t_F/t_C is > 1) for all values of most interest of parameters λ, σ; specifically, $\lambda = 50$ and $\sigma \in \{2048, 3072, 4096\}$. We also report the ratio t_F/t_C for two related protocols: protocol $\mathcal{P}_{1,pre}^{inv}$ with preprocessing (using the multiplication protocol with preprocessing from [7]), and the protocol $\mathcal{P}_{1,opm}^{inv}$ which has no preprocessing, where C chooses a prime modulus s in the online phase. The takeaways there are that, for such practical parameter values: (1) delegation would not improve C's runtime in $\mathcal{P}_{1,opm}^{inv}$; (2) delegation does improve C's runtime in $\mathcal{P}_1^{inv}$ by a multiplicative factor between 4.8 and 8.1 depending on the modulus size; and (3) the delegation improvement ratio for $\mathcal{P}_1^{inv}$, not using preprocessing, is about half the ratio of $\mathcal{P}_{1,pre}^{inv}$, which does use preprocessing, or larger.

Table 2. Performance results for the delegation of $F(x) = x^{-1} \mod m$ in $\mathbb{Z}_m^*$, where x is public, m is a σ-bit prime, $\lambda = 50$, $t_{Cm}(\eta) \leq 5m_r(\eta) + 2m(\eta) + a(\eta)$ and $t_F = 3.19\text{E-}05$ s.

Protocol/Pre-processing		t_C	t_F/t_C			$\sigma = 3072$	
			$\sigma = 2048$	$\sigma = 3072$	$\sigma = 4096$	t_P	t_C
$\mathcal{P}_{1,pre}^{inv}$	Yes	$t_{Cm}(\eta_1)$	10.008	13.394	16.235	6.30E-05	2.35E-06
$\mathcal{P}_{1,opm}^{inv}$	No	$t_{Cm}(\eta_1) + p(\eta_1)$	0.359	0.488	0.816	0	6.73E-05
$\mathcal{P}_1^{inv}$	No	$t_{Cm}(\eta_2) + i(\eta_2)$	4.782	6.543	8.111	0	4.79E-06

4.2 The "x Private Input" Scenario

Our second protocol $\mathcal{P}_2^{inv}$ starts with the client sending a randomized version of the input to the server. Then it continues with the client delegating the computation of the inverse of the randomized input, using our first protocol $\mathcal{P}_1^{inv}$. Finally, the client derives the result by removing the randomizer. Input masking techniques have already been used in many delegation protocols in the literature. In the case of inverse delegation, it is interesting to note that it does not require the client to store any preprocessing values.

Formal Description of Protocol $\mathcal{P}_2^{inv}$.

Input Scenario: x private online

Online Input to C σ, $desc(F_{inv})$, x

Online Input to C σ, $desc(F_{inv})$

Online Phase Instructions:

1. C randomly chooses $r \in G$, computes $z := x \cdot r \mod m$, and sends z to S;
2. S computes $w := z^{-1}$ and sends w to C
3. C and S use protocol $\mathcal{P}^{mul}$ for $a = z$ and $b = w$, and parameters σ, λ, resulting in C obtaining v;
4. If $v \neq 1$ then C **returns** $\perp$ and the protocol halts
5. C **returns** $y := r \cdot w \mod m$ and halts.

Properties of $\mathcal{P}_2^{inv}$: The proofs for the result correctness, input privacy and result security properties, the latter using Lemma 1, can be found in Appendix D.

The *efficiency* properties of $\mathcal{P}_2^{inv}$ follow directly by protocol inspection and the same properties of $\mathcal{P}_1^{inv}$ and $\mathcal{P}^{mul}$. In particular, note that t_C only increases by $2 \cdot m(\sigma)$ with respect to protocol $\mathcal{P}_1^{inv}$.

In Table 3 we report on the practical efficiency of the scheme, based on our software implementation of $\mathcal{P}_2^{inv}$, where we reach analogue conclusions as for $\mathcal{P}_1^{inv}$ on the effectiveness of delegation.

