LOVE a Pairing*

Diego F. Aranha[1], Elena Pagnin[2], and Francisco Rodríguez-Henríquez[3]

[1] Aarhus University, Aarhus, Denmark
dfaranha@cs.au.dk
[2] Lund University, Lund, Sweden
elena.pagnin@eit.lth.se
[3] Cryptography Research Centre, Technology Innovation Institute, UAE
Computer Science Department, Cinvestav, México
francisco@cs.cinvestav.mx

Abstract. The problem of securely outsourcing the computation of a bilinear pairing has been widely investigated in the literature. Designing an efficient protocol with the desired functionality has, however, been an open challenge for a long time. Recently, Di Crescenzo et al. (CARDIS'20) proposed the first suite of protocols for *securely and efficiently* delegating pairings with online inputs under the presence of a *malicious server*. We progress along this path with the aim of LOVE (Lowering the cost of Outsourcing and Verifying Efficiently) a pairing. Our contributions are threefold. First, we propose a protocol (LOVE) that improves the efficiency of Di Crescenzo et al.'s proposal for securely delegating pairings with online, public inputs. Second, we provide the first implementation of efficient protocols in this setting. Finally, we evaluate the performance of our LOVE protocol in different application scenarios by benchmarking an implementation using BN, BLS12 and BLS24 pairing-friendly curves. Interestingly, compared to Di Crescenzo et al.'s protocol, LOVE is up to 29.7% faster for the client, up to 24.9% for the server and requires 23-24% less communication cost depending on the choice of parameters. Furthermore, we note that our LOVE protocol is especially suited for subgroup-secure groups: checking the correctness of the delegated pairing requires up to 56.2% less computations than evaluating the pairing locally (no delegation). This makes LOVE the most efficient protocol to date for securely outsourcing the computation of a pairing with online public inputs, even when the server is malicious.

1 Introduction

Cryptographic bilinear pairings (a.k.a. pairings, in short) have proven to be an extremely versatile building block to realize novel and advanced cryptographic tools including identity-based encryption [13], short signatures [15], aggregate signatures [14], and zero knowledge-Succinct Non-interactive ARgument of Knowledge (zk-SNARK) [28]. Very recently, pairings found applications in isogeny-based cryptography, to compress public keys in key exchange [46] and to construct verifiable delay functions [22].

Pairing-based protocols critically rely on an efficient implementation of the pairing, which has computational cost far more expensive than any other of the protocol's building blocks. Several clever algorithmic breakthroughs [9,42], capitalized on efficient software and hardware implementations (see [1],[37, Chapter 11] for a comprehensive overview), producing an impressive reduction of the latency associated to a pairing. Nonetheless, as of 2015, the timing cost for the execution of a single pairing on the BN curve at the 128-bit security level, was five to six times higher than the one of a scalar multiplication (over $\mathbb{G}_1$), [47, Table II]. The considerably higher cost of evaluating a pairing motivated a line of research on how to outsource this computation in a secure and efficient way.

Secure and efficient pairing delegation. For many years, researchers and developers have addressed the problem of how a resource-constrained device (Client), can safely delegate the computation of a pairing to a much more powerful computational entity (Server). This setting is particularly relevant in the Internet of the Things (IoT): if secure and efficient pairing delegation is possible, IoT devices (acting as clients) can manage advanced pairing-based protocols without having to pay the cost of locally evaluating pairings. Intuitively, a protocol for secure and efficient pairing delegation should provide mechanisms allowing the client to verify the correctness of the output returned by the server. With respect to efficiency, we want the client's computational costs associated to such delegation be strictly less expensive than the action

* This is a full version of [5], appeared in Latincrypt 2021.

of computing the pairing solely on the client's device. However, the verification normally involves the computation of costly exponentiations (over $\mathbb{G}_T$), membership tests (in $\mathbb{G}_T$), and at times, additional lighter operations such as scalar multiplications (on $\mathbb{G}_1$ and $\mathbb{G}_2$). Progressive efficiency improvements on pairing evaluation rapidly closed the gap between the cost of verifying the delegated pairing and actually computing the pairing locally. As a result, many of the pairing delegation protocols with the verifiability property proposed to date [32,26,18,17], fail to meet the efficiency requirement stated above. This situation has called to question the whole idea of delegating a pairing in the first place.

