Single-Server Batch Delegation of Variable-Input Pairings with Unbounded Client Lifetime

Giovanni Di Crescenzo[1], Matluba Khodjaeva[2], and Dilan D. Morales Caro[2]

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

Abstract. Pairings are important building blocks in many public-key cryptosystems. Delegation of a pairing computation $e(A, B)$ from a computationally weaker client to a computationally stronger server has been advocated to expand the applicability of pairing-based cryptosystems to computing with resource-constrained devices. In this paper we investigate the problem of delegating *a batch of pairings*. State of the art solutions are only efficient for pairings of the type $e(A_i, B)$; that is, where *only one* of the input components varies across the batch inputs. In this paper we solve the problem of efficiently delegating a batch of pairings of the type $e(A_i, B_i)$; that is, where *both* input components vary across the batch inputs. We show solutions for all of the input scenarios where the A_i, B_i are public or need to remain private, and are available in the offline phase or in the online phase of the delegation protocol. Both our protocols and those in the best previous work are among the very few in the delegation literature to enjoy unlimited client lifetime (i.e., after the offline phase, the number of delegation protocols executable by the resource-constrained client is an arbitrary polynomial in the size of the offline phase storage), which is desirable in practical applications of resource-constrained client devices. The main technical components underlying our solutions consist of new probabilistic tests simultaneously verifying multiple pairing computations, with time-efficient online verification runtime and space-efficient offline storage.

Keywords: Applied Cryptography · Secure Delegation · Pairings · Batch Pairing · Bilinear Maps · Elliptic Curves · Group Theory

1 Introduction

Server-aided cryptography (starting with, e.g., [15, 1, 30]) is an active research area addressing the problem of resource-constrained clients, such as IoT devices, delegating or outsourcing cryptographic computations to computationally more powerful servers. Batch cryptography (starting with, e.g., [16]) is another active research area addressing the problem of performing multiple cryptographic operations better than independently repeating each operation, with respect to some

performance metric. Pairings (starting with, e.g., [23, 34, 7]) are important primitive operations in many public-key cryptosystems and, more generally, cryptographic protocols (see, e.g., [32, 23, 7, 9, 21, 3, 8, 29, 28]). Currently, the first two areas are 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 third area has been attracting much cryptography research in the past 20+ years because of its wide-range capabilities and demonstrated applications.

The problem studied in this paper lies in the intersection of these three areas. Specifically, we investigate how a resource-constrained client can efficiently, privately and securely delegate to a server a batch of pairing computations. 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, or in disrupting the computation and fooling the client into accepting an incorrect computation result.

More generally, we require a solution to the batch delegation of a function F to be a 2-phase, 2-party, protocol between client C and server S, where in the offline phase some information is computed and stored on C's device, and in the online phase C and S have a brief message exchange (typically, a message from C to S followed by one from S to C; see Figure 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 $F(x_1), \ldots, F(x_n)$ with probability at least δ_c;
2. ϵ_p-*input privacy*: except for probability ϵ_p, no new information about the input batch $(x_1, \ldots, x_n)$ is revealed to S;
3. ϵ_s-*result security*: S should not be able, except possibly with probability ϵ_s, to convince C to return a result different than $(F(x_1), \ldots, F(x_n))$ at the end of the protocol; and
4. $(t_F, t_S, t_C, cc, sc, mc)$-*efficiency*:
   - *client online runtime efficiency*: C's online phase runtime, denoted as t_C, should be significantly smaller than the runtime, denoted as t_F, of computing $(F(x_1), \ldots, F(x_n))$ without delegation;
   - *unbounded client lifetime*: C's storage at the end of the end of the offline phase, denoted as sc, should be independent on the batch size n;
   - 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).

While the client online runtime efficiency property is critical for delegation to be meaningful, we also stress desirable theoretical and practical consequences of the unlimited client lifetime property: (a) the same client can be used for a number of delegations which is an arbitrary polynomial of the length of the client's storage size; and (b) no physical resetting of the storage information is needed to prevent violation of input privacy and/or result security properties.

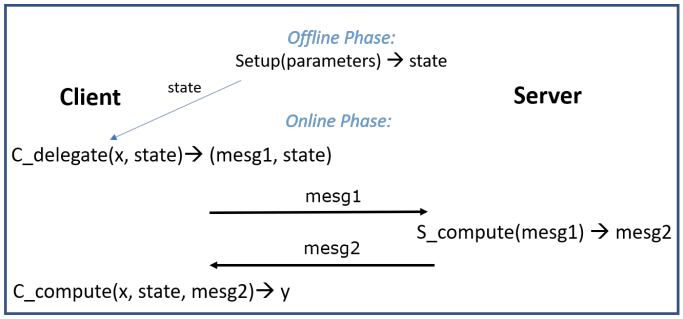

Fig. 1. Delegated computation of $y = F(x)$

Our contribution. We show single-server protocols for the delegation of a batch of n pairing computations $F(x_i)$, where function F is a pairing e, and input x_i is an arbitrary pair (A_i, B_i) of input values, for $i = 1, \ldots, n$, for all input scenarios where values A_i, B_i are public or need to remain private, and are available in the offline phase or in the online phase. All of our protocols satisfy result correctness, input privacy (unless the input is public), result security, client runtime efficiency and unbounded client lifetime, and are the first to satisfy this set of properties (for detailed comparison with the previous work showing batch pairing delegation [37, 25, 31], see discussion below for improvements on requirement satisfaction as well as Tables 3, 4, 5, 6, 7 for quantitative improvements). Our protocols achieve the unbounded client lifetime property since they use client's storage independent on n, withstand the attacks to input privacy by a set of novel and efficient input randomization techniques, and withstand the attacks to result security by a set of novel and efficient probabilistic tests. We show the client runtime efficiency property in two ways:

1. with analytical runtime expressions; here, we show that in protocols where input privacy is required (resp., is not required), the dominant term of the client's online runtime is 1 exponentiation to an exponent of length equal to the log of the target group's order + 1 exponentiation to a short exponent (resp., only 1 exponentiation to a short exponent);
2. by a software implementation of our first protocol, and derived calculation for our other protocols, using benchmark runtimes derived from our implementation, with respect to some of the most practical curve families used in recent pairing realizations; here we show that the improvement on the client's online runtime, with respect to n non-delegated pairing computations, is by a multiplicative factor between 1.09 and 4.36, depending on the curve family and protocol.

Our protocols also perform well with respect to the other targeted efficiency properties; specifically: low exchanged communication (only up to 4 group values per batch input; i.e., $cc <= 4$), only 2 messages (client delegating to server,

and server responding; i.e., $mc = 2$), and low server runtime (only 2 pairing computations per batch input by S; i.e., $t_S <= 2 \cdot t_F$).

Our results are presented through 5 novel protocols, whose input scenarios, average improvement on client online runtime over non-delegated computation (across 3 recommended elliptic curve families), and applicability to example well-known pairing-based cryptographic protocols, are captured in Table 1.

Table 1. Features of our 5 protocols: input scenario (column 2); average improvement factor (column 3); and example previous work using this input scenario and to which this delegation protocol applies (column 4).

Protocols	**Imput Scenario**	**Avg Improvement over non-delegated computation**	**Protocol applicability examples**
$\mathcal{P}_1$	A_i private online, B_i public online	1.091	[6, 19]
$\mathcal{P}_2$	A_i and B_i public online	2.007	[23, 29]
$\mathcal{P}_3$	A_i private online, B_i public offline	1.454	[3, 21]
$\mathcal{P}_4$	A_i public online, B_i public offline	3.961	[3, 7, 9, 21, 29]
$\mathcal{P}_5$	A_i and $B_i = s_iH$ private online	1.717	[8]

Comparison with Previous Work. Delegation protocols for a batch of pairings were first given in [37]. These protocols satisfy correctness, input privacy (unless the input is public), result security and unbounded client lifetime, in several input scenarios. They only work with pairing computations of the type $e(A_i, B)$, for $i = 1, \ldots, n$; that is, where *only one* of the input values varies across the batch, and only satisfy client online runtime efficiency for some (not all) of the input scenarios.

In [31] the authors show 2 delegation protocols (labeled as Protocol 3 and 4) that work for pairing computations where both input values vary across the batch. Protocol 3 satisfies correctness, result security and unbounded client lifetime, but only satisfies client online runtime efficiency for small batch sizes (of size between 5 and 13, depending on the curve family), and for all but one of the most used curve families. Protocol 4 satisfies correctness, result security and client online runtime efficiency, but does not satisfy unbounded client lifetime.

More recently, in [25], the authors showed 4 delegation protocols that work for pairing computations where both input values vary across the batch. Our analysis of these protocols showed that their protocols 1, 2 and 3 do not satisfy the result security property (see [27] for a detailed description of an attack), and protocol 4 does not satisfy client online runtime efficiency, as already mentioned by the authors.