5 No-Preprocessing Delegation for Small-Exponent Exponentiation in RSA Groups

We discuss the first protocol to delegate small-exponent exponentiation in the RSA group $\mathbb{Z}_n^*$, in the input case where both the base and the exponent are

Table 3. Performance results for the delegation of $F(x) = x^{-1} \mod m$ in $\mathbb{Z}_m^*$, where x is private, m is a σ-bit prime, $\lambda = 50$, and $t_F = 3.11\text{E-}05$ s

Protocol/Pre-processing		t_F/t_C			$\sigma = 3072$	
		$\sigma = 2048$	$\sigma = 3072$	$\sigma = 4096$	t_P	t_C
$\mathcal{P}_{2,pre}^{inv}$	Yes	2.977	3.110	2.858	6.70E-05	1.00E-05
$\mathcal{P}_{2,opm}^{inv}$	No	0.606	0.791	1.013	0	3.93E-05
$\mathcal{P}_2^{inv}$	No	2.504	2.832	2.653	0	1.09E-05

public, without preprocessing. This is obtained by carefully combining a specific variant of the square-and-multiply algorithm with the delegation protocol $\mathcal{P}^{mul}$ for multiplication from Sect. 3. Formally, we obtain the following

Theorem 3. Let σ be computational security parameter, let m be a σ-bit integer, let λ be a statistical security parameter, and let e be a $c - bit$ integer, where c is constant with respect to σ and λ. There exist (constructively) a single-server protocols $\mathcal{P}^{exp}$, for input scenario 'x and e public', for delegating computation of function F_{exp}, in group $(\mathbb{Z}_m^*, \cdot \mod m)$, satisfying the properties of 1-result-correctness, $2^{-\lambda}$-result-security, and (t_F, t_S, t_C, mc, cc)-efficiency, where, for $\eta = \lceil \lambda + \log_2 \sigma \rceil$,

- $\epsilon_s = 2^{-\lambda}$, $t_F = 1$ exponentiation to a c-bit exponent,
- $t_C = O(c)$ η-bit-modulus reductions of σ-bit integers

  $+ O(c)$ η-bit-values multiplications
- $t_S = O(c)$ multiplications and divisions mod m,
- $mc = 1$, $cc \leq 8c$.

Informal Description of $\mathcal{P}^{exp}$. Our protocol $\mathcal{P}^{exp}$ can be seen as an optimized simulation of the (iterative) square and multiply algorithm for modular exponentiation, while using a multiplication delegation subprotocol, such as protocol $\mathcal{P}^{mul}$ in Sect. 3, to compute squares and multiplications modulo n in this algorithm. A similar approach has already been taken in [8], where, however, both protocols for multiplication and exponentiation did need a preprocessing phase including the generation and storage of the small prime modulus.

The natural approach to remove the preprocessing phase would be similar as for the results in Sects. 2 and 3: replacing the small prime modulus with a small integer modulus, and letting C generate the small modulus during the protocol (as opposed to doing that in some preprocessing phase). It turns out that this is not yet sufficient to achieve effective delegation (i.e., for the delegation improvement ratio t_F/t_C to be > 1), and further optimizations are needed. Thus, we consider the iterative protocol structure of the square and multiply algorithm, and try to let the client run as many operations as possible only once instead of once for each multiplication, as it would happen on a direct simulation of the algorithm, with calls to protocol $\mathcal{P}^{mul}$ from Sect. 3. In particular, we apply the following 3 optimizations:

1. C only chooses the small integer modulus s once;
2. C only compute the reduction of m modulo the small modulus s once.
3. The protocol uses the version of the square and multiply algorithm where the exponent is expressed in binary and the 'multiply' part of the algorithm only multiplies the value computed until then by a fixed number (i.e., the input base); accordingly, C only computes the reduction of the input base modulo the small integer s once, instead of once for each multiplication operation carried out for each exponent bit $= 1$.