In 2020, Di Crescenzo, Khodjaeva, Kahrobaei and Shpilrain put forth a promising solution to realize efficient pairing delegation in the offline/online setting [21]. In a nutshell, this means that the protocol splits into two subsequent phases: an offline phase (run by the client only), followed by an online phase when the inputs to the pairing are disclosed and the client interacts with the server. The key idea is that the offline phase is independent of the pairing inputs, can be run at any point in time, and collects the bulk of the computation required from the client. In contrast, the online phase should be as lightweight as possible for the client, so that verifying the outsourced pairing computation is less expensive than evaluating the pairing locally on the client device. In this paper, we carefully investigate about the efficiency claims of [21] in the context of the new parameter recommendations for pairings at the 128- and 192-bit security level. We additionally introduce minor changes to the original protocol to further optimize its efficiency and test our implementation on a simulated client-server interaction.

Our contributions. This paper provides the first implementation of a secure and efficient protocol for pairing delegation in the offline/online setting. We focus only on the case of public inputs, because our experimental results indicate that delegating a pairing with private inputs remains inefficient and more expensive than performing the local computation. Concretely, we take the most efficient protocols proposed in [21] and make slight but clever modifications with the aim of LOVE (Lowering the cost of Outsourcing and Verifying Efficiently) a pairing. As a result, we obtain the most efficient protocols to date for securely outsourcing the computation of a pairing with online public inputs, even in cases where we cannot trust the server. We formally prove the security for our 'adjusted' protocol LOVE. Finally, we experimentally evaluate LOVE with several choices of curves at different security levels. As a byproduct (and a result of independent interest), we provide updated costs for scalar multiplication and exponentiation in pairing groups using optimized implementations. Interestingly, in lieu of the new optimizations, the performance improvement of delegating a pairing is lower than the reported in previous work, when state of the art implementations are used and the cost of membership checks in $\mathbb{G}_T$ is considered. Furthermore, our results reinforce the observation stated in [7] that even at the cost of a small performance penalty for its individual building blocks, choosing subgroup-secure parameters provides an overall better performance when the whole protocol is analyzed.

Applications. Delegating the computation of a pairing on public inputs may seem a task with little use, yet, we will argue next that it has interesting implications in the realm of efficient verification.

First of all, such a scheme can be deployed to realize server-aided signature verification for schemes that involve pairings in the verification process. This setting has been studied, e.g., in [38], and becomes of particular interest for verifications that involve several pairing computations, e.g., [4]. We note that, if one assumes a trusted set up (for instance, a set up that outputs $\gamma = e(P_1, P_2)$), verifiers could leverage the pairing γ provided by the set up in their offline phase, and thus run the signature verification without needing to ever compute a pairing locally. This simple observation is of particular interest for IoT devices, where one may wish to minimize the code loaded on a constrained device without compromising too much its limited computing resources.

Another venue of application for delegating the computation of a pairing on public inputs is the recent isogeny-based Verifiable Delay Function (VDF) construction presented in [22]. VDFs [12], have important applications for Blockchain proof of space and stake, design of trustworthy randomness beacons and benchmarking of high-end servers, among others. In a VDF setting, given an input challenge x and public parameters pp, the Prover must compute a function $\mathit{Eval}(pp, x) \mapsto (y, \pi)$, where y is the output of the function Eval and π is its proof. A second entity, known as the Verifier, must compute a decision function $\mathit{Verify}(pp, x, y, \pi) \mapsto \{True, False\}$, which determines whether the Prover satisfactorily completed its task or not. By design running Eval shall take time comparable with a prescribed delay T; more formally, it should be computationally intractable, regardless of the amount of parallelization employed by the Prover, to calculate Eval in time less than T. Moreover, once y along with its proof π are