Single-server delegation protocols for a single pairing computation have been proposed in [18, 20, 11, 26, 10, 24, 13, 14]. We note that repeating any such protocol for every input of the batch would result in protocols that satisfy correctness, input privacy (unless the input is public), result security and client online runtime efficiency, but would not satisfy the unbounded client lifetime property,

which we achieve here. One exception is our recent delegation protocol in [27] for a single pairing computation which requires no preprocessing (and thus, achieves unbounded client lifetime) in the case inputs A, B are public, but the client's online runtime is higher than the protocol in this paper for the same input scenario.

There is much other work on delegation for operations different than the pairing function studied here, for which we refer to reader to these recent surveys: [12] for other specific operations used in cryptography protocols (e.g., ring multiplication and group exponentiation), [36] for other operations beyond cryptography and [2] for computation of arbitrary functions, with clients more computationally powerful than considered here.

2 Pairing Definitions

Let $\mathbb{G}_1$, $\mathbb{G}_2$ be additive cyclic groups of order l and $\mathbb{G}_T$ be a multiplicative cyclic group of the same order l, for some large prime l. A *bilinear map* (also called *pairing* and so called from now on) is an efficiently computable map $e : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T$ with the following properties:

1. *Bilinearity:* for all $A \in \mathbb{G}_1$, $B \in \mathbb{G}_2$ and any $r, s \in \mathbb{Z}_l$, it holds that $e(rA, sB) = e(A, B)^{rs}$
2. *Non-triviality:* if U is a generator for $\mathbb{G}_1$ and V is a generator for $\mathbb{G}_2$ then $e(U, V)$ is a generator for $\mathbb{G}_T$

The last property is there to rule out the trivial case where e maps all of its inputs to 1. We denote a conventional description of the bilinear map e as $desc(e)$.

The currently most practical *pairing realizations* use an ordinary elliptic curve E defined over a field $\mathbb{F}_p$, for some large prime p, as follows. Group $\mathbb{G}_1$ is the l-order additive subgroup of $E(\mathbb{F}_p)$; group $\mathbb{G}_2$ is a specific l-order additive subgroup of $E(\mathbb{F}_{p^k})$ contained in $E(\mathbb{F}_{p^k}) \backslash E(\mathbb{F}_p)$; and group $\mathbb{G}_T$ is the l-order multiplicative subgroup of $\mathbb{F}^*_{p^k}$. Here, k is the embedding degree; i.e., the smallest positive integer such that $l|(p^k - 1)$; $\mathbb{F}_{p^k}$ is the extension field of $\mathbb{F}_p$ of degree k; and $\mathbb{F}^*_{p^k}$ is the field composed of non-zero elements of $\mathbb{F}_{p^k}$. After the Weil pairing was considered in [7], more efficient constructions have been proposed as variants of the Tate pairing, including the more recent ate pairing variants (see, e.g., [38, 35, 4] for more details on the currently most practical pairing realizations).

As in [13, 14], for our results, we further assume that $\mathbb{G}_T$ is a subgroup of a group $\mathcal{G}_T$, also contained in $\mathbb{F}^*_{p^k}$, with the following two properties:

1. testing membership in $\mathcal{G}_T$ is more efficient than testing membership in $\mathbb{G}_T$;
2. all elements of $\mathcal{G}_T$ have order $\geq l$.

This assumption is satisfied by a recently proposed security strengthening of the most practical pairing realizations. Motivated by reducing the chances of low-order attacks in cryptographic protocols, in [4] the authors proposed the notion of *subgroup-secure* elliptic curves underlying a pairing, in turn extending the notion

of $\mathbb{G}_T$-*strong* curves from [35]. As a critical step in achieving these notions, both of these papers set $\mathcal{G}_T = G_{\Phi_k(p)}$, where $G_{\Phi_k(p)}$ is the cyclotomic subgroups of order $\Phi_k(p)$ in $\mathbb{F}^*_{p^k}$, and where $\Phi_k(p)$ denotes the k-th cyclotomic polynomial. Satisfaction of above property (2) is directly implied by the definitions of both $\mathbb{G}_T$-strong and subgroup-secure curves. Satisfaction of above property (1) when $\mathcal{G}_T = G_{\Phi_k(p)}$ is detailed in Section 5.2 of [4] for the curve families BN-12, BLS-12, KSS-18, and BLS-24, in turn elaborating on Section 8.2 of [35]. There, testing membership in $\mathcal{G}_T$ is shown to only require one multiplication in $\mathbb{G}_T$ and a few lower-order Frobenius computations, a significant improvement with respect to the basic approach for testing membership in $\mathbb{G}_T$, requiring an exponentiation in $\mathbb{G}_T$ with a large exponent.

In the rest of the paper we will present protocols for the batch delegated computation of pairing e over a batch of n inputs $(A_1, B_1), \ldots, (A_n, B_n)$. One can differentiate 16 *protocol scenarios*, depending on certain features of the input components $\boldsymbol{A} = (A_1, \ldots, A_n) \in \mathbb{G}_1$ and $\boldsymbol{B} = (B_1, \ldots, B_n) \in \mathbb{G}_2$ used in the computation of pairing e: specifically, whether privacy of the inputs is desired or not, and whether the inputs are available in the offline phase or only known in the online phase. We say that $\boldsymbol{A}$ is

- *public online* if $\boldsymbol{A}$ is unknown in the offline phase but known by C and S in the online phase;
- *public offline* if $\boldsymbol{A}$ is known by both parties starting from the offline phase;
- *private online* if $\boldsymbol{A}$ is unknown in the offline phase but known by C in the online phase);
- *private offline* if $\boldsymbol{A}$ is known by C starting from the offline phase but unknown by S.

We use similar definitions for $\boldsymbol{B}$. Not all input scenarios are interesting; for instance, the ($\boldsymbol{A}$ public offline, $\boldsymbol{B}$ public offline) scenario is uninteresting since we assume that C is only resource-constrained in the online phase. Moreover, protocols for different scenarios can be related; for instance, a secure protocol for a scenario where an input is labeled as private is also a secure protocol for the otherwise identical scenario where that same input component is labeled as public. For *asymptotic efficiency* evaluation of our protocols, we will use the following definitions:

- a_i: runtime for addition in $\mathbb{G}_i$, for $i = 1, 2$;
- $m_i(\ell)$: runtime for scalar multiplication of a group value in $\mathbb{G}_i$ with an ℓ-bit scalar value, for $i = 1, 2$;
- t_M: runtime for testing membership of a value to $\mathbb{G}_T$.
- m_T: runtime for multiplication of group values in $\mathbb{G}_T$;
- $e_T(\ell)$: runtime for an exponentiation in $\mathbb{G}_T$ to an ℓ-bit exponent;
- p_T: runtime for the bilinear pairing e;
- m_l: runtime for multiplication of group values in $\mathbb{Z}_l$;
- i_l: runtime for multiplicative inversion in $\mathbb{Z}_l$.

We recall some known facts about these quantities, of interest when evaluating protocol efficiency. First, for large enough ℓ, $a_1 << m_1(\ell)$, $a_2 << m_2(\ell)$,

$m_T(\ell) << e_T(\ell)$, and $e_T(\ell) < p_T$. Also, using a double-and-add (resp., square-and-multiply) algorithm, one can realize scalar multiplication (resp., exponentiation) in additive (resp., multiplicative) groups using, for random scalars (resp., random exponents), about 1.5ℓ additions (resp., multiplications).

3 Our Batch Delegation Protocols

In this section we investigate protocols between a client and a single server for secure pairing delegation, in input scenarios where both the A_i and B_i for $i = 1, \ldots, n$ may vary across the batch. Our main results are 5 protocols with variable inputs and unbounded client lifetime, each applicable to a different input scenario, in terms of whether inputs are public or need to remain private, and whether they are known in the offline or online phase. We summarize our results with the following

Theorem 1. Let e be a pairing, let σ be its computational security parameter, and let λ be a statistical security parameter. Also, let n be an integer. There exists (constructively) single-server protocols $\mathcal{P}_j$, for $j \in \{1, \ldots, 5\}$, for the delegated computation of $e(A_i, B_i)$, for $i = 1, \ldots, n$, in input scenario is_j, satisfying δ_c-result correctness, ϵ_p-input privacy, ϵ_s-result security for values of these parameters as specified in Table 2 (top), and satisfying $(t_F, t_S, t_P, t_C, cc, sc, mc)$-efficiency, for values of these parameters as specified in Table 2 (bottom).