Formal Description and Properties of $\mathcal{P}^{exp}$. A formal description of $\mathcal{P}^{exp}$ is very similar as the construction in [8] and is detailed in Appendix E.

The result correctness and result security properties of $\mathcal{P}^{exp}$, the latter using Lemma 1, follow from the above informal description.

In Table 4 we report on the practical efficiency of the scheme, based on our software implementation of $\mathcal{P}^{exp}$, where we reach analogue conclusions as for our previous protocols, on the effectiveness of delegation.

Table 4. Performance results for the delegation of $F(x) = x^e \mod n$, when x and e are public, m is a σ-bit product of 2 same-length primes, e is a c-bit integer, and $\lambda = 50$.

Protocol/Pre-processing		t_F/t_C			
		$c = 8$		$c = 32$	
		$\sigma = 2048$	$\sigma = 3072$	$\sigma = 2048$	$\sigma = 3072$
$\mathcal{P}^{exp}_{pre}$	Yes	1.543	2.652	1.157	2.240
$\mathcal{P}^{exp}_{opm}$	No	0.288	0.571	0.500	0.782
$\mathcal{P}^{exp}$	No	1.448	2.716	1.532	2.586

6 Conclusions

We showed single-server protocols, without preprocessing, for the single-server delegation of the following operations, used in several cryptosystems:

1. modular multiplication of two public group values,
2. modular inverse of a public group value,
3. modular inverse of a private group value, and
4. exponentiation of a public base to a small public exponent in the RSA group.

Our protocols satisfy result-correctness, input-privacy (unless the input is public), result-security, and client-efficiency (with respect to non-delegated computation) for parameter values of interest in cryptography applications. To the best of our knowledge, the only other single-server protocols in the literature satisfying these properties were presented for the delegation of:

1. a pairing, with both inputs being public, and
2. a batch of public-base and public-exponent exponentiation operations in discrete-log and RSA groups.

Several open problems remain in this area of single-server delegation without preprocessing, especially with respect to operations where input privacy is required, and our results should be interpreted as a step in this direction.

Acknowledgements. Work by Matluba Khodjaeva was supported by the NSF CNS - CISE MSI Research Expansion grant N2131182 and PSC CUNY Cycle 53. Work by Giovanni Di Crescenzo, was supported by the Defense Advanced Research Projects Agency (DARPA), contract n. HR001120C0156. Approved for Public Release, Distribution Unlimited. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright annotation hereon. Disclaimer: The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of DARPA, or the U.S. Government.

A Formal Definitions

In this section we recall the formal definition (based on [12,14]), of delegation protocols and their correctness, privacy, security, and efficiency requirements.

Basic Notations. The expression $z \leftarrow T$ denotes randomly and independently choosing z from set T. By $y \leftarrow A(x_1, x_2, \ldots)$ we denote running the algorithm A on input $x_1, x_2, \ldots$ and any random coins, and returning y as output. By $(y, tr) \leftarrow (A(u_1, u_2, \ldots), B(v_1, v_2, \ldots))$ we denote running the interactive protocol between A, with input $u_1, u_2, \ldots$ and any random coins, and B, with input $v_1, v_2, \ldots$ and any random coins, where tr denotes A's and B's messages in this execution, and y is A's final output.

System Scenario: Entities and Protocol. We consider a system with a single client, denoted by C, and a single server, denoted by S, who are connected by an authenticated channel, and therefore do not consider any integrity or replay attacks on this channel. Differently than much of previous work in the area, we consider a delegation protocol without *offline phase* or *preprocessing* client computations, typically storing extra values in client's memory, and only consider client computations in what is also called *online phase* in the literature, where C has time constraints.

Let σ denote the computational security parameter (derived from hardness considerations of the underlying computational problem), and let λ denote the statistical security parameter (defined so that statistical test failure events with probability $2^{-\lambda}$ are extremely rare). Both parameters are expressed in unary notation (i.e., $1^\sigma, 1^\lambda$). We think of σ as being asymptotically much larger than λ. Let F denote a function and $desc(F)$ denote F's description. Assuming $1^\sigma, 1^\lambda, desc(F)$ are known to both C and S, we define a *client-server protocol for the delegated (n-instance) computation of F* as the execution: $\{(y, tr) \leftarrow (C(x), S)\}$, where both parties are assume to be aware of inputs

$(1^\sigma, 1^\lambda, desc(F))$, which we will ofter omit for brevity, and tr is the transcript of the communication exchanged between C and S.