produced, the output y should be easily verifiable by anyone in a much shorter Polylog(T) time. Recently, De Feo, Masson, Petit and Sanso proposed in [22] an isogeny-based VDF construction that uses a pairing for its verification algorithm. In this protocol, the verifier sets up the scheme, and checks the correctness of the evaluation's output by computing two pairings (and by performing other, less expensive checks). Notably, the pairings' inputs are public values, so it seems natural to apply our technique: include the pairing delegation setup in the VDF set up, and enjoy a more efficient verification procedure. This change clearly increases the computational demands on the Prover (running $Eval$) and thus its delay, which is a desirable feature in the VDF setting, and at the same time it speeds up the verification. At the moment, the improvement we described above only works for one of the pairings (the right hand side one, on line 2. of Verify in Figure 1 and 2 of [22]) and assuming that the verifier knows the point Q at set up time.

1.1 Related Work

The seminal work on secure pairing delegation protocols is due to Girault and Lefranc [26] who formalized this notion as *Server-Aided Verification*. The aim of [26] was to improve the efficiency of signature verification by relying on a server to carry out the expensive pairing computation. This approach sparked a long line of research, which includes more expressive models for server-aided verification [44,19,38], security notions for pairing delegation (in the framework of verifiable computation) [18], and several constructions aiming at concrete efficiency and/or better security [32,45,43,17,31,21]. Paradoxically, the state of the art in this matter seems to suggest that delegating a pairing computation in a secure and verifiable way inherently requires more computations than evaluating the pairing locally. To overcome this problem, Di Crescenzo et al. [21] adopted a new strategy. Instead of relying on the standard server-aided verification syntax (two-message protocol), they considered an offline phase (traditionally called key generation, which runs independently of the computational input), and an online phase where the pairing arguments are disclosed and the verifier (acting as a client) interacts with the server. The offline/online approach seems a winning concept: it allows the verifier to run the bulk of computations during the offline phase, which may happen at any point in time before the actual pairing computation is needed. Once the pairing arguments are disclosed, the verifier enjoys more efficient procedures that rely on the output of the expensive offline phase. While this setting is promising, [21] provides no concrete implementation of the suggested protocols and the efficiency estimates are extrapolated from a hypothetical text-book implementation using the well-known, but by now outdated, performance figures from [16].

Interestingly, the problem of pairing delegation appears to be easier in the *batch* setting, where the client wants to compute several pairings $e(A_i, B_i)$ for $A_i \in \mathbb{G}_1$ and $B_i \in \mathbb{G}_2$. The first solution came out in 2007, when Tsang, Chow and Smith [41] proposed the first batch pairing delegation protocols and related security notions. They classified the possible pairing arguments in 16 types (all combinations of public/secret, variable/constant inputs) and proposed protocols tailored to 4 of these settings. Unfortunately, their main protocol was limited to pairings sharing the same secret first argument and involved costly exponentiation in the target group. Later, Mefenza and Vergnaud [35] proposed new efficient batch pairing delegation protocols in the same settings by adopting the endomorphism idea from Guillevic and Vergnaud [31] and reducing the size of exponents. Performance improvements ranged from 40% to 74% at the 128-bit security level in comparison with previous work.

2 Preliminaries

Notation We denote by λ (resp. σ) the computational (resp. statistical) security parameter of a scheme. We use *choosing at random* or *randomly choosing* to refer to sampling from the given set according to the uniform distribution, and denote this by $x \xleftarrow{\$} X$. We denote by $\mathsf{poly}(\lambda)$ a generic polynomial function in the variable λ, and by negl a negligible function, that is $\mathsf{negl}(\lambda) < 1/\mathsf{poly}(\lambda)$, for any poly and large enough values of λ. We denote by $\mathsf{cost}(\cdot)$ a function that, given as input an algorithm returns its computational cost (in some desired computational model). Unless otherwise specified, all groups we work with have order q, which is a 2λ-bit prime; and P_i denotes a generator of the group cyclic group $\mathbb{G}_i$. We denote by $\mathtt{Bool}(\cdot)$ the boolean function that returns 1 if the statement given in input is true / satisfied, and 0 otherwise.

The parameters $p, q, \phi_k(p)$ and k, denote the base field prime, the pairing group order and the k-th cyclotomic polynomial evaluated at p and the embedding degree, respectively. These parameters are formally defined next.