Table 2. (Top:) A first set of properties of our protocols: input scenario is_j, result correctness parameter δ_c, input privacy parameter ϵ_p and result security parameter ϵ_s. (Bottom): Efficiency properties of our protocols: number of messages mc, C's storage sc at the end of the offline phase (in # of group values), max term cc_{max} of communication complexity cc (in # of group values), max term $t_{C,max}$ of C's online runtime t_C, max term $t_{S,max}$ of S's runtime t_S, and max term t_P of offline runtime t_P.

Protocol	**Input scenario** is_j	δ_c	ϵ_p	ϵ_s
$\mathcal{P}_1$ [§3.1]	$\boldsymbol{A}$ private online, $\boldsymbol{B}$ public online	1	0	$2^{-\lambda}$
$\mathcal{P}_2$ [§3.2]	$\boldsymbol{A}$ public online, $\boldsymbol{B}$ public online	1	n.a.	$2^{-\lambda}$
$\mathcal{P}_3$ [§3.3]	$\boldsymbol{A}$ private online, $\boldsymbol{B}$ public offline	1	$\text{neg}(\alpha)$	$2^{-\lambda}$
$\mathcal{P}_4$ [§3.4]	$\boldsymbol{A}$ public online, $\boldsymbol{B}$ public offline	1	n.a.	$2^{-\lambda}$
$\mathcal{P}_5$ [§3.5]	$\boldsymbol{A}, \boldsymbol{B}$ priv online, $B_i = s_i H$, s_i priv online, H pub online	1	0	$2^{-\lambda}$

Protocol	mc	sc	cc_{max}	$t_{C,max}$	$t_{S,max}$	$t_{P,max}$
$\mathcal{P}_1$ [§3.1]	2	4	$n+1$ values in $\mathbb{G}_T$	$n(e_T(r) + e_T(\lambda) + m_2(r))$	$2n \cdot p_T$	p_T
$\mathcal{P}_2$ [§3.2]	2	4	$n+1$ values in $\mathbb{G}_T$	$n(e_T(\lambda) + m_2(r))$	$2n \cdot p_T$	p_T
$\mathcal{P}_3$ [§3.3]	2	3	$n+1$ values in $\mathbb{G}_T$	$n(e_T(r) + e_T(\lambda))$	$2n \cdot p_T$	$n \cdot m_2(r)$
$\mathcal{P}_4$ [§3.4]	2	3	$n+1$ values in $\mathbb{G}_T$	$n \cdot e_T(\lambda)$	$2n \cdot p_T$	$n \cdot m_2(r)$
$\mathcal{P}_5$ [§3.5]	2	3	$n+2$ values in $\mathbb{G}_T$	$n(e_T(r) + e_T(\lambda))$	$n \cdot p_T$	p_T

The main takeaway from this theorem is that C can delegate the n pairing computations with arbitrarily varying inputs to S using a protocol that satisfies

result correctness, input privacy, result security, and unbounded client lifetime, (i.e., the offline complexity of the protocol does not depend on n), and various efficiency property (most notably, client online runtime is lower than a non-delegated pairing computation). Note that this improves over the naive parallel repetition of state of the art protocols to delegate a single pairing [13, 14], which would incur offline complexity linear in n, and would not satisfy the unbounded client lifetime property. Moreover, in our protocol, client online runtime is also lower than non-delegated computation in implementations of some of the currently most recommended curve families, as detailed in Section 4.

3.1 Input scenario: $\boldsymbol{A}$ private online, $\boldsymbol{B}$ public online

Table 3. Protocols comparison in the input scenario ($\boldsymbol{A}$ private online, $\boldsymbol{B}$ public online). The expressions for t_C only include higher-order functions p_T, e_T, m_1, m_2.

Protocols	C**'s storage** (sc)	C**'s online calculation** (t_C)
[25] [§ 3.7]	$n+4$	$n(e_T(r)+e_T(\lambda)+2\,m_1(r)+m_2(\lambda))$
Ours [§ 3.1]	3	$n(e_T(r)+e_T(\lambda)+2\,m_1(r)+m_2(r)+m_1(\lambda))$

Informal Description of Protocol $\mathcal{P}_1$: A first goal in this protocol is for C to delegate the computation of $e(A_i, B_i)$, while preserving privacy of the first inputs A_i, for $i = 1, \ldots, n$. C achieves this efficiently by sending a masked version $\hat{A}_i$ of input A_i, using a random value $r_i \in \mathbb{Z}_l$, receiving $w_i = e(\hat{A}_i, B_i)$ from S, and then computing $y_i = w_i^{1/a_i}$. A second goal in the protocol is for C to check that all values $w_1, \ldots, w_n$ returned by S are correctly computed. C efficiently achieves to this by asking S to compute more pairings and by (1) a deterministic membership test that verifies that values $w_1, \ldots, w_n$ belong to $\mathcal{G}_T$; and (2) a new probabilistic test that verifies a probabilistic condition relating these extra pairing values:

$$\prod_{i=1}^{n} e(Z_i, B_i) = e(U, I) \cdot e(U, Q) \cdot \prod_{i=1}^{n} y_i^{b_i},$$

where $Z_i = b_i A_i + u_i U$, $I := (\sum_{i=1}^{n} u_i \cdot B_i) - Q$, u_i are random values from $\mathbb{Z}_l$, and U, Q are random values from $\mathbb{G}_1$. This test includes important optimizations such as using small random values $b_i \in [1, 2^\lambda]$ both as scalars in $\mathbb{G}_1$ and as exponents in $\mathbb{G}_T$, instead of full-domain values, extending a similar technique in the small-exponent test of [5], and a novel input masking technique that efficiently blends with the pairing's bilinearity property. A formal description follows.

Formal Description of Protocol $\mathcal{P}_1$:

Offline Input to C and S: $1^\sigma, 1^\lambda, desc(e)$

Offline phase instructions:

1. C randomly chooses $U \in \mathbb{G}_1$ and $Q \in \mathbb{G}_2$,
2. C sets $v := e(U, Q)$ and $ov := (U, Q, v)$

Online Input to C and S: $A_1, \ldots, A_n \in \mathbb{G}_1$ and $B_1, \ldots, B_n \in \mathbb{G}_2$

Online Input to C: ov

Online phase instructions:

1. C uniformly and independently chooses $u_1, \ldots, u_n \in \mathbb{Z}_l$
   C computes $I := (\sum_{i=1}^{n} u_i \cdot B_i) - Q$
   For $i = 1, \ldots, n$
   C randomly chooses $b_i \in [1, 2^\lambda]$ and $a_i \in \mathbb{Z}_l$
   C sets $\hat{A}_i = a_i \cdot A_i$ and $Z_i := b_i \cdot A_i + u_i \cdot U$
   C sends $I, U, (\hat{A}_1, Z_1), \ldots, (\hat{A}_n, Z_n)$ to S
2. S computes
   $w_0 := (\prod_{i=1}^{n} e(Z_i, B_i))/e(U, I)$
   $w_i := e(\hat{A}_i, B_i)$, for $i = 1, \ldots, n$
   S sends $w_0, \ldots, w_n$ to C
3. C computes $y_i := w_i^{1/a_i}$ for $i = 1, \ldots, n$
   C checks that
   (Membership Test:) $w_j \in \mathcal{G}_T$ for $j = 0, \ldots, n$
   (Probabilistic Test:) $w_0 = v \cdot \prod_{i=1}^{n} y_i^{b_i}$
   If any of these tests fails,
   C **returns** $\perp$ and the protocol halts
   C **returns** $y_1, \ldots, y_n$

Properties of protocol $\mathcal{P}_1$: *The efficiency properties* are verified by protocol inspection. In particular:

- *Storage Complexity and Unbounded Client Lifetime:* the number of bits stored on C's device at the end of the offline phase is < 4 group values, for *any* value of n. This directly implies the unbounded client lifetime property (i.e., the number n of delegation protocol executions can be an arbitrary polynomial in C's device storage size at the end of the offline phase).
- *Round complexity:* the online phase of the protocol only requires two messages: one from C to S, followed by one from S to C.
- *Communication complexity:* during the online phase, C sends $2n + 1$ values in $\mathbb{G}_1$ and 1 value in $\mathbb{G}_2$, and S sends $n + 1$ values in $\mathbb{G}_T$.
- *Runtime complexity:* the runtime property directly follows by protocol inspection. In Table 2 (bottom) we capture the max term $t_{C,max}$ of C's online runtime t_C, the max term $t_{S,max}$ of S's runtime t_S, and the max term $t_{P,max}$ of the offline runtime t_P. In Table 3 we capture a more detailed expression of C's online runtime t_C. In particular, $t_{C,max}$ consists of 2 exponentiations in $\mathbb{G}_T$ for each pairing instance, one with a full-domain exponent and one with a shorter, λ-bit, exponent.

