# Secure and Efficient Delegation of Elliptic-Curve Pairing

Giovanni Di Crescenzo[1], Matluba Khodjaeva[2],
Delaram Kahrobaei[3], and Vladimir Shpilrain[4]

[1] Perspecta Labs Inc. Basking Ridge, NJ, USA.
Email: gdicrescenzo@perspectalabs.com
[2] CUNY John Jay College of Criminal Justice. New York, NY, USA.
Email: mkhodjaeva@jjay.cuny.edu
[3] University of York. Heslington, York, UK.
Email: delaram.kahrobaei@york.ac.uk
[4] City University of New York. New York, NY, USA.
Email: shpil@groups.sci.ccny.cuny.edu

**Abstract.** Many public-key cryptosystems and, more generally, cryptographic protocols, use pairings as important primitive operations. To expand the applicability of these solutions to computationally weaker devices, it has been advocated that a computationally weaker client delegates such primitive operations to a computationally stronger server. Important requirements for such delegation protocols include privacy of the client's pairing inputs and security of the client's output, in the sense of detecting, except for very small probability, any malicious server's attempt to convince the client of an incorrect pairing result.
In this paper we show that the computation of bilinear pairings in all known pairing-based cryptographic protocols can be efficiently, privately and securely delegated to a single, possibly malicious, server. Our techniques provides efficiency improvements over past work in all input scenarios, regardless on whether inputs are available to the parties in an offline phase or only in the online phase, and on whether they are public or have privacy requirements. The client's online runtime improvement is, for some of our protocols almost 1 order of magnitude, no matter which practical elliptic curve, among recently recommended ones, is used for the pairing realization.

**Keywords:** Secure Delegation · Pairings · Cryptography · Elliptic Curves

## 1 Introduction

Server-aided cryptography is an active research direction addressing the problem of computationally weaker clients delegating the most expensive cryptographic computations to computationally powerful servers. Recently, this area is seeing an increased interest because of shifts in modern computation paradigms towards cloud/fog/edge computing, large-scale computations over big data, and computations with low-power devices, such as RFIDs and smart-grid readers.

The first formal model for delegation of cryptographic operations was introduced in [27], where the authors especially studied delegation of group exponentiation, as this operation is a cornerstone of so many cryptographic schemes and protocols. In this model, we have a client, with an input $x$, who delegates to one or more servers the computation of a function $F$ on the client's input, and the main desired requirements are:

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

As in all previous work in the area, protocols can be partitioned into (a) an offline phase, where input $x$ is not yet known, but somewhat expensive computation can be performed by the client or a client deployer and stored on the client's device, and (b) an online phase, where we assume the client runtime is limited, and thus help by the server is needed to compute $F(x)$.

**Our Contributions.** In this paper we show that bilinear pairings can be efficiently, privately and securely delegated to a single, possibly malicious, server. We consider different meaningful protocol scenarios, depending on whether each of the two inputs $A$ and $B$ to the pairing is labeled as offline (i.e., available to the client already in the offline phase) or online (i.e., only available to the client in the online phase), and depending on whether each of the two inputs to the pairing is public (i.e., known to both client and server) or private (i.e., only known to the client and needs to remain private from the server). Our results improve previous work across all input scenarios (thus being applicable to all pairing-based cryptographic protocols in the literature) and are presented through 5 main novel protocols, whose input scenarios, improvement over previous best protocol and over non-delegated computation, and relevance to example well-known pairing-based cryptographic protocols, are captured in Table 1. Our efficiency improvements over previous schemes, in some cases almost reaching 1 order of magnitude, are measured with respect to all of the 4 recently proposed practical elliptic curves with security levels between 128 and 256 bits, as benchmarked in [10]. In our first two protocols, the client's most expensive operation is an exponentiation to a short (i.e., 128-bit) exponent. No such protocol had been previously offered in the literature. Moreover, in all of our protocols, the client only performs 1 or 2 exponentiations to a short exponent in the pairing target group, as opposed to full-domain exponentiations in past work. Our largest efficiency improvements are in the most practically relevant scenarios where at least one of the two pairing inputs is known in the offline phase. When both pairing inputs are known in the online phase, for some elliptic curves we show the first protocol improving over non-delegated computation.

**Related Work.** Delegating computation has been and continues to be an active research area, with the increased importance of new computation paradigms,

**Table 1.** For each of our 5 main protocols, we list the input scenario (column 2); the average, across 4 often recommended elliptic curves, multiplicative improvement factor on the online client's runtime over previous best protocol (column 3) and over non-delegated pairing computation (column 4); and some example previous work using this input scenario and to which this delegation protocol applies (column 5).

| | **Input Scenario** | **Avg Improvement** | | **Protocol applicability examples** |
|---|---|---|---|---|
| | | **over previous best** | **over non-delegated computation** | |
| **1.** | A public online, B public offline | 5.55 | 9.41 | [1,7,9,26,31] |
| **2.** | A private online, B public offline | 4.20 | 9.36 | [1,26] |
| **3.** | A private online, B private offline | 4.36 | 5.34 | [1,7,8,31] |
| **4.** | A public online, B public online | 2.79 | 4.16 | [28,31] |
| **5.** | A private online, B private online | 1.75 | 1.31 | |

such as computing with low-power devices, cloud/fog/edge computing, etc. In its early years, a number of solutions had been proposed and then attacked in follow-up papers. The first formal model for secure delegation protocols was presented in [27]. There, a secure delegation protocol is formally defined essentially as a secure function evaluation (in the sense of the concept first proposed in [37]) of the client's function delegated to the server. Follow-up models from [22] and [12,18] define separate requirements of correctness, (input) privacy and (result) security. There, privacy is defined as indistinguishability of two different inputs from the client, even after corrupting the server; and security is defined as the adversary's inability to convince the client of an incorrect function output, even after corrupting the server. We can partition all other (single-server) secure delegation protocols we are aware of in 4 main classes, depending on whether they delegate (a) elliptic curve pairings; (b) group exponentiation [18,19,27,13,16,20,32]; (c) other specific computations (e.g., linear algebra operations) [33,3,6,4,21]; and (d) an arbitrary polynomial-size circuit evaluation [22,17,24].

With respect to (a), pairing delegation was first studied in a work by Girault et al. [23]. However, they only considered computation secrecy but no security against a malicious server. Guillevic et al. [25] proposed a more efficient scheme but their method increases communication complexity between client and server and their scheme does not provide security against a malicious server. Protocols with this latter property for delegating $e(A, B)$ have first been provided by Chevallier-Mames et al. [14,15] and later by Kang et al. [30], but the drawback of the protocol in [14] is that it is more costly for the client than a non-delegated computation. Canard et al. [11] improved their construction and proposed more efficient and secure pairing delegation protocols. In particular, in [11] the authors showed that in their protocols the client's runtime is strictly lower than non-delegated computation of a pairing on the so-called KSS-18 curve [29]. Later, Guillevic et al. [25] showed that in protocols in [11] the client is actually less efficient than in a non-delegated computation of the pairing for the state of the art optimal ate pairing on a Barreto-Naehrig curve.

## 2 Notations and Definitions

In this section we recall known definition and facts about pairings (in Section 2.1) and known definitions for delegation protocols, including their correctness, security, privacy and efficiency requirements (in Section 2.2).