2.1 Pairings

Let E be an elliptic curve defined over the finite field $\mathbb{F}_p$, where p is a large prime. Denote by $E(\mathbb{F}_p)$ the set of points $(x, y) \in \mathbb{F}_p$ that satisfy the elliptic curve equation along with the point at infinity denoted by $\mathcal{O}$. It is known that $E(\mathbb{F}_p)$ forms an additive Abelian group with respect to the elliptic point addition operation. Let $\#E$ denote the cardinality of $E(\mathbb{F}_p)$, and let q be a large prime that divides $\#E$ with $\gcd(q, p) = 1$. Then, the *embedding degree* of a curve is defined as the smallest integer k, such that q divides $p^k - 1$. Let $\mathbb{F}_{p^k}$ be an *extension field* of $\mathbb{F}_p$ of degree k, and let $\mathbb{F}^*_{p^k}$ be the field composed by the non-zero elements of $\mathbb{F}_{p^k}$. We say that $\mathbb{G}_1, \mathbb{G}_2$ and $\mathbb{G}_T$ are an order-q subgroup of $E(\mathbb{F}_p)$, an order-q subgroup of $E(\mathbb{F}_{p^k})$, and the order-q subgroup of $\mathbb{F}^*_{p^k}$, respectively. Groups $\mathbb{G}_1, \mathbb{G}_2$ are typically written additively, while group $\mathbb{G}_T$ is always written multiplicatively.

The standard procedure for computing a pairing is based on an iterative algorithm, proposed by Victor Miller in 1986 [36]. Let $R \in E(\mathbb{F}_{p^k})$ and let s be a non-negative integer. A *Miller function* $f_{s,R}$ of length s is a function in $\mathbb{F}_{p^k}(E)$ with divisor $(f_{s,R}) = s(R) - (sR) - (s-1)(\infty)$, where ∞ denotes the point at infinity. Miller's algorithm calculates a value f that is only unique up to a multiplicative power of q. The *reduced* Tate pairing computes a *final exponentiation* step, where the value f is raised to the power $(p^k - 1)/q$. This exponentiation is known as the final exponentiation, and maps the result into the desired subgroup of q-th roots of unity. For even embedding degree k and k-th cyclotomic polynomial $\psi_k(\cdot)$, the final exponentiation is split in the *easy* and *hard* parts as $(p^k-1)/q = \left[(p^{k/2} - 1) \cdot (p^{k/2} + 1)/\phi_k(p)\right] \cdot [\phi_k(p)/q]$. This way one gets a bilinear pairing, whose main properties are summarized below.

A pairing is an efficiently-computable map $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ defined over groups of prime order q, that enjoys the following properties:

Bilinearity $e(aP_1, bP_2) = e(P_1, P_2)^{ab}$, $\forall a, b \xleftarrow{\$} \mathbb{Z}_q, P_1 \in \mathbb{G}_1$ and $P_2 \in \mathbb{G}_2$
Non-degeneracy if P_1 and P_2 are generators of $\mathbb{G}_1$ and $\mathbb{G}_2$ respectively, then $g_T = e(P_1, P_2)$ is a generator for $\mathbb{G}_T$.

The pairing e is of Type 1 (symmetric) if $\mathbb{G}_1 = \mathbb{G}_2$. This implies that the curve is equipped with a *distortion map* to produce a linearly independent second argument for non-degeneracy. The pairing e is of Type 3 (asymmetric) if $\mathbb{G}_1 \neq \mathbb{G}_2$ and there are no homomorphisms between the two groups. In the latter case a *twist* is typically used to compress group elements in $\mathbb{G}_2$.

The state of the art in pairing-based cryptography employs the optimal Ate pairing [42] operating on a *family* of curves of small embedding degree, called *pairing-friendly* [23]. Pairing-friendly curves are specified by means of associated parameterized polynomial formulae for the prime modulus p and the prime order subgroup q. For the sake of efficiency, these formulae are instantiated using *seeds* with low Hamming weight (cf. Table 1). Known pairing-friendly families offer different trade-offs between the field sizes (for security in $\mathbb{G}_T$), and curve orders (for security in $\mathbb{G}_1$ and $\mathbb{G}_2$). With the aim of achieving a better performance, we normally choose larger embedding degrees when targeting higher security levels. This design decision allows us to work with moderate sizes of the base field and the curve order.