The *correctness* property follows by showing that if C and S follow the protocol, C always outputs $y_i = e(A_i, B_i)$ for $i = 1, \ldots, n$ since

$$y_i = w_i^{1/a_i} = e(\hat{A}_i, B_i)^{1/a_i} = e(a_i \cdot A_i, B_i)^{1/a_i} = e(A_i, B_i)$$

Also, the 2 tests performed by C are always passed. The membership test is always passed by the pairing definition. Defining for brevity $\hat{B} := \sum_{i=1}^{n} u_i \cdot B_i$, note that the probabilistic test is always passed since

$$\begin{aligned} w_0 &= \frac{1}{e(U, I)} \prod_{i=1}^{n} e(Z_i, B_i) = \frac{1}{e(U, \hat{B} - Q)} \prod_{i=1}^{n} e(b_i \cdot A_i + u_i \cdot U, B_i) \\ &= \frac{e(U, Q)}{e(U, \hat{B})} \prod_{i=1}^{n} e(A_i, B_i)^{b_i} \prod_{i=1}^{n} e(u_i\, U, B_i) = \frac{e(U, Q)}{e(U, \hat{B})} \prod_{i=1}^{n} y_i^{b_i} \cdot \prod_{i=1}^{n} e(U, u_i \cdot B_i) \\ &= \frac{v}{e(U, \hat{B}))} \prod_{i=1}^{n} y_i^{b_i} \cdot e\left(U, \sum_{i=1}^{n} u_i \cdot B_i\right) = v \prod_{i=1}^{n} y_i^{b_i}. \end{aligned}$$

This implies that C never returns $\perp$, and thus always returns $y_i = e(A_i, B_i)$ for $i = 1, \ldots, n$.

The *privacy* property of the protocol against any malicious S follows by observing that, C's message $(I, (\hat{A}_1, Z_1), \ldots, (\hat{A}_n, Z_n))$ to S does not leak any information about C's inputs $A_1, \ldots, A_n$. This follows because for all $i = 1, \ldots, n$ (a) values $\hat{A}_i = a_i \cdot A$ is uniformly and independently distributed in $\mathbb{G}_1$ as a_i is a random scalar in $\mathbb{Z}_l$ and $\mathbb{G}_1$ is cyclic and (b) values Z_i are uniformly and independently distributed in $\mathbb{G}_1$, as so are U_i by similar reasoning. Moreover, by similar reasoning, this message does not leak any information about b_i, a fact useful in the proof of the security property.

To prove the *security* property against any malicious S, we need to compute an upper bound ϵ_s on the security probability that S convinces C to output a y_i such that $y_i \neq e(A_i, B_i)$ for $i = 1, \ldots, n$. We obtain that $\epsilon_s \leq 2^{-\lambda}$, as a consequence of the following 3 facts, which we later prove:

1. tuple $((\hat{A}_1, Z_1), \ldots, (\hat{A}_n, Z_n))$ leaks no information about $(b_1, \ldots, b_n)$ to S;
2. for any S's message $w = (w_0, \ldots, w_n)$ such that at least one of $w_1, \ldots, w_n$ is different than what would be returned according to the protocol instructions, the number of tuples $(b_1, \ldots, b_n)$ for which w satisfies both the membership and the probabilistic test in step 4 of the protocol is $\leq 2^{\lambda(n-1)}$;
3. for any S's message $w = (w_0, \ldots, w_n)$ such that at least one of $w_1, \ldots, w_n$ is different than what would be returned according to the protocol instructions, the probability that w satisfies the probabilistic test is $\leq 2^{-\lambda}$.

Towards proving Fact 1, we observe that: (a) all $\hat{A}_i$ values are uniformly and independently distributed in $\mathbb{G}_1$ since so are the a_i values in $\mathbb{Z}_l$; (b) conditioned on the $(\hat{A}_1, \ldots, \hat{A}_n)$ tuple, all $Z_1, \ldots, Z_n$ values are also uniformly distributed in $\mathbb{G}_1$ since so are the $u_1 U, \ldots, u_n U$ values, when u_i values are uniformly and

independently chosen from $\mathbb{Z}_l$. Thus, the distribution of $((\hat{A}_1, Z_1), \ldots, (\hat{A}_n, Z_n))$ is independent from that of $b_1, \ldots, b_n$, from which Fact 1 follows.

Towards proving Fact 2, let $w = (w_0, \ldots, w_n)$ be the tuple that would be returned by S according to the protocol, and assume that a malicious algorithm Adv, corrupting S, returns a different tuple $w' = (w'_0 \ldots, w'_n)$. Note that if $w'_i \notin \mathcal{G}_T$ for some $j \in \{0, \ldots, n\}$, the tuple w' does not satisfy the membership test. Thus, we assume that $w'_j \in \mathcal{G}_T$ for all $j \in \{0, \ldots, n\}$. Because $\mathcal{G}_T$ is a multiplicative group, we can write

$$w'_j = d_j \cdot w_j,$$

for some $d_j \in \mathcal{G}_T$ such that not all of the d_j, for $j = 0, \ldots, n$, are equal to 1.

Now, assume that one of the d_i is different than 1 for $i = 1, \ldots, n$; specifically, assume wlog that $d_n \neq 1$, and consider the following equivalent rewritings of the probabilistic test, obtained by variable substitutions and simplifications:

$$\begin{aligned} w'_0 = v \cdot \prod_{i=1}^{n} {y'_i}^{b_i} &= v \cdot \prod_{i=1}^{n} (w'_i)^{b_i/a_i} \\ d_0 \cdot w_0 &= v \cdot \prod_{i=1}^{n} (d_i \cdot w_i)^{b_i/a_i} \\ d_0 &= \prod_{i=1}^{n} (d_i)^{b_i/a_i}, \end{aligned}$$

where the last equality follows from the protocol's correctness property. Now, if for any $b_1, \ldots, b_{n-1}$ there exist two distinct b_n and b'_n, assuming without loss of generality that $b_n > b'_n$, such that

$$d_0 = \left(\prod_{i=1}^{n-1} (d_i)^{b_i/a_i} \right) \cdot (d_n)^{b_n/a_n} \text{ and } d_0 = \left(\prod_{i=1}^{n-1} (d_i)^{b_i/a_i} \right) \cdot (d_n)^{b'_n/a_n},$$

then $d_n^{(b_n - b'_n)/a_n} = (d_n^{1/a_n})^{(b_n - b'_n)} = 1$. By our assumption that every element in $\mathcal{G}_T$ has order $> l$, which is $> 2^\lambda$, and by observing that $b_n - b'_n < 2^\lambda$, we derive that $(d_n)^{1/a_n}$ cannot have order $\leq b_n - b'_n$. Thus the equality $(d_n^{1/a_n})^{(b_n - b'_n)} = 1$ can only hold when $b_n = b'_n$. This shows that for any $b_1, \ldots, b_{n-1}$, there is at most one b_n for which both the membership and probabilistic verification tests are met, and thus are at most $2^{\lambda(n-1)}$ tuples $(b_1, \ldots, b_n)$ for which the tuple w satisfies both tests. This proves Fact 2.

Towards proving Fact 3, first recall Fact 1 saying that C's message $((\hat{A}_1, Z_1), \ldots, (\hat{A}_n, Z_n))$ does not leak any information about tuple $(b_1, \ldots, b_n)$, and thus all values in $\{1, \ldots, 2^\lambda\}^n$ are still equally likely for such tuple even when conditioning over C's message. Then we can compute an upper bound on the probability that S's message w satisfies the probabilistic test, as the ratio between the number of $(b_1, \ldots, b_n)$ tuples for which w satisfies both tests, and the number of values of $(b_1, \ldots, b_n)$ that are still equally likely even when conditioning over C's message. In this ratio, the numerator is at most $2^{\lambda(n-1)}$ (by Fact 2), and the denominator is $= 2^{\lambda n}$ (by Fact 1). This proves Fact 3.

3.2 Input scenario: $\boldsymbol{A}$ and $\boldsymbol{B}$ public online

In the input scenario ($\boldsymbol{A}$ and $\boldsymbol{B}$ public online), we can define protocol $\mathcal{P}_2$, by slight modifications to protocol $\mathcal{P}_1$; i.e., by setting $a_i := 1$, for all $i = 1, \ldots, n$. Thus, during the online phase in Step 1, C does not compute $\hat{A}_i$ as $a_i \cdot A_i$ and does not send $\hat{A}_i$ to S, for $i = 1, \ldots, n$. Also, in Step 4, C outputs $y_i = w_i$ (instead of $y_i = w_i^{1/a_i}$) which eliminates n group exponentiations in $\mathbb{G}_T$ in C's online calculation.