### 2.1 Pairings

*Bilinear Maps.* Let $\mathcal{G}_1$, $\mathcal{G}_2$ be additive cyclic groups of order $l$ and $\mathcal{G}_T$ 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 $e : \mathcal{G}_1 \times \mathcal{G}_2 \rightarrow \mathcal{G}_T$ with the following properties:

1. *Bilinearity:* for all $A \in \mathcal{G}_1$, $B \in \mathcal{G}_2$ 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 $\mathcal{G}_1$ and $V$ is a generator for $\mathcal{G}_2$ then $e(U, V)$ is a generator for $\mathcal{G}_T$

The last property rules out the trivial scenario 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 $\mathcal{G}_1$ is the $l$-order additive subgroup of $E(\mathbb{F}_p)$; group $\mathcal{G}_2$ is a specific $l$-order additive subgroup of $E(\mathbb{F}_{p^k})$ contained in $E(\mathbb{F}_{p^k}) \setminus E(\mathbb{F}_p)$; and group $\mathcal{G}_T$ 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 $l|(p^k - 1)$. 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., [36] for more details on the currently most practical pairing realizations).

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

- $a_1$ (resp. $a_2$) denotes the runtime for addition in $\mathcal{G}_1$ (resp. $\mathcal{G}_2$);
- $m_1(\ell)$ (resp. $m_2(\ell)$) denotes the runtime for scalar multiplication of a group value in $\mathcal{G}_1$ (resp. $\mathcal{G}_2$) with an $\ell$-bit scalar value;
- $m_T$ denotes the runtime for multiplication of group values in $\mathcal{G}_T$;
- $e_T(\ell)$ denotes the runtime for an exponentiation in $\mathcal{G}_T$ to an $\ell$-bit exponent;
- $p_T$ denotes the runtime for the bilinear pairing $e$;
- $t_M$ denotes the runtime for testing membership of a value to $\mathcal{G}_T$.

We recall some well-known facts about these quantities, of interest when evaluating the efficiency of our protocols. First, for large enough $\ell$, $a_1 << m_1(\ell)$, $a_2 << m_2(\ell)$, $m_T(\ell) << e_T(\ell)$, 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 $1.5\ell$ additions (resp., multiplications). Finally, membership of a value $w$ in $\mathcal{G}_T$ can be tested using one exponentiation in $\mathcal{G}_T$ to the $l$-th power (i.e., checking that $w^l = 1$), or, for some specific elliptic curves, including some of the most recommended in practice, using about 1 multiplication in $\mathcal{G}_T$ and lower-order Frobenius-based simplifications (see, e.g., [34, 5]).

For *concrete efficiency* evaluation of our protocols, we will use benchmark results from [10] for the runtime of an optimal ate pairing and of the other most expensive operations (i.e., scalar multiplication in groups $\mathcal{G}_1$, $\mathcal{G}_2$ and exponentiation in $\mathcal{G}_T$) for the best curve families, also recalled in Table 2 below. We will also neglect lower-order operations such as equality testing, assignments, Frobenius-based simplifications, etc.

**Table 2.** Benchmark results (obtained by [10] on an Intel Core i7-3520M CPU averaged over thousands of random instances) for scalar multiplications in $\mathcal{G}_1, \mathcal{G}_2$ and exponentiations in $\mathcal{G}_T$ relative to an optimal ate pairing based on some of the best known curve families, measured in millions (M) of clock cycles.

| **Sec. level** | **Family-$k$** | **Pairing $e$** | **Scal. mul. in $\mathcal{G}_1$** | **Scal. mul. in $\mathcal{G}_2$** | **Exp. in $\mathcal{G}_T$** |
|---|---|---|---|---|---|
| **128-bits** | BN-12 | 7.0 | 0.9 | 1.8 | 3.1 |
| **192-bits** | BLS-12 | 47.2 | 4.4 | 10.9 | 17.5 |
| | KSS-18 | 63.3 | 3.5 | 9.8 | 15.7 |
| **256-bits** | BLS-24 | 115.0 | 5.2 | 27.6 | 47.1 |

## 2.2 Delegation Protocols: Definitions

Our protocol modeling builds on previous papers, including [22, 27, 12, 18].

**Basic notations.** The expression $y \leftarrow T$ denotes the probabilistic process of randomly and independently choosing $y$ from set $T$. The expression $y \leftarrow A(x_1, x_2, \ldots)$ denotes the (possibly probabilistic) process of running algorithm $A$ on input $x_1, x_2, \ldots$ and any necessary random coins, and obtaining $y$ as output. The expression $(y_A, y_B) \leftarrow (A(x_{A,1}, x_{A,2}, \ldots), B(x_{B,1}, x_{B,2}, \ldots))$ denotes the (possibly probabilistic) process of running an interactive protocol between $A$, taking as input $x_{A,1}, x_{A,2}, \ldots$ and any necessary random coins, and $B$, taking as input $x_{B,1}, x_{B,2}, \ldots$ and any necessary random coins, where $y_A, y_B$ are $A$ and $B$'s final outputs, respectively, at the end of this protocol's execution.

**System scenario, entities, and protocol.** We consider a system with two types of parties: clients and servers, where a client's computational resources are expected to be more limited than a server's ones, and therefore clients are interested in delegating the computation of specific functions to servers. In all our solutions, we consider a single *client*, denoted as $C$, and a single *server*, denoted as $S$. We assume that the communication link between $C$ and $S$ is private or not subject to confidentiality, integrity, or replay attacks, and note that such attacks can be separately addressed using known applied cryptography techniques. As in all previous work in the area, our time model includes an *offline phase*, which may coincide with a client device setup and/or data pre-deployment, and a later *online phase*, which may coincide with client operations, possibly use the data gathered in the offline phase, and where the client is resource-constrained. For simplicity of description, we will consider a generic algorithm run by the client in the offline phase, but note that other parties may also run it (e.g., a client-deploying server, the client itself, etc.) and assume that this decision is better settled at implementation time.

Let $\sigma$ denote the computational security parameter (i.e., the parameter derived from hardness studies of the underlying computational problem), and let $\lambda$ denote the statistical security parameter (i.e., a parameter such that events with probability $2^{-\lambda}$ are extremely rare). Both parameters are expressed in unary notation (i.e., $1^\sigma, 1^\lambda$). When performing numerical performance analysis, we use $\lambda = 128$ and a value of $\sigma$ that depends on the elliptic curve and security that we want to use. In particular, [10] reports such values for some of today's most practical curves, including BN-12 curves (embedding degree $k = 12$, security level 128-bits) for which $\sigma = 461$, BLS-12 curves ($k = 12$, security level 192-bits) for which $\sigma = 635$, KSS-18 curves ($k = 18$, security level 192-bits) for which $\sigma = 508$, and BLS-24 curves ($k = 24$, security level 256-bits) for which $\sigma = 629$.

Assuming $desc(e)$ is a description of pairing $e : \mathcal{G}_1 \times \mathcal{G}_2 \rightarrow \mathcal{G}_T$ known to both $C$ and $S$, we define a *client-server protocol for the delegated computation of* $e$ as a 2-party, 2-phase, communication protocol between $C$ and $S$, denoted as $(C(1^\sigma, 1^\lambda, desc(e), x_C), S(1^\sigma, 1^\lambda, desc(e), x_S))$, where $x_C = (x_{C,f}, x_{C,n})$, and consisting of the following steps:

1. $pp \leftarrow \text{Offline}(1^\sigma, 1^\lambda, desc(e), x_{C,f})$,
2. $(y_C, y_S) \leftarrow (C(1^\sigma, 1^\lambda, desc(e), pp, x_{C,n}), S(1^\sigma, 1^\lambda, desc(e), x_S)$.

One can differentiate 16 *protocol scenarios*, depending on certain features of the inputs $A \in \mathcal{G}_1$ and $B \in \mathcal{G}_2$ to pairing $e$. We say that the first input $A$ is

- *public online* if $x_{C,n}$ and $x_S$ include $A$ but $x_{C,f}$ does not (i.e., $A$ is unknown in the offline phase but known by both parties in the online phase);
- *public offline* if $x_{C,n}, x_{C,f}$ and $x_S$ include $A$ (i.e., $A$ is known by both parties starting from the offline phase);
- *private online* if $x_{C,n}$ include $A$ but $x_{C,f}, x_S$ do not (i.e., $A$ is unknown in the offline phase but known by $C$ in the online phase);
- *private offline* if $x_{C,n}$ and $x_{C,f}$ include $A$ but $x_S$ does not (i.e., $A$ is known by $C$ starting from the offline phase but unknown by $S$).