Selecting a suitable pairing-friendly curve and its associated finite fields and pairing parameters requires trying many *seeds* with low Hamming weight, until a curve with the right performance properties and security requirements is found, inside the chosen family. Design aspects to be considered include the existence of endomorphisms to accelerate scalar multiplication and exponentiation in the pairing groups, the degree of the twist, an optimized *towering* to represent $\mathbb{F}_{p^k}$, efficient ways to test for membership or to hash bit strings to group elements, among others. Security requirements include the hardness of solving the discrete logarithm problem in all groups, and the necessity of verifying that group elements have the right order and were not maliciously selected. The latter is alleviated by choosing curves providing *subgroup security* [7], which mandates that $E(\mathbb{F}_p)$ and $E(\mathbb{F}_{p^k})$ do not contain subgroups significantly smaller than the subgroups $\mathbb{G}_1$ and $\mathbb{G}_2$, both of prime order q. The related $\mathbb{G}_T$-strength security notion applies this idea to $\mathbb{G}_T$ only [39]. Checking the order of group elements is called *subgroup membership testing*.

After the TNFS algorithm was proposed to solve the discrete logarithm in parameterized composite-degree extension fields [33], prime-order Barreto-Naehrig curves [10] lost the top performance spot at 128-bit security. Currently, the families that offer better performance are Barreto-Lynn-Scott curves (BLS) [8] with embedding degree 12 at the 128-bit security level, and 24 at the 192-bit security level [6]. The corresponding curve with embedding degree 48 has been considered for the 256-bit security level [34].

3 Delegating Pairings with Online Public Inputs

In this section, we recall part of Di Crescenzo et al.'s work [21] both for completeness and for providing more intuitive notations, descriptions, as well as a more rigorous formalism. Concretely, we begin by presenting a formal framework for offline/online pairing delegation, and a suitable security model. Our goal here is to spell out the details of the intuitions provided in [21], by having rigorous definitions, which simplify the well-established VC model of [25] to the case of pairing delegation. We then describe the original protocol for online public inputs provided in [21], with an improved notation, and along with correctness, security and efficiency considerations.

3.1 Modeling Offline/Online Pairing Delegation Protocols

We describe a formal model for offline/online pairing delegation. In a nutshell, this model makes use of correctness and (output) security as introduced for verifiable computation (VC) by Gennaro, Gentry and Parno [25]. These notions are, however, adapted (and simplified) to the special setting of our work. We prefer to re-name the standard VC algorithms (KeyGen, ProbGen, Compute, and Verify) to something with a more explicit meaning for our setting, namely (offSetup, onSetup, Compute, and onVerify).

Definition 1 (Offline/Online Pairing Delegation). *An offline/online protocol for pairing delegation consists of the five algorithms* (GlobalSetup, offSetup, onSetup, Compute, onVerify) *:*