We observe that, similarly as proved for $\mathcal{P}_1$, protocol $\mathcal{P}_2$ satisfies the result correctness and security properties. The efficiency properties for $\mathcal{P}_2$ are similar as for $\mathcal{P}_1$, with the only differences being in the improved bound for $t_C \leq n \cdot (a_1 + a_2 + m_1(r) + m_1(\lambda) + m_2(r) + m_T + e_T(\lambda) + t_M) + t_M$. In Table 4 we show that C's storage is constant, thus implying the unbounded client lifetime property, and online runtime t_C is significantly improved with respect to the only previous protocol (Algorithm 3 of [31]) satisfying this property.

Table 4. Protocols comparison in the input scenario ($\boldsymbol{A}$ and $\boldsymbol{B}$ public online). The expressions of t_C only includes higher-order functions p_T, e_T, m_1, m_2.

Protocols	C**'s storage** (sc)	C**'s online calculation** t_C
[14] for each pairing [§3]	$6n$	$n(e_T(\lambda) + m_2(\lambda) + m_1(r))$
[31][Alg.4]	n	$p_T + n(e_T(\lambda) + m_2(\lambda) + m_1(r))$
[31][Alg.3]	0	$n^2 e_T(2\lambda) + n(m_2(\lambda) + m_1(\lambda))$
Our $\mathcal{P}_2$ in [§ 3.2]	4	$n(e_T(\lambda) + m_2(r) + m_1(r) + m_1(\lambda))$

3.3 Input scenario: $\boldsymbol{A}$ private online and $\boldsymbol{B}$ public offline

In this input scenario, we can define protocol $\mathcal{P}_3$, by starting with protocol $\mathcal{P}_1$, and applying the following modifications to improve C's efficiency. Specifically, during the offline phase: (a), in Step 1, C does not need to use random value $Q \in \mathbb{G}_2$; (b) in Step 3, C calculates I and v as $I := \sum_{i=1}^{n} u_i \cdot B_i$, $v := e(U, I)$, and C sets ov as $ov := (U, I, s, v)$. During the online phase: (a) in Step 1, C does not need to compute I, as it was computed in the offline phase; this significantly reduces C's online runtime as it eliminates n scalar multiplications in $\mathbb{G}_2$; (b) in Step 1, C does not need to send values U and I to S; (c) in Step 3, S calculates w_0 as $w_0 := \prod_{i=1}^{n} e(Z_i, B_i)$ which eliminates calculation of $e(U, I)$. The resulting probabilistic test can be written as:

$$\prod_{i=1}^{n} e(Z_i, B_i) = e(U, I) \cdot \prod_{i=1}^{n} y_i^{b_i},$$

where $Z_i = b_i A_i + u_i U$, $I := (\sum_{i=1}^{n} u_i \cdot B_i)$, u_i are random values from $\mathbb{Z}_l$, b_i are small random values from $[1, 2^\lambda]$, U is a random value from $\mathbb{G}_1$, and

$y_i = e(\hat{A}_i, B_i)^{1/a_i} = e(a_i A_i, B_i)^{1/a_i} = e(A_i, B_i)$. This test is more efficient than in $\mathcal{P}_1$ as it uses a simplified input masking technique.

Formal Description of Protocol $\mathcal{P}_3$: Let $m = 2n\lceil \log(l+1) \rceil$, and let $G : \{0,1\}^\alpha \rightarrow \{0,1\}^m$ denote a pseudo-random generator, for some integer α. We note that G can be implemented using AES in, say, Counter mode, whose runtime is much more efficient than a pairing computation. A formal description follows, mostly focusing on differences between $\mathcal{P}_3$ and $\mathcal{P}_1$.

Offline Input to C and S: 1^α, $desc(e), desc(G)$

Offline Input to C: $B_1, \ldots, B_n$

Offline phase instructions:

1. C randomly chooses $U \in \mathbb{G}_1$ and seed $s \in \{0,1\}^\alpha$,
2. C computes $p = G(s)$, applies a conventional encoding to write p as a sequence of $2n$ values in $\mathbb{Z}_l^*$, and sets $u_1, \ldots, u_n$ as the first n such values that belong to $\mathbb{Z}_l^*$
3. C computes $I := (\sum_{i=1}^n u_i \cdot B_i)$ and $v = e(U, I)$
4. C sets $ov := (U, I, s, v)$

Input to C and S: $B_1, \ldots, B_n \in \mathbb{G}_2$

Online Input to C: $A_1, \ldots, A_n \in \mathbb{G}_1$ and ov

Online phase instructions:

1. C computes $u_1, \ldots, u_n$ from s exactly as in step 2 of the offline phase
2. C computes $(\hat{A}_1, Z_1), \ldots, (\hat{A}_n, Z_n)$ as in $\mathcal{P}_1$ and sends them to S
3. S sets $w_0 = \prod_{i=1}^n e(Z_i, B_i)$, computes $w_1, \ldots, w_n$ as in $\mathcal{P}_1$. and sends w_0, $w_1, \ldots, w_n$ to C
4. C checks $w_0, \ldots, w_n$ and returns $y_1, \ldots, y_n$ (or $\perp$), exactly as done in $\mathcal{P}_1$.

Properties of protocol $\mathcal{P}_3$: Similarly as proved for $\mathcal{P}_1$, we can show that protocol $\mathcal{P}_3$ satisfies result correctness, input privacy and result security.

When compared with our protocol $\mathcal{P}_1$ for a more demanding input scenario, $\mathcal{P}_3$ has improved efficiency properties, the main differences being in reduced values for $t_C \leq n \cdot (a_1 + i_l + 2\, m_1(r) + m_1(\lambda) + m_T + e_T(r) + e_T(\lambda) + t_M) + t_M$, $t_S \leq 2n \cdot p_T + n \cdot m_T + i_l$ and $cc = 2n$ values in $\mathbb{G}_1 + (n+1)$ values in $\mathbb{G}_T$.

When compared with the most basic solution for this input scenario (i.e., direct repetition for each instance in the batch of the most efficient protocol [13] to delegate a single pairing computation), it also improves storage sc and offline phase runtime t_P at the cost of some low-order increase in the client online runtime t_C, which still remains lower than non-delegated computation. More detailed expressions for sc and t_C are captured in Table 5.

3.4 Input scenario: *A* public online, *B* public offline

In this input scenario we can define protocol $\mathcal{P}_4$, by slight modifications to protocol $\mathcal{P}_3$; i.e., by setting $a_i := 1$, for $i = 1, \ldots, n$. As a result, during the online

Table 5. Protocols comparison in the input scenario ($\boldsymbol{A}$ private online and $\boldsymbol{B}$ public offline). The expression of t_C only includes higher-order functions p_T, e_T, m_1, m_2.

Protocols	C**'s storage** (sc)	C**'s online calculation** (t_C)
[13]§ 3.2 for each input	$4n$	$n(e_T(\lambda) + m_1(\lambda))$
Our $\mathcal{P}_3$ in [§ 3.3]	3	$n(e_T(r) + e_T(\lambda) + 2\,m_1(r) + m_1(\lambda))$

phase in Step 1, C does not compute $\hat{A}_i$ as $a_i \cdot A_i$ and C does not send $\hat{A}_i$ to S, for $i = 1, \ldots, n$. This eliminates n scalar multiplications in group $\mathbb{G}_1$ from C's online runtime. Also, in Step 3, C outputs $y_i := w_i$, which eliminates n exponentiations in group $\mathbb{G}_T$ from C's online runtime.

We observe that, similarly as proved for $\mathcal{P}_3$, protocol $\mathcal{P}_4$ satisfies the result correctness and result security properties.

When compared with our protocol $\mathcal{P}_3$ for a more demanding input scenario, protocol $\mathcal{P}_4$ has improved efficiency properties, the main differences being in the reduced values for $t_C \leq n{\cdot}(a_1{+}a_2{+}m_1(r){+}m_1(\lambda){+}m_2(r){+}m_T{+}e_T(\lambda){+}t_M){+}t_M$, and $cc = n$ values in $\mathbb{G}_1$ and $+(n+1)$ values in $\mathbb{G}_T$.

When compared with the most basic solution for this input scenario (i.e., direct repetition for each instance in the batch of the most efficient protocol [13] to delegate a single pairing computation), $\mathcal{P}_4$ also improves storage sc and offline phase runtime t_P at the cost of some low-order increase in the client online runtime t_C, which still remains lower than non-delegated computation. More detauled expressions for sc and t_C are captured in Table 6.

Table 6. Protocols comparison in the input scenario ($\boldsymbol{A}$ public online and $\boldsymbol{B}$ public offline). The expression of t_C only includes higher-order functions e_T, m_1, m_2.