We use similar definitions for the second input $B$. As an example, the scenario denoted as '($A$ public offline, $B$ public offline)' is the protocol scenario where both inputs $A$ and $B$ are known to both parties $C$ and $S$ starting from the offline phase. (This scenario is the least interesting since we assume that $C$ is only resource-constrained in the online phase.) While there could be 15 additional distinct scenarios, we now make some observations that reduce the number of most interesting scenarios for the purpose of our study: (1) as the definition of pairing is symmetric across the two inputs, half of the protocol scenarios are of less interest, as a secure protocol for one scenario is also secure for the symmetric scenario; (2) 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 is labeled as public; (3) a secure protocol for a scenario where an input is labeled as online is also a secure protocol for the otherwise identical scenario where that same input is labeled as offline. Note that, despite these 3 facts, it is still of interest to analyze a less demanding scenario if a more efficient protocol can be found for it. Moreover, by reviewing usages of pairings in cryptography papers, we noted that a large majority of scenarios involve at least one of the two inputs being known offline, which we study in greater detail in Section 3.

Let $\sigma, \lambda$ be the security parameters, and let $(C, S)$ be a client-server protocol for the delegated computation of a pairing $e$, and fix a protocol scenario.

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

**Definition 1.** We say that $(C, S)$ satisfies $\delta_c$-*correctness* if for any $(A, B)$ in $e$'s domain, it holds that

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

for some $\delta_c$ close to 1, where experiment CorrExp is detailed below:

```
CorrExp_e$(1^\sigma, 1^\lambda)$
1. $pp \leftarrow \text{Offline}(desc(e), x_{C,f})$
2. $(y_C, y_S) \leftarrow (C(pp, x_{C,n}), S(x_S))$
3. if $y_C = e(A, B)$ then **return:** 1 else **return:** 0
```

**Security Requirement.** Informally, the most basic security requirement would state the following: if $C$ follows the protocol, a malicious adversary corrupting $S$ cannot convince $C$ to obtain, at the end of the protocol, some output $y'$ different from the value $y$ obtained by evaluating pairing $e$ on $C$'s input $(A, B)$. To define a stronger and more realistic security requirement, we augment the adversary's power so that the adversary can even choose inputs to $C$ and $S$, including $A \in \mathcal{G}_1$ and $B \in \mathcal{G}_2$, before attempting to convince $C$ of an incorrect output. We also do not restrict the adversary to run in polynomial time. A formal definition follows.

**Definition 2.** We say that $(C, S)$ satisfies $\epsilon_s$-*security against a malicious adversary* if for any algorithm $Adv$ returning inputs for $C$ and $S$ for the fixed protocol scenario, it holds that

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

for some $\epsilon_s$ close to 0, where experiment SecExp is detailed below:

```
SecExp_e,Adv$(1^\sigma, 1^\lambda)$
1. $(x_{C,f}, x_{C,n}, x_S, aux) \leftarrow Adv(desc(e))$
2. $pp \leftarrow \text{Offline}(desc(e), x_{C,f})$
3. $(y', aux) \leftarrow (C(pp, x_{C,n}), Adv(aux))$
4. if $y' = \perp$ or $y' = e(A, B)$, for $A \in \mathcal{G}_1, B \in \mathcal{G}_2$ then **return:** 0 else **return:** 1.
```

**Privacy Requirement.** Informally, the privacy requirement should guarantee the following: if $C$ follows the protocol, a malicious adversary corrupting $S$ cannot obtain any information about $C$'s input $(A, B)$ from a protocol execution. This is formalized by extending the indistinguishability-based approach typically used in formal definitions for encryption schemes. That is, the adversary can pick two inputs $(x_{C,f,b}, x_{C,n,b}, x_{S,b})$, for $b = 0, 1$; then, one of these two inputs is chosen at random and used by $C$ in the protocol with the adversary acting as $S$, and finally the adversary tries to guess which input was used by $C$. Note that depending on the protocol scenario, the adversary is trying to learn about only one of the two pairing inputs or both (or even none, in which case this requirement becomes vacuous). As for security, we do not restrict the adversary to run in polynomial time. A formal definition follows.

**Definition 3.** We say that $(C, S)$ satisfies $\epsilon_p$*-privacy (in the sense of indistinguishability) against a malicious adversary* if for any algorithm $Adv$ returning inputs for the fixed protocol scenario, it holds that

$$\left| \text{Prob}\left[\, out \leftarrow \text{PrivExp}_{\text{e,Adv}}(1^\sigma, 1^\lambda) \,:\, out = 1 \right] \,-\, 1/2 \,\right| \,\le\, \epsilon_p,$$

for some $\epsilon_p$ close to 0, where experiment PrivExp is detailed below:

```
PrivExp_{e,Adv}$(1^\sigma, 1^\lambda)$
1. $((x_{C,f,0}, x_{C,n,0}, x_{S,0}), (x_{C,f,1}, x_{C,n,1}, x_{S,1}), aux) \leftarrow Adv(desc(e))$
2. $b \leftarrow \{0, 1\}$
3. $pp \leftarrow \text{Offline}(desc(e), x_{C,f,b})$
4. $(y', d) \leftarrow (C(pp, x_{C,n,b}), Adv(aux))$
5. if $b = d$ then return: 1 else return: 0.
```

**Efficiency Metrics and Requirements.** Let $(C, S)$ be a client-server protocol for the delegated computation of pairing $e$. We say that $(C, S)$ has *efficiency parameters* $(t_F, t_P, t_C, t_S, cc, mc)$, if $e$ can be computed (without delegation) using $t_F(\sigma, \lambda)$ atomic operations, $C$ can be run in the offline phase using $t_P(\sigma, \lambda)$ atomic operations and in the online phase using $t_C(\sigma, \lambda)$ atomic operations, $S$ can be run using $t_S(\sigma, \lambda)$ atomic operations, $C$ and $S$ exchange a total of at most $mc$ messages, of total length at most $cc$. While we naturally try to minimize all these protocol efficiency metrics, our main goal is to design protocols where

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

In all our protocols $t_S \le 5t_F$, so we actually devote most of our attention on asymptotic analysis of $t_C$ and target a concrete performance ratio $t_C/t_F < 1$, which we achieve for all protocol scenarios and all 4 practical curves for which pairing benchmark runtimes are reported in [10].

## 3 Delegating Pairings with one Offline Input

In this section we investigate client-server protocols for secure pairing delegation, in various scenarios where one of the pairing inputs is already known to the client in the offline phase. Our main results are 3 new protocols, each applicable to a different scenario. For each of these, we give a formal statement of our result, an asymptotic and a concrete efficiency comparison with the previous best protocols in the same scenario, an informal description of the ideas behind the protocol, a formal description of the protocol and a proof of the protocol's correctness, privacy and security properties.

### 3.1 Protocol scenario: ($A$ public online, $B$ public offline)

Our first protocol satisfies the following

**Theorem 1.** Let $e$ be a pairing, as defined in Section 2.1, let $\sigma$ be its computational security parameter, and let $\lambda$ be a statistical security parameter. There exists (constructively) a client-server protocol $(C, S)$ for delegating the computation of $e$, when input $A$ is publicly known in the online phase, and input $B$ is publicly known in the offline phase, which satisfies 1-correctness, $2^{-\lambda}$-security, 0-privacy, and efficiency with parameters $(t_F, t_S, t_P, t_C, cc, mc)$, where

- $t_F = p_T$, $t_S = 2\,p_T$ and $t_P = p_T$;
- $t_C \leq a_1 + m_1(\lambda) + m_T + e_T(\lambda) + t_M$;
- $cc = 1$ value in $\mathcal{G}_2 + 2$ values in $\mathcal{G}_T$ and $mc = 2$.