- GlobalSetup$(\lambda) \rightarrow$ `bilin.group` *this is a randomized algorithm that takes as input a value* λ *(the computational security parameter) and returns the description of a bilinear group* `bilin.group` $= (q, \mathbb{G}_1, P_1, \mathbb{G}_2, P_2, \mathbb{G}_T, e)$*, where* q *is a* 2λ*-bit prime, and* e *is a pairing. We assume* `bilin.group` *is implicitly available to all subsequent algorithms. (This is a one-time set up).*
- offSetup$(\sigma) \rightarrow$ `off.pp` *this is a randomized algorithm that takes as input a value* σ *(the statistical security parameter). It returns some values* `off.pp`*. (This algorithm is run in by the* client *during the offline phase).*
- onSetup$($`off.pp`$, (A, B)) \rightarrow$ (`pub`, `sec`) *this is a randomized algorithm that takes as input* `off.pp`*, and a pairing argument* $(A, B) \in \mathbb{G}_1 \times \mathbb{G}_2$*. It returns a public value* `pub`*, and a secret value* `sec`*. (This algorithm is run by the client, and is the first algorithm of the online phase. At this point* `off.pp` *and* `sec` *are only known to the* client*, while* `pub` *will be sent to the* server*).*
- Compute$($`pub`$) \rightarrow$ `out` *this is a deterministic algorithm that takes as input the public value* `pub`*; and returns a public output* `out`*. (This algorithm is run by the* server*, and is the second algorithm of the online phase).*
- Verify$($`sec`$,$ `out`$) \rightarrow$ `value` *this is a deterministic algorithm that takes as input the secret value* `sec` *(generated by the online setup) and the server's output* `out`*. It returns a value* `value` $\in \{\mathbb{G}_T \cup \perp\}$*. Intuitively,* `value` $= \perp$ *rejects the outcome of the delegation, while if* `value` $\in \mathbb{G}_T$ *we expect that* `value` $= e(A, B)$*. (This algorithm is run by the* client*, and is the last algorithm of the online phase. It is designed to verify the correctness of the computation carried out by the* server*).*

Figure 1 displays a graphical summary of the syntax introduced in Definition 1.

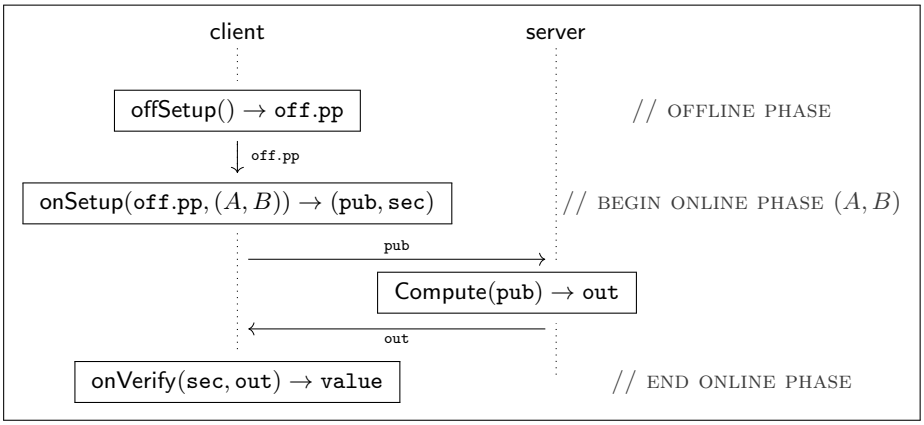

Fig. 1. Diagram visualizing the model for offline/online pairing delegation. Notably, the pairing arguments (A, B) are revealed only at the start of the online phase. The one-time GlobalSetup is omitted from the picture.

A protocol for offline/online delegation of a pairing computation is correct if for all possible input arguments $(A, B) \in \mathbb{G}_1 \times \mathbb{G}_2$, (and for any possible randomness used by $\mathsf{offSetup}$ and $\mathsf{onSetup}$) the protocol execution returns $\mathtt{value} = e(A, B)$, assuming all algorithms are run honestly. This is formalized by the following definition (which is closely similar to the correctness for VC in [25], but tailored to our case of interest).

Definition 2 (Correctness). *A protocol for offline/online pairing delegation is correct if for any value of λ and σ, and for all of possible input arguments $(A, B) \in \mathbb{G}_1 \times \mathbb{G}_2$ it holds that:*

$$Prob\left[\mathtt{value} = e(A,B) \;\middle|\; \begin{array}{c} \mathtt{bilin.group} \leftarrow \mathsf{GlobalSetup}(\lambda) \\ \mathtt{off.pp} \leftarrow \mathsf{offSetup}(\sigma) \\ (\mathtt{pub}, \mathtt{sec}) \leftarrow \mathsf{onSetup}(\mathtt{off.pp}, (A,B)) \\ \mathtt{out} \leftarrow \mathsf{Compute}(\mathtt{pub}) \\ \mathtt{value} \leftarrow \mathsf{onVerify}(\mathtt{sec}, \mathtt{out}) \end{array}\right] = 1.$$