Protocols	C**'s storage** (sc)	C**'s online calculation** t_C
[13]§3.1 for each pairing	$2n$	$n(e_T(\lambda) + m_1(\lambda))$
Our $\mathcal{P}_4$ in [§ 3.4]	3	$n(e_T(\lambda) + m_1(r) + m_1(\lambda))$

3.5 Input scenario: $\boldsymbol{A}$ and $\boldsymbol{B} = \boldsymbol{s} \cdot \boldsymbol{H}$ private online

In this section we investigate client-server protocols for the delegated computation of pairings with more general input scenarios, where both the A_i and the B_i may vary across the batch. Specifically, we consider pairing computations of the form $e(A_i, B_i)$, for $i = 1, \ldots, n$, where the batch inputs can have arbitrarily different first component values $\boldsymbol{A} = (A_1, \ldots, A_n)$ as well as different second component values $\boldsymbol{B} = (B_1, \ldots, B_n)$, where each $B_i = s_i H$, where H is a generator of group $\mathbb{G}_2$, and $\boldsymbol{s} = (s_1, \ldots, s_n)$ is a tuple of scalars from $\mathbb{Z}_l$. Specifically, we consider the input scenario ($\boldsymbol{A}$ private online, $\boldsymbol{s}$ private online, H public online). Although this scenario is somewhat less general than the ($\boldsymbol{A}$ private online, $\boldsymbol{B}$ private online) scenario, we note that input scenarios of this

type are not uncommon in constructions of cryptographic protocols from the literature; for instance, they are used in [8] and many of its follow-up papers.

Protocol $\mathcal{P}_5$ description: We note that the approach where C delegates $e(A_i, H)$ using [37], and then exponentiates the result to s_i results in a protocol, called $\mathcal{P}_{5,0}$, where the client online runtime is not smaller than non-delegated computation (see Table 8 (bottom)). Instead, in our protocol $\mathcal{P}_5$, we avoid this exponentiation by processing s_i via a much more efficient multiplication in $\mathbb{Z}_l$. Specifically, C delegates to S n pairing computations $e(\hat{A}_i, H)$, for $i = 1, \ldots, n$, where H is the publicly known generator of group $\mathbb{G}_2$, $\hat{A}_i$ is obtained from input A_i by a multiplication with a scalar that is obtained by multiplying a random value in $\mathbb{Z}_l$ times the value s_i such that $B_i = s_i H$. Then, building on [37], $\hat{A}_0$ is set as a random linear combination of $\hat{A}_1, \ldots, \hat{A}_n$ added to a random value U where S calculates $w_0 = e(\hat{A}_0, H)/e(U, I)$ and sends $w_0, \ldots, w_n$ to C. Later, C uses S's answers $e(\hat{A}_i, H)$ to compute the desired pairing outputs $e(A_i, B_i)$ and all answers, including $e(\hat{A}_0, H)$, to probabilistically test the correctness of all answers via a random exponential equation over $\mathcal{G}_T$. As before, we use efficient membership tests in $\mathbb{G}_T$ and smaller exponents (i.e., λ-bit exponents instead of full-domain exponents) in this equation to significantly improve client runtime. The probabilistic test used here verifies the condition:

$$e(\hat{A}_0, H)/e(U, I) = e(U, Q) \cdot \prod_{i=1}^{n} w_i^{b_i},$$

where $w_i = e(\hat{A}_i, H)$, $\hat{A}_i = (a_i s_i \mod l) A_i$, $\hat{A}_0 := U + \sum_{i=1}^{n} b_i \cdot \hat{A}_i$, b_i are random values from $[1, 2^\lambda]$, U is a random values from $\mathbb{G}_1$, Q is a random value from $\mathbb{G}_2$, and $I = H - Q$. This test includes the masking and optimizations ideas in tests in our previous protocols, and additionally uses a single random mask to preserve privacy of both A_i and s_i values, so to effectively replace one extra exponentiation in $\mathbb{G}_T$ with a much more efficient multiplication in $\mathbb{Z}_l$. A formal description follows.

Input scenario: $A_1, \ldots, A_n$ private online, $B_1, \ldots, B_n$ private online, such that $B_i = s_i H$, for some random $s_i \in \mathbb{Z}_l$ and some public online generator H of $\mathbb{G}_2$

Offline Input to C and S: $1^\sigma, 1^\lambda$, $desc(e)$

Offline phase instructions:

1. C randomly chooses $U \in \mathbb{G}_1$ and $Q \in \mathbb{G}_2$,
2. C sets $v := e(U, Q)$ and $ov := (U, Q, v)$

Online Input to C: $A_1, \ldots, A_n \in \mathbb{G}_1$, $s_1, \ldots, s_n \in \mathbb{Z}_l$ and $B_1, \ldots, B_n \in \mathbb{G}_2$ such that $B_i = s_i \cdot H$ for $i = 1, \ldots, n$ and ov.

Online Input to C and S: generator $H \in \mathbb{G}_2$

Online phase instructions:

1. For $i = 1, \ldots, n$,
   C randomly chooses $b_i \in [1, 2^\lambda]$ and $a_i \in \mathbb{Z}_l^*$

C computes $\hat{A}_i := (a_i \cdot s_i \mod l) \cdot A_i$
C sets $I := H - Q$ and $\hat{A}_0 := U + \sum_{i=1}^{n} b_i \cdot \hat{A}_i$
C sends $I, \hat{A}_0, \ldots, \hat{A}_n$ to S

2. S computes $w_0 = e(\hat{A}_0, H)/e(U, I)$, and $w_i := e(\hat{A}_i, H)$ for $i = 0, \ldots, n$
   S sends $w_0, \ldots, w_n$ to C
3. C checks that
   (Membership Test:) $w_i \in \mathcal{G}_T$ for $i \in [0, n]$
   (Probabilistic Test:) $w_0 = v \cdot \prod_{i=1}^{n} w_i^{b_i}$
   If any of these tests fails, C **returns** $\perp$ and the protocol halts
   C **returns** $y_1 := w_1^{1/a_1}, \ldots, y_n := w_n^{1/a_n}$

Table 7. Protocols comparison in the input scenario with private online $A_1, \ldots, A_n$ and $B_1, \ldots, B_n$ where $B_i = s_i H$ and both random scalar s_i and H are known in online phase for $i = 1, \ldots, n$. (The expression of t_C only includes higher-order functions e_T, m_1, m_2.)

Protocols	C**'s storage** (sc)	C**'s online calc.** t_C
[14]§4 for each pairing	$15n$	$n(e_T(\lambda) + m_2(r) + m_2(\lambda) + 3\,m_1(r))$
[25][Alg.4]	$n + 3$	$n(e_T(r) + e_T(\lambda) + m_2(r) + m_2(\lambda) + 2m_1(r))$
Our $\mathcal{P}_5$ in [§ 3.5]	3	$n(e_T(r) + e_T(\lambda) + m_1(r) + m_1(\lambda))$

Properties of protocol $\mathcal{P}_5$: The analysis of the *efficiency*, *privacy* and *security* properties are very similar to those of protocol $\mathcal{P}_1$.

The proof of the *correctness* property is somewhat different, when we have to show that the probabilistic test is always passed, and that when both tests are passed, C always outputs $y_i = e(A_i, B_i)$ for $i = 1, \ldots n$. To see that the probabilistic test is always passed, note that

$$w_0 = \frac{e(\hat{A}_0, H)}{e(U, I)} = \frac{1}{e(U, H - Q)} \cdot e\left(\left(U + \sum_{i=1}^{n} b_i \cdot \hat{A}_i\right), H\right)$$
$$= \frac{e(U, H)}{e(U, H)e(U, Q)^{-1}} \cdot \prod_{i=1}^{n} e(\hat{A}_i, H)^{b_i} = v \cdot \prod_{i=1}^{n} w_i^{b_i}$$

The membership test is always passed by the pairing definition. When both tests are passed, C never returns $\perp$, and thus returns

$$y_i = w_i^{1/a_i} = e(\hat{A}_i, H)^{1/a_i \mod l} = e((a_i s_i \mod l) \cdot A_i, H)^{1/a_i \mod l}$$
$$= e(A_i, H)^{a_i \cdot s_i \cdot (1/a_i) \mod l} = e(A_i, H)^{s_i} = e(A_i, s_i \cdot H) = e(A_i, B_i)$$

4 Practical Performance Analysis of our Protocols

We show a numerical performance analysis of our protocols $\mathcal{P}_1, \ldots, \mathcal{P}_5$ as well as previous protocols from the literature in each of the considered input scenarios,

which confirms our parameterized efficiency results. Our numerical performance analysis consists of obtaining benchmark results, captured in Table 8, for the runtime of an optimal ate pairing and of other operations (i.e., addition and scalar multiplication in groups $\mathbb{G}_1$ and $\mathbb{G}_2$, and multiplication, membership testing and exponentiation in $\mathbb{G}_T$) using some of the currently most practical elliptic curve families (i.e., KSS 18-P638 and BLS 24-P509 for security level 192 and BLS-P315 for security level 128). In Table 9 we compare the performance of our protocols in Section 3 with past work for the same input scenario. We evaluated protocol $\mathcal{P}_1$ in two different ways: (1) measuring online client runtime of a software implementation of the protocol, and (2) analytically deriving the performance using the benchmark in Table 8. Also we measured the marginal error of (1) vs. (2), which is observed to be arguably small (specifically, between 2.85% and 20.87%). We then analytically derived the performance of protocols $\mathcal{P}_2, \ldots, \mathcal{P}_5$ using the benchmarks in Table 8.