The main takeaway from this theorem is that $C$ can securely and efficiently delegate to $S$ the computation of a bilinear pairing whose first input $A$ is publicly known in the online phase and second input $B$ is already publicly known in the offline phase. In particular, in the online phase $C$ only performs one exponentiation to a $\lambda$-bit exponent in $\mathcal{G}_T$, and 1 multiplication to a $\lambda$-bit scalar in $\mathcal{G}_1$, as well as other lower-order operations. (See Table 3 for a concrete comparison with best previous work, also showing estimated ratios of $C$'s online runtime $t_C$ and the runtime $t_F$ of a non-delegated pairing calculation ranging between 0.077 and 0.160 depending on the curve used.) Additionally, $C$ only computes 1 pairing in the offline phase, $S$ only computes 2 pairings, and $C$ and $S$ only exchange 2 messages containing a small number of group values.

**Table 3.** Protocol comparison in the scenario ($A$ public online, $B$ public offline)

| **Protocols** | $t_C$ | **Ratio:** $t_C/t_F$ | | | |
|---|---|---|---|---|---|
| | | **BN-12** | **BLS-12** | **KSS-18** | **BLS-24** |
| | | $\sigma = 461$ | $\sigma = 635$ | $\sigma = 508$ | $\sigma = 629$ |
| [14, 15] [Sec 6.1] | $a_2 + m_2(\sigma) + m_T + e_T(\sigma) + t_M$ | 0.702 | 0.603 | 0.404 | 0.651 |
| Ours [Sec 3.1] | $a_1 + m_1(\lambda) + m_T + e_T(\lambda) + t_M$ | 0.160 | 0.094 | 0.077 | 0.093 |

**Protocol Description.** The main idea in this protocol is that since both inputs $A$ and $B$ are publicly known, $S$ can compute $w_0 = e(A, B)$ and send $w_0$ to $C$, along with some efficiently verifiable 'proof' that $w_0$ was correctly computed. This proof is realized by the following 3 steps: first, $C$ sends to $S$ a randomized version $Z_1$ of value $A$, then $S$ computes and sends to $C$ pairing value $w_1 = e(Z_1, B)$; and finally $C$ verifies that $w_1 \in \mathcal{G}_T$ and uses $w_1$ and a pairing value computed in the offline phase in an efficient probabilistic verification for the correctness of $w_0$. A formal description follows.

*Offline Input to C:* $B \in \mathcal{G}_2$

*Offline phase instructions:*

1. $C$ randomly chooses $U_1 \in \mathcal{G}_1$
2. $C$ sets $v_1 := e(U_1, B)$

*Online Input to C and S:* $1^\sigma, 1^\lambda$, $desc(e)$, $A \in \mathcal{G}_1$, $B \in \mathcal{G}_2$

*Online phase instructions:*

1. $C$ randomly chooses $b \in \{1, \ldots, 2^\lambda\}$
   $C$ sets $Z_1 := b \cdot A + U_1$
   $C$ sends $Z_1$ to $S$
2. $S$ computes $w_0 := e(A, B)$ and $w_1 := e(Z_1, B)$
   $S$ sends $w_0, w_1$ to $C$
3. (Membership Test:) $C$ checks that $w_0 \in \mathcal{G}_T$
   (Probabilistic Test:) $C$ checks that $w_1 = w_0^b \cdot v_1$
   If any of these tests fails then $C$ **returns** $\perp$ and the protocol halts
   $C$ **returns** $y = w_0$

**Protocol Properties:** The *efficiency* properties are verified by protocol inspection. In particular, we note that $C$'s calculation of $Z_1$ only requires 1 multiplication in $\mathcal{G}_1$ to a short, $\lambda$-bit, scalar, $C$'s membership test only requires 1 multiplication in $\mathcal{G}_T$, as discussed in Section 2.2, and $C$'s probabilistic test only requires 1 multiplication and 1 exponentiation in $\mathcal{G}_T$ to a short, $\lambda$-bit, exponent.

The *correctness* property follows by showing that if $C$ and $S$ follow the protocol, $C$ always outputs $y = e(A, B)$. We first show that the 2 tests performed by $C$ are always passed. The membership test is always passed by the pairing definition; the probabilistic test is always passed since

$$w_1 = e(Z_1, B) = e(b \cdot A + U_1, B) = e(A, B)^b \cdot e(U_1, B) = w_0^b \cdot v_1.$$

This implies that $C$ never returns $\perp$, and thus always returns $y = w_0 = e(A, B)$.

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

1. $Z_1$ leaks no information about $b$ to $S$;
2. for any $S$'s message $(w_0, w_1)$ different than what would be returned according to the protocol instructions, there is only one $b$ for which $(w_0, w_1)$ satisfy both the membership and the probabilistic test in step 3 of the protocol;
3. for any $S$'s message $(w_0, w_1)$ different than what would be returned according to the protocol instructions, the probability that $(w_0, w_1)$ satisfies the probabilistic test is $\leq 2^{-\lambda}$.

Towards proving Fact 1, we observe that $Z_1$ is uniformly distributed in $\mathcal{G}_1$ since so is $U_1$, which is unknown to $S$. Thus, the distribution of $Z_1$ is independent from that of $b$, from which Fact 1 follows.

Towards proving Fact 2, let $(w_0, w_1)$ be the values that would be returned by $S$ according to the protocol, and assume a malicious algorithm $Adv$ corrupting $S$ returns a different pair $(w_0', w_1')$. Because $\mathcal{G}_T$ is cyclic, we can consider a generator $g$ for $\mathcal{G}_T$ and write $w_i = g^{a_i}$, for $i = 1, 2$. Note that if the membership and probabilistic test, both values in $(w_0', w_1')$ are verified to be in $\mathcal{G}_T$. Then we can write

$$w_0' = g^u \cdot w_0 \text{ and } w_1' = g^v \cdot w_1 \text{ for some } u, v \in \mathbb{Z}_l \text{ such that } u \neq 0 \text{ or } v \neq 0.$$

Now, assume wlog that $u \neq 0 \mod l$ and consider the following equivalent rewritings of the probabilistic test, obtained by variable substitutions and simplifications:

$$\begin{aligned} w_1' &= (w_0')^b \cdot v_1 \\ g^v \cdot w_1 &= (g^u \cdot e(A, B))^b \cdot e(A, U_1) \\ g^v \cdot e(A, Z_1) &= g^{ub} \cdot e(A, B)^b \cdot e(A, U_1) \\ g^v &= g^{ub} \\ v &= ub \mod l. \end{aligned}$$

Now, if there exist two distinct $b_1$ and $b_2$ such that

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

then $u(b_1 - b_2) = 0 \mod l$ then $b_1 - b_2 = 0 \mod l$ (i.e $b_1 = b_2$) because $u \neq 0 \mod l$. This shows if $u \neq 0 \mod l$ then that $b$ is unique. On the other hand, if $u = 0 \mod q$ then the above calculation implies that $v = 0 \mod q$, and thus $S$ is honest. This proves Fact 2.

Towards proving Fact 3, note that, by Fact 1, $C$'s message $Z_1$ does not leak any information about $b$. This implies that all values in $\{1, \ldots, 2^\lambda\}$ are still equally likely even when conditioning over message $Z_1$. Then, by using Fact 2, the probability that $S$'s message $(w_0, w_1)$ satisfies the probabilistic test, is 1 divided by the number $2^\lambda$ of values of $b$ that are still equally likely even when conditioning over message $Z_1$. This proves Fact 3.