Correctness Requirement. Informally, the correctness requirement states that if both parties follow the protocol, C obtains some output at the end of the protocol, and this output is, with high probability, equal to the value obtained by evaluating function F on C's input. Formally, we say that a no-preprocessing client-server protocol (C, S) for the delegated computation of F satisfies δ_c-*correctness* if for any x in $Dom(F)$,

$$\text{Prob}\left[\, out \leftarrow \text{CorrExp}_\text{F} \;:\; out = 1 \,\right] \geq \delta_c,$$

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

1. $(y, tr) \leftarrow (C(x), S)$
2. if $y = F(x)$, then **return:** 1 else **return:** 0

Privacy Requirement. Informally, the privacy requirement should guarantee the following: if C follows the protocol, a malicious adversary corrupting S cannot obtain any information about C's input x from a protocol execution. This is formalized by extending the indistinguishability-based approach typically used in definitions for encryption schemes. Let (C, S) be a no-preprocessing client-server protocol for the delegated computation of F. We say that (C, S) satisfies ϵ_p*-privacy (in the sense of indistinguishability) against a malicious adversary* if for any algorithm A, it holds that

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

for some ϵ_p close to 0, where experiment PrivExp is:

1. $(x_0, x_1, aux) \leftarrow A(desc(F))$
2. $b \leftarrow \{0, 1\}$
3. $(y, tr) \leftarrow (C(x_b), A(aux))$
4. $d \leftarrow A(tr, aux)$
5. if $b = d$ then **return:** 1 else **return:** 0.

Security Requirement. Informally, the security requirement states that for any efficient and malicious adversary corrupting S and even choosing C's input tuple x, at the end of the protocol, C cannot be convinced to obtain some output tuple z containing a value $z \neq F(x)$ Formally, we say that the client-server protocol (C, S) for the delegated n-instance computation of F satisfies ϵ_s*-security against a malicious adversary* if for any algorithm A,

$$\text{Prob}\left[\, out \leftarrow \text{SecExp}_\text{F,A} \;:\; out = 1 \,\right] \leq \epsilon_s,$$

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

1. $(\vec{x}, aux) \leftarrow A(desc(F))$

2. $(\vec{z}, tr) \leftarrow (C(x), A(aux))$
3. if $z \in \{\perp, F(x)\}$ then **return:** 0 else **return:** 1.

We consider different input scenarios, where the input x may be *private* or *public*. The above definition considered the "x private" input scenario. The definition for the "x public" input scenario is obtained by the following slight modifications: (1) S is also given x as input; (2) no input privacy is required.

B Properties of $\mathcal{P}^{mul}$

The *correctness* property follows by observing that if C and S follow the protocol, then S computes w_0, w_1 as $w = a \cdot b = w_0 \cdot m + w_1$ and the equation $a \cdot b = w_0 \cdot m + w_1$ is satisfied over $\mathbb{Z}$ and is therefore satisfied also modulo the small prime s. This prevents C to return $\perp$, and allows C to return the correct output value $w_1 = w \mod m = a \cdot b \mod m$.

To prove the *security* property against any malicious S we need to compute an upper bound ϵ_s on the security probability that an adversary corrupting S convinces C to output a y such that $y \neq a \cdot b \mod m$.

We continue the proof of the unbounded security property by defining the following events:

- $e_{y,\neq}$, defined as "C outputs y such that $y \neq a \cdot b \mod m$"
- e_t, defined as "S's message contains w_0, w_1 such that $a \cdot b \neq w_0 \cdot m + w_1 \mod m$".