Denoting a pairing as $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$, and by p_T its runtime, by e_T the runtime of exponentiation in $\mathbb{G}_T$ and by m_i the runtime of scalar multiplication in $\mathbb{G}_i$ and a_i the runtime of addition in $\mathbb{G}_i$, for $i = 1, 2$. our estimation procedure for delegation protocols $\mathcal{P}_1, \ldots, \mathcal{P}_5$ for pairing e, worked as follows:

1. Set statistical parameter λ as $= 40$ (an event happening with probability $\leq 2^{-40}$ has occurrence expectancy of 1 in $\geq 2^{20}$ trillion protocol executions, thus capturing confidence requirements in all use cases we can think of).
2. Set security parameter σ based on best available attacks.
3. Consider $\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T$ and pairing e for which order l satisfies $l >= 2^{2\sigma}$, and measure runtimes for $p_T, e_T, m_1, m_2, m_T, a_1, a_2, t_M$ and generate benchmark values as in Table 8
4. Use above benchmark values to estimate t_C in the given delegation protocol, neglecting lower-order operations such as equality testing, assignments, Frobenius-based simplifications, etc.
5. return: p_T/t_C.

Our software implementation was carried out on a macOS Big Sur Version 11.7.6 laptop with a 2.7 GHz processor Dual-Core Intel core i5 and 8 GB DDR3 RAM memory. We used C code for protocol $\mathcal{P}_1$ and the benchmark calculations, using the relic-toolkit library [33] and the gmp library.

5 Conclusions

We show the first protocols for the delegation of a batch of pairing computations $e(A_i, B_i)$ with variable inputs A_i, B_i, satisfying a number of desirable privacy, security and performance properties; most notably: result correctness, input privacy, result security, unbounded client lifetime, and client runtime efficiency. Our results settle problems unsolved since the original 2007 paper of [37]. These protocols can be used to reduce runtime of resource-constrained devices in multiple runs of essentially all known pairing-based cryptographic schemes and protocols,

Table 8. Benchmark results for scalar multiplications in $\mathbb{G}_1, \mathbb{G}_2$ and exponentiations in $\mathbb{G}_T$ relative to an optimal ate pairing based on some of the currently most practical curve families. All columns measured in millions of clock cycles. Standard deviation lies in the following ranges: (1) BLS24-P509 in a range (0,0.0077); (2) KSS18-P638 in a range (0,0.0099); (3) BLS24-P315 in a range (0,0.04).

Curves	**KSS18-P638**	**BLS24-P509**	**BLS24-P315**
Operation \Security	**192**	**192**	**128**
a_1	7.00E-06	3.00E-06	2.00E-06
$m_1(r)$	6.53E-03	3.63E-03	2.09E-03
$m_1(\lambda)$	1.65E-03	1.06E-03	6.93E-04
a_2	7.00E-06	3.00E-06	2.00E-06
$m_2(r)$	1.27E-02	8.91E-03	6.22E-03
$m_2(\lambda)$	2.86E-03	2.96E-03	2.74E-03
m_T	3.80E-05	7.20E-05	7.40E-05
$e_T(r)$	1.25E-02	1.49E-02	8.76E-03
$e_T(\lambda)$	2.67E-03	3.41E-03	4.56E-03
t_M	2.27E-04	5.09E-04	4.32E-04
p_T	4.04E-02	3.78E-02	3.05E-02

Table 9. Ratio p_T/t_C of non-delegated pairing computation to client runtime in the online phase, with respect to protocols $\mathcal{P}_1, \ldots, \mathcal{P}_5$ with comparison to other protocols, where the statistical security parameter $\lambda = 40$, 100 time repetitions

Protocols	**Storage**	**Ratio:** p_T/t_C		
		KSS18-P638 $r = 474$	**BLS24-P509** $r = 409$	**BLS24-P315** $r = 253$
Input Scenario:(A_i private online, B_i public online)				
[25] §3.7	$n+4$	1.277	1.274	1.437
Our $\mathcal{P}_1$ in § 3.1 runtime measurement	4	1.193	1.086	1.191
Our $\mathcal{P}_1$ in § 3.1 using benchmarks in Table 8	4	0.944	1.047	1.225
Benchmark vs measuring error	NA	20.87%	3.59%	2.85%
Input Scenario:(A_i and B_i public online)				
[14] §3 for each pairing	$6n$	3.210	3.387	2.935
Our $\mathcal{P}_2$ in § 3.2	4	1.700	2.151	2.169
Input Scenario:(A_i private online, B_i public offline)				
[13]§3.2 for each pairing	$4n$	8.724	7.385	5.228
Our $\mathcal{P}_3$ in § 3.3	3	1.339	1.390	1.633
Input Scenario:(A_i public online, B_i public offline)				
[13]§3.1 for each pairing	$2n$	8.810	7.495	5.297
Our $\mathcal{P}_4$ in § 3.4	3	3.636	4.358	3.889
Input Scenario:(A_i and B_i private online); In our $\mathcal{P}_5$: $B_i = s_i \cdot H$				
[14]§4 for each pairing	$15n$	1.047	1.351	1.428
[25] Alg. 4	$n+3$	0.912	0.980	1.111
Our $\mathcal{P}_{5,0}$ in § 3.5	2	0.794	0720	0.986
Our $\mathcal{P}_5$ in § 3.5	3	1.709	1.604	1.838

such as, for instance, those for key agreement [23], advanced public key encryption types [8, 3, 29, 19], identity-based encryption [7], hierarchical identity-based encryption [6], and signature schemes [9, 21, 29].

Acknowledgements. Many thanks to Delaram Kahrobaei, Vladimir Shpilrain, and Damien Vergnaud for interesting discussions. Work by Matluba Khodjaeva and Dilan Morales Caro was supported by a Faculty Scholarship grant from the Office for the Advancement of Research at John Jay College and PSC CUNY Cycles 53 and 54. Work by Giovanni Di Crescenzo was supported by the Defense Advanced Research Projects Agency (DARPA), contract n. HR001120C0156. 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. Distribution Statement "A" (Approved for Public Release, Distribution Unlimited).

References

1. M. Abadi, J. Feigenbaum, J. Kilian, *On Hiding Information from an Oracle* In: J. Comput. Syst. Sci. 39(1): 21-50 (1989)
2. H. Ahmad, L. Wang, H. Hong, J. Li, H. Dawoo, M. Ahmed, Y. Yang, *Primitives towards verifiable computation: a survey.* In Front. Comput. Sci. 2018, Vol. 12, Issue (3) : 451-478.
3. S.S. Al-Riyami, K.G. Paterson, *Certificateless Public Key Cryptography.* In: Laih C.S. (eds) Advances in Cryptology - ASIACRYPT 2003.
4. P.S.L.M. Barreto, C. Costello, R. Misoczki, M. Naehrig, G.C.C.F. Pereira, G. Zanon, *Subgroup security in pairing-based cryptography.* In: Lauter K., Rodríguez-Henríquez F. (eds) Progress in Cryptology – LATINCRYPT 2015.
5. M. Bellare, J. Garay, and T. Rabin, *Fast batch verification for modular exponentiation and digital signatures.*, In Proc. of Eurocrypt 98, pp. 236-250, Springer, 1998
6. D. Boneh, X. Boyen, and E. Goh, *Hierarchical Identity Based Encryption with Constant Size Ciphertext.* In: Proc. of Eurocrypt 2005. LNCS vol. 3494, Springer.
7. D. Boneh, M. Franklin, *Identity-based Encryption from the Weil Pairing.* In: Proc. of CRYPTO 2001. LNCS vol. 2139, Springer.
8. D. Boneh, G. Di Crescenzo, R. Ostrovsky, G. Persiano, *Public Key Encryption with Keyword Search.* In: Proc. of EUROCRYPT 2004. LNCS vol. 3027, Springer.
9. D. Boneh, B. Lynn, H. Shacham, *Short Signatures from the Weil Pairing.* In: Boyd C. (eds) Advances in Cryptology — ASIACRYPT 2001. LNCS vol 2248. Springer.
10. S. Canard, J. Devigne, O. Sanders, *Delegating a pairing can be both secure and efficient.* In Proc. of ACNS 2014. LNCS vol 8479. Springer.
11. B. Chevallier-Mames, J.S. Coron, N. McCullagh, D. Naccache, M. Scott, *Secure delegation of elliptic-curve pairing. Cryptology ePrint Archive.* In: Proc. of CARDIS 2010. LNCS vol 6035. Springer. Also IACR EPrint 2005/150.
12. G. Di Crescenzo, M. Khodjaeva, D. Kahrobaei, and V. Shpilrain: *A Survey on Delegated Computation.* In Proc. of DLT 2022, LNCS, Springer: pp. 33-53.