### 3.2 Protocol scenario: ($A$ Private Online, $B$ Public Offline)

Our second protocol satisfies the following

**Theorem 2.** Let $e$ be a pairing, as defined in Section 2.1, let $\sigma$ be its computational security parameter, and let $\lambda$ be a statistical security parameter. There exists (constructively) a client-server protocol $(C, S)$ for delegating the computation of $e$, when input $A$ is privately known in the online phase, and input $B$ is publicly known in the offline phase, which satisfies 1-correctness, $2^{-\lambda}$-security, 0-privacy, and efficiency with parameters $(t_F, t_S, t_P, t_C, cc, mc)$, where

- $t_F = p_T$, $t_S = 2\,p_T$ and $t_P = 2\,p_T$;
- $t_C \leq 2\,a_1 + m_1(\lambda) + 2\,m_T + e_T(\lambda) + t_M$;
- $cc = 2$ values in $\mathcal{G}_1 + 2$ values in $\mathcal{G}_T$ and $mc = 2$.

The main takeaway from this theorem is that $C$ can securely and efficiently delegate to $S$ the computation of a bilinear pairing whose first input $A$ is known to $C$ in the online phase and has to remain private, while second input $B$ is publicly known in the offline phase. In particular, in the online phase $C$ only performs 1 exponentiation to a $\lambda$-bit exponent in $\mathcal{G}_T$ and 1 multiplication to a $\lambda$-bit scalar in $\mathcal{G}_1$, and lower-order operations. (See Table 4 for a concrete comparison with best previous work, also showing estimated ratios of $C$'s online runtime to a non-delegated pairing calculation ranging between 0.078 and 0.161 depending on the curve used.) Additionally, $C$ only computes 2 pairings in the offline phase, $S$ only computes 2 pairings, and $C$ and $S$ only exchange 2 messages containing a small number of group values.

**Table 4.** Protocols comparison in the scenario ($A$ private online, $B$ public offline)

| **Protocols** | $t_C$ | **Ratio:** $t_C/t_F$ | | | |
|---|---|---|---|---|---|
| | | **BN-12** | **BLS-12** | **KSS-18** | **BLS-24** |
| | | $\sigma = 461$ | $\sigma = 635$ | $\sigma = 508$ | $\sigma = 629$ |
| [30] [Sec 4.3] | $a_1 + m_1(\sigma) + m_T + e_T(\sigma)$ | 0.572 | 0.464 | 0.304 | 0.455 |
| [11] [Sec 5.2] | $2\,a_1 + m_1(\sigma) + 2\,m_T + e_T(\sigma) + t_M$ | 0.574 | 0.465 | 0.304 | 0.456 |
| Ours [Sec 3.2] | $2\,a_1 + m_1(\lambda) + 2\,m_T + e_T(\lambda) + t_M$ | 0.161 | 0.095 | 0.078 | 0.094 |

**Protocol Description.** This protocol uses as a starting point the protocol from Section 3.1, but includes an additional technique to achieve the additional property that input $A$ remains private. As $S$ does not know $A$, it cannot directly compute $e(A, B)$ as before. Instead, $C$ sends an additional randomly masked version $Z_0$ of $A$ and lets $S$ compute $w_0 = e(Z_0, B)$, where the mask is based on a value $U_0$ for which $C$ had computed $v_0 = e(U_0, B)$ in the offline phase. Using $U_0$ and $v_0$, $C$ can both compute $e(A, B)$ as $w_0 \cdot v_0$ and run membership and probabilistic tests analogously to the previous protocol. A formal description follows.

*Offline Input to C:* $B \in \mathcal{G}_2$

*Offline phase instructions:*

1. $C$ randomly chooses $U_0, U_1 \in \mathcal{G}_1$, and $b \in \{1, \ldots, 2^\lambda\}$
2. $C$ sets $v_0 = e(U_0, B)$ and $v_1 = e(U_1, B)$

*Online Input to C:* $1^\sigma, 1^\lambda$, $desc(e)$, $A \in \mathcal{G}_1$, $B \in \mathcal{G}_2$

*Online Input to S:* $1^\sigma, 1^\lambda$, $desc(e)$, $B \in \mathcal{G}_1$

*Online phase instructions:*

1. $C$ sets $Z_0 := A - U_0$ and $Z_1 := b \cdot A + U_1$
   $C$ sends $Z_0, Z_1$ to $S$
2. $S$ computes $w_0 := e(Z_0, B)$ and $w_1 := e(Z_1, B)$
   $S$ sends $w_0, w_1$ to $C$
3. (Membership Test:) $C$ checks that $w_0 \in \mathcal{G}_T$
   $C$ computes: $y := w_0 \cdot v_0$
   (Probabilistic Test:) $C$ checks that $w_1 = y^b \cdot v_1$
   If any of these tests fails then $C$ **returns** $\perp$ and the protocol halts
   $C$ **returns** $y$

**Properties of protocol** $(C, S)$**:** The *efficiency* properties are verified by protocol inspection. In particular, we note that with respect to the protocol in Section 3.1, this protocol gains privacy of input $A$ with very small additional overhead: 1 additional subtraction in $\mathcal{G}_1$ and 1 additional multiplication in $\mathcal{G}_T$ with respect to $C$'s online work, and 1 additional pairing computation with respect to $C$'s offline work.

The *correctness* property follows by showing that if $C$ and $S$ follow the protocol, $C$ always output $y = e(A, B)$. We show that the 2 tests performed by $C$ are always passed. The membership test is always passed by the pairing definition; the probabilistic test is always passed since

$$w_1 = e(Z_1, A) = e(b \cdot A + U_1, B) = e(A, B)^b \cdot e(U_1, B) = y^b \cdot v_1$$

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

$$\begin{aligned} y = w_0 \cdot v_0 &= e(Z_0, B) \cdot e(U_0, B) = e(A - U_0, B) \cdot e(U_0, B) \\ &= e(A, B) \cdot e(U_0, B)^{-1} \cdot e(U_0, B) = e(A, B). \end{aligned}$$

The *privacy* property of the protocol against any arbitrary malicious $S$ follows by observing that $C$'s only message $(Z_0, Z_1)$ to $S$ does not leak any information about $C$'s input $A$, because both $Z_0$ and $Z_1$ are uniformly and independently distributed in $\mathcal{G}_1$, as so are $U_0$ and $U_1$. Moreover, by essentially the same reasoning, this message does not leak any information about $b$, a fact which we also use in the proof of the security property.

To prove the *security* property against any malicious $S$ we need to compute an upper bound $\epsilon_s$ on the security probability that $S$ convinces $C$ to output a $y$ such that $y \neq e(A, B)$. We obtain that $\epsilon_s \leq 2^{-\lambda}$ as a consequence of the following 3 facts:

1. $(Z_0, Z_1)$ leaks no information about $b$ to $S$;
2. for any $S$'s message $(w_0, w_1)$ different than what would be returned according to the protocol instructions, there is only one $b$ for which $(w_0, w_1)$ satisfy both the membership and the probabilistic test in step 3 of the protocol;
3. for any $S$'s message $(w_0, w_1)$ different than what would be returned according to the protocol instructions, the probability that $(w_0, w_1)$ satisfies the probabilistic test is $\leq 2^{-\lambda}$.

We note that these 3 facts are proved similarly as in the proof of the security property for the protocol in Section 3.1, with a few minor changes due to $C$'s message now being of the form $(Z_0, Z_1)$ instead of just $Z_1$, and the probabilistic test now being $w_1 = (w_0 \cdot v_0)^b \cdot v_1$ instead of $w_1 = w_0^b \cdot v_1$.

Specifically, to prove Fact 1, we observe that the pair $(Z_0, Z_1)$ is uniformly distributed in $\mathcal{G}_1$ since so is pair $(U_0, U_1)$, which is unknown to $S$. Thus, the distribution of $(Z_0, Z_1)$ is independent from that of $b$, from which Fact 1 follows.