We now compute an upper bound on the probability of event $e_{y,\neq}$, conditioned on event e_t. We observe that, when event e_t is true, it holds that $a \cdot b \mod m \neq w_1$. In this scenario, for event $e_{y,\neq}$ to happen, it needs to hold that

$$(a \mod s)(b \mod s) = (w_0 \mod s)(m \mod s) + w_1 \mod s.$$

This happens when

$$(a \cdot b - w_0 \cdot m - w_1) = 0 \mod s.$$

By setting $x = a \cdot b - w_0 \cdot m - w_1$, and applying Lemma 1 for this value of x, we obtain that the probability that $x = 0 \mod s$ is at most $2^{-\lambda}$, which implies the following

Fact 3. $\text{Prob}\,[\, e_{y,\neq} \,|\, e_t \,] \leq 2^{-\lambda}$

We then observe that when event e_t is false, then the message from S follows the protocol and therefore $e_{y,\neq}$ is also false. This implies the following

Fact 4. $\text{Prob}\,[\, e_{y,\neq} \,|\, \neg e_t \,] = 0$

We can now compute an upper bound on the probability of event $e_{y,\neq}$. We have that Prob $[\, e_{y,\neq} \,]$ is

$$\begin{aligned} &= \text{Prob}\,[\, e_t \,]\, \text{Prob}\,[\, e_{y,\neq} | e_t \,] + \text{Prob}\,[\, \neg\, e_t \,]\, \text{Prob}\,[\, e_{y,\neq} | \neg\, e_t \,] \\ &\leq \text{Prob}\,[\, e_{y,\neq} | \neg\, e_t \,] + \text{Prob}\,[\, e_{y,\neq} | \neg\, e_t \,] \\ &\leq \text{Prob}\,[\, e_{y,\neq} | \neg\, e_t \,] \;\leq\; 2^{-\lambda}, \end{aligned}$$

where the first equality and the first inequality follow from basic probability facts; the second inequality follows by applying Fact 4, and the last inequality follows by applying Fact 3.

C Properties of $\mathcal{P}_1^{inv}$

The *result correctness* property follows directly by observing that if C and S follow the protocol, the same property of $\mathcal{P}^{mul}$ implies that

$$z = a \cdot b \mod m = x \cdot w \mod m = x \cdot (x^{-1}) \mod m = 1,$$

after which C returns $y = w = x^{-1} \mod m$.

To prove the *result security* property against any malicious S we need to compute an upper bound ϵ_s on the security probability that an adversary corrupting S convinces C to output a y such that $y \neq x^{-1} \mod m$. Assume this adversary sends w' to C and runs $\mathcal{P}^{mul}$ with C, resulting in C obtaining z'. Now, because C checks whether $z' \neq 1$, the only possible cheating strategy for the adversary is that of convincing C to accept that $z' = 1$ and z' is the product of x and w', even when w' is not the inverse of x. By the result security property of $\mathcal{P}^{mul}$, this can only happen with probability at most $2^{-\lambda}$.

D Properties of $\mathcal{P}_2^{inv}$

The *result correctness* property follows directly by observing that if C and S follow the protocol, the same property of $\mathcal{P}^{mul}$ implies that

$$v = z \cdot w \mod m = (x \cdot r) \cdot z^{-1} \mod m = (x \cdot r) \cdot (x \cdot r)^{-1} \mod m = 1,$$

after which C returns $y = r \cdot w = r \cdot (x \cdot r)^{-1} = r \cdot r^{-1} \cdot x^{-1} = x^{-1} \mod m$.

The *input privacy* follows by observing that C only sends a random group value to S.

To prove the *result security* property against any malicious S we need to compute an upper bound ϵ_s on the security probability that an adversary corrupting S convinces C to output a y such that $y \neq x^{-1} \mod m$. Assume this adversary, after receiving z from c, sends w' to C and runs $\mathcal{P}^{mul}$ with C, resulting in C obtaining v'. Now, because C checks whether $v' \neq 1$, the only possible cheating strategy for the adversary is that of convincing C to accept that $v' = 1$ and v' is the product of z and w', even when w' is not the inverse of z. By the result security property of $\mathcal{P}^{mul}$, this can only happen with probability $\leq 2^{-\lambda}$.