13. G. Di Crescenzo, M. Khodjaeva, D. Kahrobaei, V. Shpilrain, *Secure and Efficient Delegation of Elliptic-Curve Pairing.* In: Proc. of ACNS 2020. LNCS, vol 12146. Springer, Cham.
14. G. Di Crescenzo, M. Khodjaeva, D. Kahrobaei, V. Shpilrain, *Secure and Efficient Delegation of Pairings with Online Inputs.* In: Proc. of CARDIS 2020. LNCS, vol. 12609. Springer.
15. J. Feigenbaum, *Encrypting Problem Instances: Or ..., Can You Take Advantage of Someone Without Having to Trust Him?*, in Proc. of CRYPTO 1985: 477-488
16. A. Fiat, *Batch RSA.* In Journal of Cryptology, v. 10, n. 2, 1997.
17. R. Gennaro, C. Gentry, B. Parno, *Non-interactive verifiable computing: Outsourcing computation to untrusted workers.* In: Proc. of CRYPTO 2010, LNCS 6223, pp. 465–482.
18. M. Girault, D. Lefranc, *Server-aided verification: Theory and practice.* In: Roy, B.K. (ed.) ASIACRYPT 2005. LNCS, vol. 3788, pp. 605–623.
19. V. Goyal, O. Pandey, A. Sahai, B. Waters, *Attribute-based encryption for fine-grained access control of encrypted data.* In: Proc. of ACM CCS 2006, ACM, pp. 89–98.
20. Guillevic, A., Vergnaud, D., *Algorithms for outsourcing pairing computation.* In: Proc. of CARDIS 2014, LNCS vol. 8968. Springer.
21. F. Hess, *Efficient Identity Based Signature Schemes Based on Pairings.* In: Nyberg K., Heys H. (eds) Selected Areas in Cryptography. SAC 2002.
22. S. Hohenberger, A. Lysyanskaya, *How to securely outsource cryptographic computations.* In: Proc. of TCC 2005, pp. 264–282, Springer.
23. A. Joux, *A One Round Protocol for Tripartite Diffie-Hellman.* In: Proc. of ANTS 2000, pp. 385-394.
24. E.J. Kachisa, E.F. Schaefer, M. Scott, *Constructing Brezing-Weng pairing friendly elliptic curves using elements in the cyclotomic field.* In: Galbraith S.D., Paterson K.G. (eds) Pairing-Based Cryptography – Pairing 2008. LNCS vol. 5209. Springer.
25. O. Kalkar, I. Sertkaya, and S. Tutdere, *On The Batch Outsourcing Of Pairing Computations*, in: The Computer Journal, 2022.
26. B.G. Kang, M.S. Lee, J.H. Park, *Efficient delegation of pairing computation.* In: IACR Cryptology ePrint Archive, n. 259, 2005.
27. M. Khodjaeva and G. Di Crescenzo, *On Single-Server Delegation Without Precomputation.* In: Proc. of SECRYPT 2023, pp. 540-547, Scitepress.
28. S. Kumar, Y. Hu, M. P. Andersen, R. A. Popa, D. E. Culler, *JEDI: Many-to-Many End-to-End Encryption and Key Delegation for IoT.* In: USENIX Security Symposium 2019: 1519-1536.
29. J.K. Liu, M.H. Au, W. Susilo, *Self-generated-certificate public-key cryptography and certificateless signature/encryption scheme in the standard model.* In: Proc. ACM Symp. on Information, Computer and Communications Security. ACM Press (2007).
30. T. Matsumoto, K. Kato, H. Imai, *An improved algorithm for secure outsourcing of modular exponentiations.* In: Proc. of CRYPTO 1988, pp. 497–506, LNCS, Springer.
31. T. Mefenza, and D. Vergnaud, *Verifiable outsourcing of pairing computations*, Technical report, 2018.
32. D. Moody, R. Peralta, R. Perlner, A. Regenscheid, A. Roginsky, and L. Chen, *Report on Pairing-based Cryptography*, in J. Res. Natl. Inst. Stand. Technol. 120: 11–27, 2015
33. D. F. Aranha, C. P. L. Gouvêa, T. Markmann, R. S. Wahby, K. Liao, *RELIC is an Efficient LIbrary for Cryptography*, https://github.com/relic-toolkit/relic
34. R. Sakai, K. Ohgishi, and M. Kasahara, *Cryptosystems based on pairing*, in Symposium on Cryptography and Information Security (SCIS), 2000

35. M. Scott, *Unbalancing pairing-based key exchange protocols.* In: IACR Cryptology ePrint Archive, n. 688, 2013.
36. Z. Shan, K. Ren, M. Blanton, C. Wang, *Practical Secure Computation Outsourcing: A Survey.* In ACM Comput. Surv. 51(2): 31:1-31:40, 2018.
37. P. Tsang, S. Chow, and S. Smith *Batch pairing delegation.* In: Proceedings of International Workshop on Security, 2007, pp.74–90. Springer, Berlin.
38. F. Vercauteren, *Optimal Pairings.* In: IEEE Transactions on Information Theory, vol. 56, no. 1, pp. 455–461, Jan. 2010.

A Formal Definitions

In this section we recall the formal definition (based on [17, 22]), of delegation protocols for a batch of function values, for arbitrary functions, as well as their correctness, privacy, security, and efficiency requirements.

Basic notations. By $\boldsymbol{x}$ we denote the n-tuple $(x_1, \ldots, x_n)$. 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. As in previous work in the area, we consider a delegation protocol to include both an *offline phase*, typically consisting of client setup and deployment, and an *online phase*, typically starting when inputs to the delegated function are available to the client. While the protocol in the online phase is designed by taking into account the parties' different computation power, in particular reducing as much as possible the client's computation, this is not the case in the offline phase, which could be run, for instance, by a trusted deployer which at the end stores information on the client device. A system architecture can be found in Figure 1.

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 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: $\{pp \leftarrow \text{Offline}(1^\sigma, 1^\lambda, desc(F)); (\boldsymbol{y}, tr) \leftarrow (C(pp, \boldsymbol{x}), S)\}$. Step 1 is executed in the offline phase, when no input to the function F is available yet, and could be executed either by C or a third party not colluding with S. Step 2 is executed in the online phase, when inputs $\boldsymbol{x} = (x_1, \ldots, x_n)$ to the function F are available to C, and at the end of

this phase, C learns $\boldsymbol{y} = (y_1, \ldots, y_n)$, where $y_i = F(x_i)$, for $i = 1, \ldots, n$, and tr is the transcript of the communication exchanged between C and S. We will often omit $desc(F), 1^\sigma, 1^\lambda, tr$ for brevity.

Correctness Requirement. Informally, 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 n-tuple of values obtained by evaluating function F on C's n inputs. Formally, we say that a client-server protocol (C, S) for the delegated n-instance computation of F satisfies δ_c-*correctness* if for any $\boldsymbol{x} = (x_1, \ldots, x_n)$ in $Dom(F)$,

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

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

1. $pp \leftarrow \text{Offline}(desc(F))$
2. $(\boldsymbol{y}, tr) \leftarrow (C(pp, \boldsymbol{x}), S)$
3. if $y_i = F(x_i)$ for all $i = 1, \ldots, n$, 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 inputs $\boldsymbol{x}$ 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 client-server protocol for the delegated n-instance 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

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

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

1. $pp \leftarrow \text{Offline}(desc(F))$
2. $(\boldsymbol{x}_0, \boldsymbol{x}_1, aux) \leftarrow A(desc(F))$
3. $b \leftarrow \{0, 1\}$
4. $(\boldsymbol{y}, tr) \leftarrow (C(pp, \boldsymbol{x}_b), A(aux))$
5. $d \leftarrow A(tr, aux)$
6. 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 $\boldsymbol{x}$, at the end of the protocol, C cannot be convinced to obtain some output tuple $\boldsymbol{z}$ containing a value $z_i \neq F(x_i)$ for some $i \in \{1, \ldots, n\}$. 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,

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

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

1. $pp \leftarrow \text{Offline}(desc(F))$
2. $(\boldsymbol{x}, aux) \leftarrow A(desc(F))$
3. $(\boldsymbol{z}, tr) \leftarrow (C(pp, \boldsymbol{x}), A(aux))$
4. if $z_i \in \{\perp, F(x_i)\}$ for all $i = 1, \ldots, n$, then **return:** 0 else **return:** 1.