Towards proving Fact 2, we only note that the rewriting of the probabilistic test is slightly different than in Section 3.1, but again brings to the same conclusion $v = ub \mod l$. Specifically, the probabilistic test is now rewritten as

$$\begin{aligned} w_1' &= (w_0' \cdot v_0)^b \cdot v_1 \\ g^v \cdot w_1 &= (g^u \cdot e(A, Z_0) \cdot e(A, U_0))^b \cdot e(A, U_1) \\ g^v \cdot e(A, Z_1) &= g^{ub} \cdot e(A, B)^b \cdot e(A, U_1) \\ g^v &= g^{ub} \\ v &= ub \mod l. \end{aligned}$$

Then, the rest of the proof for Fact 2 continues to hold.

The proof for Fact 3 still holds with only syntactic changes by modifying $Z_1$ into $(Z_0, Z_1)$.

### 3.3 Protocol scenario: ($A$ Private Online, $B$ Private Offline)

Our third protocol satisfies the following

**Theorem 3.** Let $e$ be a pairing, as defined in Section 2.1, let $\sigma$ be its computational security parameter, and let $\lambda$ be a statistical security parameter. There exists (constructively) a client-server protocol $(C, S)$ for delegating the computation of $e$, when input $A$ is privately known in the online phase, and input $B$ is privately known in the offline phase, which satisfies 1-correctness, $2^{-\lambda}$-security, 0-privacy, and efficiency with parameters $(t_F, t_S, t_P, t_C, cc, mc)$, where
- $t_F = p_T$, $t_S = 2\,p_T$ and $t_P = 2\,p_T + 2\,m_1(\sigma) + m_2(\sigma) + i_l$;
- $t_C \leq 2\,a_1 + m_1(\sigma) + m_1(\lambda) + 2\,m_T + e_T(\lambda) + t_M$;
- $cc = 3$ values in $\mathcal{G}_1 + 2$ values in $\mathcal{G}_T$ and $mc = 2$.

The main takeaway from this theorem is that $C$ delegates to $S$ can securely and efficiently delegate to $S$ the computation of a bilinear pairing where both inputs $A$ and $B$ have to remain private and first input $A$ (resp., second input $B$) is known to $C$ in the online (resp., offline) phase. In particular, in the online phase $C$ only performs 2 multiplications and 1 exponentiation to a $\lambda$-bit exponent in $\mathcal{G}_T$, 2 additions and 2 multiplications in $\mathcal{G}_1$, and 1 group membership verification in $\mathcal{G}_T$. (See Table 5 for a concrete comparison with best previous work, also showing estimated ratios of $C$'s online runtime to a non-delegated pairing calculation ranging between 0.13 and 0.29 depending on the curve used.) Additionally, $C$ only computes 2 pairings in the offline phase, $S$ only computes 2 pairings, and $C$ and $S$ only exchange 2 messages containing a small number of group values.

**Table 5.** Protocols comparison in the scenario ($A$ private online, $B$ private offline)

| **Protocols** | $t_C$ | **Ratio:** $t_C/t_F$ | | | |
|---|---|---|---|---|---|
| | | **BN-12**<br>$\sigma = 461$ | **BLS-12**<br>$\sigma = 635$ | **KSS-18**<br>$\sigma = 508$ | **BLS-24**<br>$\sigma = 629$ |
| [30] [Sec 4.2] | $a_1 + m_1(\sigma) + m_T$<br>$+2\, e_T(\sigma) + t_M$ | 1.016 | 0.836 | 0.552 | 0.865 |
| Ours [Sec 3.3] | $2\, a_1 + m_1(\sigma) + m_1(\lambda)$<br>$+2\, m_T + e_T(\lambda) + t_M$ | 0.290 | 0.188 | 0.133 | 0.139 |

**Protocol description.** The main idea in this protocol builds on those in protocols from Section 3.1 and 3.2. The difference between the scenario in this section and the scenario in Section 3.2 is that here input $B$ has to remain private. Thus, $S$ cannot directly compute $e(Z_0, B), e(Z_1, B)$ as before. Instead, $C$ applies an additional layer of random masks, based on a single random value $r$, as follows. First, $Z_0 = r^{-1}B$ is used as a masked variant of $B$. Next, $Z_1 = r(A - U_0)$ and $Z_2 = r(bA - U_1)$ are used as doubly-masked variants of $A$, using $r$ to both further mask previously computed values $(A - U_0)$ and $(bA - U_1)$ as well as cancel out exponent $r^{-1}$ after pairing computations. $S$ again computes 2 pairing values: $w_0 = e(Z_1, Z_0)$ and $w_1 = e(Z_2, Z_0)$. Using $U_0, U_1$ and $v_0, v_1$, $C$ can both compute $e(A, B)$ as $w_0 v_0$ and efficiently run a membership test for $w_0$ and a probabilistic test based on $w_1$ analogously to the previous two protocols (since mask $r$ gets canceled out in the pairing computations). The protocol also redistributes the computation of the double masking of $Z_1$ and $Z_2$ so to reduce online runtime at the expense of some additional offline runtime. A formal description follows.

*Offline Input to C:* $B \in \mathcal{G}_2$

*Offline phase instructions:*

1. $C$ randomly chooses $U_0, U_1 \in \mathcal{G}_1$, $b \in \{1, \ldots, 2^\lambda\}$ and $r \in \mathbb{Z}_l$
2. $C$ sets
   - $v_0 = e(U_0, B)$ and $v_1 = e(U_1, B)$
   - $Z_0 := r^{-1} \cdot B$
   - $Z_{1,1} := -r \cdot U_0$
   - $Z_{2,1} := r \cdot U_1$

*Online Input to C:* $1^\sigma, 1^\lambda$, $desc(e)$, $A \in \mathcal{G}_1$, $B \in \mathcal{G}_2$

*Online Input to S:* $1^\sigma, 1^\lambda$, $desc(e)$

*Online phase instructions:*

1. $C$ sets $Z_{1,0} = Z_{2,0} = rA$, $Z_1 = Z_{1,0} + Z_{1,1}$ and $Z_2 = bZ_{2,0} + Z_{2,1}$
   $C$ sends $Z_0, Z_1, Z_2$ to $S$
2. $S$ computes $w_0 := e(Z_1, Z_0)$ and $w_1 := e(Z_2, Z_0)$
   $S$ sends $w_0, w_1$ to $C$
3. (Membership Test:) $C$ checks that $w_0 \in \mathcal{G}_T$
   $C$ computes: $y = w_0 \cdot v_0$
   (Probabilistic Test:) $C$ checks that $w_1 = y^b \cdot v_1$

```
    if any of these tests fails $C$ **returns** $\perp$ and the protocol halts
    $C$ **returns** $y$
```

**Protocol Properties:** The *efficiency* properties are verified by protocol inspection. In particular, we note that with respect to the protocol in Section 3.2, this protocol gains privacy of input $B$ with very small additional overhead: 1 additional scalar multiplication in $\mathcal{G}_1$ with respect to $C$'s online work, 2 scalar multiplications in $\mathcal{G}_1$ with respect to $C$'s offline work, and 1 additional group value sent from $C$ to $S$.