E Protocol $\mathcal{P}^{exp}$

To formally define protocol $\mathcal{P}^{exp} = (C, S)$ for the delegated computation of $x^e \mod m$, we use definitions and algorithms from protocol $\mathcal{P}^{mul}$ as well as an optimized version of it, as mentioned in Sect. 5 and further discussed below.

First, by $\mathcal{P}^{mul} = (S_m, C_m)$ we denote a protocol for the delegation of function F_{mul} with statistical parameter λ_m, for public inputs a and b, such as the protocol in Sect. 3. In particular, the notation $(q, r) \leftarrow S_m(a, b)$ refers to an execution of the $\mathcal{P}^{mul}$ server's algorithm with inputs a, b, returning message (q, r) for C, such that $a \cdot b = q \cdot m + r$, where $0 \leq r < m$. Similarly, the notation $d \leftarrow C_m(a, b, q, r)$ refers to an execution of the $\mathcal{P}^{mul}$ client's algorithm with inputs a, b, and server's message (q, r), and returning decision bit d where $d = 1/0$ depending on whether C_m accepts/does not accept the statement $r = a \cdot b \mod n$.

While algorithm S will run S^m, algorithm C will run an optimized version of C^m, which reuses the same modulus s, and the same values $m' = m \mod s$ and $x' = x \mod s$, whenever possible across the multiple uses of multiplication delegation within exponentiation delegation, as we now define. Given a randomly chosen η-bit integer s, and values $m' = m \mod s$ and $x' = x \mod s$, we define the notation $d \leftarrow C'_m(a, b, q, r, s, m', x')$ to refer to a variant of algorithm C_m, where the computation of s and m' are replaced by the use of its arguments s, m', and the use of x as a product factor in correspondence of a bit of exponent e being $= 1$ is replaced by the use of its argument x'. Here, by using C'_m, the client only computes the values s, m', x' once, while by using C_m, it would have recomputed each of these values either $\log e$ or about $(\log e)/2$ times.

We now formally describe protocol $\mathcal{P}^{exp}$ to delegate small-exponent exponentiation function $F_{exp,c}$, which maps $x \in \mathbb{Z}_m^*$ to $x^e \mod m$. in a group $\mathbb{Z}_m^*$, where x and e are public, and e has c bits.

Online Input to C and S: $1^\sigma, 1^\lambda, 1^c$, $m \in \{0,1\}^\sigma$, $x \in \mathbb{Z}_m^*$, $e \in \{0,1\}^c$

Online Phase of $\mathcal{P}^{exp}$:

1. S sets $z = x$, $y = 1$ and $i = 1$
2. While $e > 1$ do
   - S computes $(q_{1i}, r_{1i}) = S_m(z, z)$ and sets $z = r_{i1}$
   - if e is even then
     - S sets $q_{2i} = r_{2i} = 0$, $i = i + 1$ and $e = e/2$
   - if e is odd then
     - S computes $(q_{2i}, r_{2i}) = S_m(z, x)$ and sets
     - S sets $z = r_{i2}$, $i = i + 1$ and $e = (e - 1)/2$
3. S sends $((q_{11}, r_{11}, q_{21}, r_{21}), \ldots, (q_{1c}, r_{1c}, q_{2c}, r_{2c})$ to C
4. C sets $i = 1$ and $z = x$
5. C randomly chooses an η-bit integer s, where $\eta = \lceil \lambda + \log_2 \sigma \rceil$
   C computes $m' = m \mod s$ and $x' = x \mod s$
6. While $e > 1$ do
   - if e is even then
     - C computes $d_{1i} = C_m(z, z, q_{1i}, r_{1i}, s, m', x')$

if $d_{1i} = 0$ then C halts
else C sets $z = r_{1i}$, $i = i + 1$ and $e = e/2$ if e is odd then
C computes $d_{1i} = C_m(z, z, q_{1i}, r_{1i}, s, m', x')$ and sets $z = r_{1i}$
C computes $d_{2i} = C_m(z, x', q_{2i}, r_{2i}, s, m', x')$ and sets $z = r_{2i}$
if $d_{1i} = 0$ or $d_{2i} = 0$ then C halts
else C sets $i = i + 1$ and $e = (e - 1)/2$