The *correctness* property follows by showing that if $C$ and $S$ follow the protocol, $C$ always output $y = e(A, B)$. We show that the 2 tests performed by $C$ are always passed. The membership test is always passed by the pairing definition. To see that the probabilistic test is always passed, first note that $Z_1 = Z_{1,0} + Z_{1,1} = r(A - U_0)$, and $Z_2 = Z_{2,0} + Z_{2,1} = r(bA + U_1)$. Then

$$\begin{aligned} w_1 &= e(Z_2, Z_0) = e(r \cdot (b \cdot A + U_1), r^{-1} \cdot B) \\ &= e(b \cdot A + U_1, B) = e(A, B)^b \cdot e(U_1, B) = y^b \cdot v_1. \end{aligned}$$

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

$$\begin{aligned} y &= w_0 \cdot v_0 = e(Z_1, Z_0) \cdot e(U_0, B) = e(r \cdot (A - U_0), r^{-1} \cdot B) \cdot e(U_0, B) \\ &= e(A, B) \cdot e(U_0, B)^{-1} \cdot e(U_0, B) = e(A, B). \end{aligned}$$

The *privacy* property of the protocol against any arbitrary malicious $S$ follows by observing that $C$'s only message $(Z_0, Z_1, Z_2)$ to $S$ does not leak any information about $C$'s input $A$ or $B$. This follows because (a) values $Z_1, Z_2$ are uniformly and independently distributed in $\mathcal{G}_1$, as so are $U_0, U_1$; and (b) value $Z_0 = rA$ is uniformly and independently distributed in $\mathcal{G}_1$ as $r$ is a random scalar in $\mathbb{Z}_l$ and $\mathcal{G}_1$ is cyclic. Moreover, by similar reasoning, this message does not leak any information about $b$, a fact useful in the proof of the security property.

The proof for the *security* property is a direct extension of the proof of the security property for protocols in Section 3.1 and 3.2, and therefore we only discuss relevant changes. As before, we compute an upper bound $\epsilon_s$ on the security probability that $S$ convinces $C$ to output a $y$ such that $y \neq e(A, B)$, and we obtain that $\epsilon_s \leq 2^{-\lambda}$ as a consequence of 3 facts, formulated analogously to those in Section 3.1 and 3.2.

Fact 1 says that $C$'s message $(Z_0, Z_1, Z_2)$ leaks no information about $b$ to $S$. This follows by a proof similar (in fact, simpler) than the proof for the privacy property for the same protocol.

Towards proving Fact 2, we only note that the rewriting of the probabilistic test is slightly different than in Section 3.2, but again brings to the same conclusion $v = ub \mod l$. Specifically, the probabilistic test is now rewritten as

$$\begin{aligned} w_1' &= (w_0' \cdot v_0)^b \cdot v_1 \\ g^v \cdot w_1 &= (g^u \cdot e(Z_0, Z_1) \cdot e(A, U_0))^b \cdot e(A, U_1) \end{aligned}$$

$$\begin{aligned}
g^v \cdot e(Z_0, Z_2) &= g^{ub} \cdot e(r^{-1}A, r(B - U_0))^b \cdot e(A, U_0))^b \cdot e(A, U_1) \\
g^v \cdot e(r^{-1}A, r(bB + U_1)) &= g^{ub} \cdot e(A, B - U_0)^b \cdot e(A, bB + U_1))^b \cdot e(A, U_1) \\
g^v \cdot e(A, B)^b \cdot e(A, U_1) &= g^{ub} \cdot e(A, B)^b \cdot e(A, U_1) \\
g^v &= g^{ub} \\
v &= ub \mod l.
\end{aligned}$$

Then, the rest of the proof for Fact 2 continues to hold.

The proof for Fact 3 still holds with only syntactic changes by using $(Z_0, Z_1, Z_2)$ as $C$'s message.

**Extension.** We observe that the above pairing delegation protocol can also be used as a secure pairing delegation protocol in the (A Public Online, B Private Offline) scenario, in which case the improvement over previous work is as shown in Table 6 below.

**Table 6.** Protocols comparison in the scenario ($A$ public online, $B$ private offline)

| **Protocols** | $t_C$ | **Ratio:** $t_C/t_F$ | | | |
|---|---|---|---|---|---|
| | | **BN-12** | **BLS-12** | **KSS-18** | **BLS-24** |
| | | $\sigma = 461$ | $\sigma = 635$ | $\sigma = 508$ | $\sigma = 629$ |
| [14, 15] [Sec 6.2] | $a_2 + m_2(\sigma) + m_T$ $+ e_T(\sigma) + t_M$ | 1.145 | 0.973 | 0.652 | 1.060 |
| Ours [Sec 3.3] | $2\,a_1 + m_1(\sigma) + m_1(\lambda)$ $+ 2\,m_T + e_T(\lambda) + t_M$ | 0.290 | 0.188 | 0.133 | 0.139 |

## 4 Delegating Pairings with Online Inputs

In this section we show that our secure pairing delegation protocols in Section 3 (i.e., in scenarios where at least one input is known in the offline phase) can be combined and give protocols for scenarios where both inputs are known in the online phase. We informally describe two protocols in two different input scenarios, depending on a public/private requirement for both inputs, we defer the formal description to a longer version of the paper, and show in Table 7 a performance comparison with previous constructions for the same scenarios. The performance improvement, although smaller than for the protocols in Section 3, is still significant as for some curves we obtain the first protocol for the (A private online, B private online) scenario with client online runtime smaller than the non-delegated pairing computation time.

**Scenario (A public online, B public online).** Our starting point to design a secure pairing delegation protocol in this scenario is the protocol in Section 3.1, since in both scenarios inputs $A$ and $B$ are publicly known. In that protocol, however, $C$ computes $e(U_1, B)$ in the offline phase, for some random $U_1 \in \mathcal{G}_1$, which is not possible in the current scenario given that $B$ is only known in the online phase. This problem is solved by $C$ delegating the computation of

$e(B, U_1)$, which is equal to $e(U_1, B)$, to $S$ using the protocol in Section 3.3 for the (B private online, $U_1$ private offline) scenario, which suffices for the current scenario, where B is public online and $U_1$ is randomly chosen in the offline phase. Moreover, after combining the two protocols, we observe that this combination has two independent probabilistic tests, which would result in 2 separate exponentiations in $\mathcal{G}_T$ to $\lambda$-bit exponents by $C$.

**Scenario (A private online, B private online).** We start by observing that: (a) $A$ and $B$ are not publicly known and therefore $S$ cannot directly compute $e(A, B)$ as in Section 4; (a) $A$ and $B$ are only known in the online phase and therefore none of the protocols in Section 3 solves this case. However, it turns out that $C$ can suitably randomize both $A$ and $B$ and then use, as a black-box, protocols for the ($A$ public online, $B$ public online) and ($A$ private online, $B$ private offline) scenarios from previous sections, as follows. In the offline phase, $C$ randomly chooses $r \in \mathbb{Z}_l$ and $U \in \mathcal{G}_1$ and set $s = r^{-1}$. Then $C$ and $S$ run the protocol in the ($A'$ public online, $B'$ public online) scenario, where $A' = rA$ and $B' = r^{-1}(B - U)$, and the protocol in the ($A''$ private online, $B''$ private offline) case, where $A'' = A$ and $B'' = U$.

**Table 7.** Protocols comparison in scenarios where both $A$ and $B$ are known online

| **Protocols** | **Scenario** | **Ratio:** $t_C/t_F$ | | | |
|---|---|---|---|---|---|
| | | **BN-12** | **BLS-12** | **KSS-18** | **BLS-24** |
| | | $\sigma = 461$ | $\sigma = 635$ | $\sigma = 508$ | $\sigma = 629$ |
| [14, 15] [Sec 5.2] | A and B public online | 1.719 | 1.439 | 0.956 | 1.517 |
| [11] [Sec 4.1] | A and B public online | 0.832 | 0.697 | 0.460 | 0.697 |
| Ours [Sec 4] | A and B public online | 0.492 | 0.329 | 0.228 | 0.235 |
| [14, 15] [Sec 4.1] | A and B private online | 2.606 | 2.182 | 1.453 | 2.337 |
| [30] [Sec 3] | A and B private online | 1.719 | 1.439 | 0.956 | 1.517 |
| [11] [Sec 5.1] | A and B private online | 1.658 | 1.391 | 0.917 | 1.390 |
| Ours [Sec 4] | A and B private online | 1.090 | 0.777 | 0.540 | 0.649 |

## 5 Conclusions

In this paper we showed techniques for a computationally weaker client to efficiently, privately and securely delegate bilinear pairings to a single, possibly malicious, server. Efficiency gains obtained by our resulting protocols with respect to the main metric (client's online runtime) can be up to almost 1 order of magnitude, regardless of which of the most practical elliptic curves are used for the pairing realization. Our techniques improve the state of the art on all input scenarios and are therefore applicable to essentially all known pairing-based cryptographic protocols. Our largest improvements are in scenarios where at least one of the two pairing inputs is known in the offline phase, which happens to be a very typical situation in published protocols (e.g., one input is part of a public key). Even when both pairing inputs are known in the online phase, for some elliptic curves we show the first protocol that improves over non-delegated computation when both inputs have privacy requirements.

## References

1. S.S. Al-Riyami, K.G. Paterson, *Certificateless Public Key Cryptography.* In Laih C.S. (eds) Advances in Cryptology - ASIACRYPT 2003.
2. N. Asokan, Gene Tsudik, Michael Waidner, *Server-Supported Signatures*, in Journal of Computer Security 5(1): 91-108 (1997)
3. M. Atallah, K. Pantazopoulos, J. Rice, E. Spafford, *Secure outsourcing of scientific computations.* In Advances in Computers, 54, pp. 215–272, 2002.
4. M. Atallah and K. Frikken, *Securely outsourcing linear algebra computations*, In Proc. of 5th ACM ASIACCS, 2010, pp. 48–59.
5. 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) LATINCRYPT 2015. LCNS vol. 9230. Springer.
6. D. Benjamin and M. Atallah, *Private and cheating-free outsourcing of algebraic computations*, In 6th Ann. Conf. on Privacy, Security and Trust, 2008, pp. 240–245.
7. D. Boneh, M. Franklin, *Identity-based encryption from the weil pairing.* In Kilian J. (eds) Advances in Cryptology — CRYPTO 2001. LNCS vol 2139. Springer.
8. D. Boneh, G. Di Crescenzo, R. Ostrovsky, G. Persiano, *Public Key Encryption with Keyword Search.* In Cachin C., Camenisch J.L. (eds) Advances in Cryptology - 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. J.W. Bos, C. Costello, M. Naehrig, *Exponentiating in pairing groups.* In Lange T., Lauter K., Lisoněk P. (eds) SAC 2013. LNCS vol 8282. Springer.
11. S. Canard, J. Devigne, O. Sanders: *Delegating a pairing can be both secure and efficient.* In Boureanu I., Owesarski P., Vaudenay S. (eds) Applied Cryptography and Network Security. ACNS 2014. LNCS vol 8479. Springer
12. B. Cavallo, G. Di Crescenzo, D. Kahrobaei, V. Shpilrain, *Efficient and secure delegation of group exponentiation to a single server.* In Proc. of Intern. Workshop on Radio Frequency Identification: Security and Privacy Issues: pp. 156–173, LNCS, Springer.
13. X. Chen, J. Li, J. Ma, Q. Tang, W. Lou, *New Algorithms for Secure Outsourcing of Modular Exponentiations.* In IEEE Transactions on Parallel and Distributed Systems, vol. 25, no. 9, pp. 2386–2396, Sept. 2014.
14. B. Chevallier-Mames, J.S. Coron, N. McCullagh, D. Naccache, M. Scott: *Secure delegation of elliptic-curve pairing. Cryptology ePrint Archive.* In report 2005/150 (2005), http://eprint.iacr.org/2005/150.
15. B. Chevallier-Mames, J.S. Coron, N. McCullagh, D. Naccache, M. Scott: *Secure delegation of elliptic-curve pairing. Cryptology ePrint Archive.* In Gollmann D., Lanet JL., Iguchi-Cartigny J. (eds) CARDIS 2010. LNCS vol 6035. Springer.
16. C. Chevalier, F. Laguillaumie, D. Vergnaud, *Privately Outsourcing Exponentiation to a Single Server: Cryptanalysis and Optimal Constructions.* In Askoxylakis I., Ioannidis S., Katsikas S., Meadows C. (eds) Computer Security – ESORICS 2016. LNCS vol. 9878. Springer.

17. K. Chung K, Y. Kalai, S. Vadhan, *Improved delegation of computation using fully homomorphic encryption.* In Proc. of CRYPTO 2010, pp. 483–501, LNCS 6223, Springer, 2010.
18. G. Di Crescenzo, M. Khodjaeva, D. Kahrobaei, and V. Shpilrain, *Practical and Secure Outsourcing of Discrete Log Group Exponentiation to a Single Malicious Server.* In Proc. of 9th ACM Cloud Computing Security Workshop (CCSW 2017), pp. 17–28, 2017.
19. G. Di Crescenzo, D. Kahrobaei, M. Khodjaeva, V. Shpilrain, *Efficient and Secure Delegation to a Single Malicious Server: Exponentiation over Non-abelian Groups.* In Proc. of ICMS 2018, pp. 137–146, LNCS 10931, Springer.
20. M. Dijk, D. Clarke, B. Gassend, G. Suh, S. Devadas, *Speeding Up Exponentiation using an Untrusted Computational Resource.* In Designs, Codes and Cryptography, 39 (2), pp. 253-273, 2006.
21. D. Fiore, R. Gennaro, *Publicly verifiable delegation of large polynomials and matrix computations, with applications.* In Proc. of ACM CCS Conf. 2012, pp. 501–512.
22. 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.
23. M. Girault, D. Lefranc, *Server-aided verification: Theory and practice.* In Roy, B.K. (ed.) ASIACRYPT 2005. LNCS, vol. 3788, pp. 605–623.
24. S. Goldwasser, Y. Tauman Kalai, G. N. Rothblum, *Delegating Computation: Interactive Proofs for Muggles.* In J. of the ACM (JACM), vol. 62(4), pp 1–64, 2015.
25. Guillevic, A., Vergnaud, D.: *Algorithms for outsourcing pairing computation.* In Joye M., Moradi A. (eds) Smart Card Research and Advanced Applications. CARDIS 2014. LNCS vol 8968. Springer.
26. F. Hess, *Efficient Identity Based Signature Schemes Based on Pairings.* In Nyberg K., Heys H. (eds) Selected Areas in Cryptography. SAC 2002.
27. S. Hohenberger, A. Lysyanskaya, *How to securely outsource cryptographic computations.* In Proc. of TCC 2005, pp. 264–282, Springer.
28. Antoine Joux, *A One Round Protocol for Tripartite Diffie-Hellman.* In Proc. of ANTS 2000, pp. 385-394
29. 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.
30. B.G. Kang, M.S. Lee, J.H. Park, *Efficient delegation of pairing computation.* In journal IACR Cryptology ePrint Archive, pp 259, 2005, Citeseer.
31. J.K. Liu, M.H. Au, W. Susilo, *Self-generated-certificate publickey cryptography and certificateless signature/encryption schemein the standard model.* In Proc. ACM Symp. on Information, Computer and Communications Security. ACM Press (2007).
32. X. Ma and J. Li, F. Zhang, *Outsourcing computation of modular exponentiations in cloud computing.* In Cluster Computing (2013) 16:787-796 (also INCoS 2012).
33. 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.
34. M. Scott, *Unbalancing pairing-based key exchange protocols.* In IACR Cryptology ePrint Archive, vol. 2013, pp. 688, 2013, Citeseer. http://eprint.iacr.org/
35. Y. Shi, J. Li, *Provable Efficient Certificateless Public Key Encryption.* In Cryptology ePrint, Archive, Report 2005/284, 2005.
36. F. Vercauteren, *Optimal Pairings.* In IEEE Transactions on Information Theory, vol. 56, no. 1, pp. 455–461, Jan. 2010.
37. A. Yao, *Protocols for secure computations*, In Proc. of 23rd IEEE FOCS, pp. 160–168, 1982.