7. C **returns**: $y = r_{2i}$ and halts

References

1. Abadi, M., Feigenbaum, J., Kilian, J.: On hiding information from an oracle. J. Comput. Syst. Sci. **39**(1), 21–50 (1989)
2. Ahmad, H., et al.: Primitives towards verifiable computation: a survey. Front. Comput. Sci. **12**(3), 451–478 (2018)
3. Bellare, M., Garay, J.A., Rabin, T.: Fast batch verification for modular exponentiation and digital signatures. In: Nyberg, K. (eds.) EUROCRYPT 1998. LNCS, vol. 1403, pp. 236–250. Springer, Heidelberg (1998). https://doi.org/10.1007/BFb0054130
4. Bouillaguet, C., Martinez, F., Vergnaud, D.: Cryptanalysis of modular exponentiation outsourcing protocols. Comput. J. **65**(9), 2299–2314 (2022)
5. Cavallo, B., Di Crescenzo, G., Kahrobaei, D., Shpilrain, V.: Efficient and secure delegation of group exponentiation to a single server. In: Mangard, S., Schaumont, P. (eds.) Radio Frequency Identification. Security and Privacy Issues, pp. 156–173. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-24837-0_10
6. Crandall, R., Pomerance, C.: Prime Numbers: A Computational Perspective, 2nd edn. Springer, New York (2005). https://doi.org/10.1007/0-387-28979-8
7. Di Crescenzo, G., Khodjaeva, M., Shpilrain, V., Kahrobaei, D., Krishnan, R.: Single-server delegation of ring multiplications from quasilinear-time clients. In: Proceedings of SINCONF 2021, pp. 1–8 (2021)
8. Di Crescenzo, G., et al.: On single-server delegation of RSA. In: Bella, G., Doinea, M., Janicke, H. (eds.) SecITC 2022. LNCS, vol. 13809, pp. 81–101. Springer, Cham (2023). https://doi.org/10.1007/978-3-031-32636-3_5
9. Di Crescenzo, G., Khodjaeva, M., Kahrobaei, D., Shpilrain, V.: Computing multiple exponentiations in discrete log and RSA groups: from batch verification to batch delegation. In: Proceedings of CNS 2017, pp. 531–539 (2017)
10. Di Crescenzo, G., Khodjaeva, M., Morales Caro, D.: Single-server batch delegation of variable-input pairings with unbounded client lifetime. In: Proceedings of ADIoT 2023, ESORICS 2023 Workshops, LNCS. Springer, to appear (2023)
11. Feigenbaum, J.: Encrypting problem instances. In: Williams, H.C. (eds.) CRYPTO 1985. LNCS, vol. 218, pp. 477–488. Springer, Heidelberg (1986). https://doi.org/10.1007/3-540-39799-X_38
12. Gennaro, R., Gentry, C., Parno, B.: Non-interactive verifiable computing: outsourcing computation to untrusted workers. In: Rabin, T. (eds.). CRYPTO 2010. LNCS, vol. 6223, pp. 465–488. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-14623-7_25
13. Girault, M., Lefranc, D.: Server-aided verification: theory and practice. In: Roy, B. (eds.). ASIACRYPT 2005. LNCS, vol. 3788, pp. 605–623. Springer, Heidelberg (2005). https://doi.org/10.1007/11593447_33

14. Hohenberger, S., Lysyanskaya, A.: How to securely outsource cryptographic computations. In: Kilian, J. (eds.). TCC 2005. LNCS, vol. 3378, pp. 264–282. Springer, Heidelberg (2005). https://doi.org/10.1007/978-3-540-30576-7_15
15. Kaminski, M.: A note on probabilistically verifying integer and polynomial products. J. ACM **36**(1), 142–149 (1989)
16. Karatsuba, A., Ofman, Y.: Multiplication of many-digital numbers by automatic computers. Proc. USSR Acad. Sci. **145**, 293–294 (1963). Translation in Physics-Doklady **7**, 595–596 (1963)
17. Karatsuba, A.A.: The complexity of computations. Proc. Steklov Inst. Math. **211**, 169–183 (1995). Translation from Trudy Mat. Inst. Steklova **211**, 186–202 (1995)
18. Karp, R.M., Rabin, M.O.: Efficient randomized pattern-matching algorithms. In: Rep. TR-31-81. Harvard Univ. Center for Research in Computing Technology, Cambridge (1981)
19. Khodjaeva, M., Di Crescenzo, G.: On single-server delegation without precomputation. In: Proceedings of 20th International Conference on Security and Cryptography, SECRYPT 2023, ScitePress, pp. 540–547 (2023)
20. Kalkar, O., Sertkaya, I., Tutdere, S.: On the batch outsourcing of pairing computations. Comput. J. **66**(10), 2437–2446 (2022)
21. Liu, J.K., Au, M.H., Susilo, W.: Self-generated-certificate public-key cryptography and certificateless signature/encryption scheme in the standard model. In: Proceedings of the ACM Symposium on Information, Computer and Communications Security. ACM Press (2007)
22. Matsumoto, T., Kato, K., Imai, H.: Speeding up secret computations with insecure auxiliary devices. In: Goldwasser, S. (eds.). CRYPTO 1988. LNCS, vol. 403, pp. 497–506. Springer, New York (1990). https://doi.org/10.1007/0-387-34799-2_35
23. Mefenza, T., Vergnaud, D.: Verifiable outsourcing of pairing computations. Technical report (2018)
24. Rangasamy, J., Kuppusamy, L.: Revisiting single-server algorithms for outsourcing modular exponentiation. In: Chakraborty, D., Iwata, T. (eds.). INDOCRYPT 2018. LNCS, vol. 11356, pp. 3–20. Springer, Cham (2018). https://doi.org/10.1007/978-3-030-05378-9_1
25. Rivest, R., Shamir, A., Adleman, L.: A method for obtaining digital signatures and public-key cryptosystems. Commun. ACM **21**(2), 120–126 (1978)
26. Rosser, J., Schoenfeldl, L.: Approximate formulas for some functions of prime numbers. Ill. J. Math. **6**, 64–94 (1962)
27. Shan, Z., Ren, K., Blanton, M., Wang, C.: Practical secure computation outsourcing: a survey. ACM Comput. Surv. **51**(2), 31:1–31:40 (2018)
28. Su, Q., Zhang, R., Xue, R.: Secure outsourcing algorithms for composite modular exponentiation based on single untrusted cloud. Comput. J. **63**, 1271 (2020)
29. Tong, L., Yu, J., Zhang, H.: Secure outsourcing algorithm for bilinear pairings without pre-computation. In: Proceedings of IEEE DSC (2019)
30. Tsang, P.P., Chow, S.S.M., Smith, S.W.: Batch pairing delegation. In: Miyaji, A., Kikuchi, H., Rannenberg, K. (eds.) Advances in Information and Computer Security. LNCS, vol. 4752, pp. 74–90. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-75651-4_6
31. Wasserman, H., Blum, M.: Software reliability via run-time result-checking. J. ACM **44**(6), 826–849 (1997). Proc. IEEE FOCS 94

32. Yao, A.: A lower bound to palindrome recognition by probabilistic turing machines. In: Tech. Rep. STAN-CS-77-647 (1977)
33. Zhou, K., Afifi, M., Ren, J.: ExpSOS: secure and verifiable outsourcing of exponentiation operations for mobile cloud computing. IEEE Trans. Inf. Forens. Secur. **12**(11), 2518–2531 (2017)