A protocol for offline/online delegation of a pairing computation is secure if no adversary (in the shoes of a malicious server) is able to produce a value $\mathtt{out}^*$ that is not rejected by the verifier and that results in a incorrect output $\mathtt{value}^* \neq e(A, B)$. This is formalized in the following security definition and the experiment $\mathbf{Exp}^{\mathsf{sec}}_{\mathcal{A}}$. Notably, $\mathbf{Exp}^{\mathsf{sec}}_{\mathcal{A}}$ is a simplification of the security experiment for VC in [25]: we reduce the number of adversarial queries to a single one, since in the setting of [21], every new input (A, B) requires a new run of $\mathsf{offSetup}$. Our adversary is a pair of algorithms $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ that share an internal state st.

Security Experiment $\mathbf{Exp}^{\mathsf{sec}}_{\mathcal{A}}(\lambda, \sigma)$

1 : $\mathtt{bilin.group} \leftarrow \mathsf{GlobalSetup}(\lambda)$
2 : $\mathtt{off.pp} \leftarrow \mathsf{offSetup}(\sigma)$
3 : $((A, B), st) \leftarrow \mathcal{A}_1(\lambda, \sigma, \mathtt{bilin.group})$
4 : $(\mathtt{pub}, \mathtt{sec}) \leftarrow \mathsf{onSetup}(\mathtt{off.pp}, (A, B))$
5 : $\mathtt{out}^* \leftarrow \mathcal{A}_2(st, \mathtt{pub}, (A, B))$
6 : $\mathtt{value}^* \leftarrow \mathsf{onVerify}(\mathtt{sec}, \mathtt{out}^*)$
7 : **if** $\mathtt{value}^* = \perp$ **return** 0
8 : **if** $\mathtt{value}^* = e(A, B)$ **return** 0
9 : **return** 1

We remark that, in order to reach the winning condition in $\mathbf{Exp}^{\mathsf{sec}}_{\mathcal{A}}$, the adversary needs to produce an output $\mathtt{out}^*$ that is not rejected by the verification (i.e., $\mathtt{value}^* \neq \perp$) and that yields an incorrect value $\mathtt{value}^* \neq e(A, B)$. Such an output would indeed fool the client into accepting an incorrect value as the result of the outsourced pairing computation. A protocol is secure if any adversary has only negligible probability of winning the security experiment $\mathbf{Exp}^{\mathsf{sec}}_{\mathcal{A}}$.

Definition 3 (Security). *A protocol for offline/online pairing delegation is said to be secure if for any probabilistic, polynomial time algorithm $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ it holds that:*

$$Prob\left[\mathbf{Exp}^{\mathsf{sec}}_{\mathcal{A}}(\lambda, \sigma) = 1\right] \leq 2^{-\sigma} + \mathsf{negl}(\lambda).$$

Regarding efficiency, we cannot use the amortized efficiency framework of VC, where the computational cost of running KeyGen –our $\mathsf{offSetup}$– can be amortized over several executions of the core delegation protocol. In our case, for security reasons, the output of $\mathsf{offSetup}$ can be used only for a single pairing delegation. As we discussed already in the introduction, it is hopeless to expect a pairing delegation protocol be efficient in the strictest sense; the best we can hope to achieve is efficiency in the online verification. This is formalized in the following definition.

Definition 4 (Efficient Online Verification). *A protocol for offline/online pairing delegation is said to have efficient online verification if $(\mathsf{cost}(\mathsf{onSetup}) + \mathsf{cost}(\mathsf{onVerify})) < \mathsf{cost}(e(\cdot, \cdot))$, i.e., the cost of running the online phase on the client-side is less than the cost of computing the pairing on the client's device.*

3.2 Di Crescenzo et al.'s Protocol

In [21], Di Crescenzo et al. propose five different protocols for securely delegating the computation of $e(A, B)$, given the points $A \in \mathbb{G}_1, B \in \mathbb{G}_2$. The most efficient protocol (described in Section 3 of [21], and here in Figure 2) works in the setting where (A, B) are public. In the protocol description, the value q (which determines the size of the field from which r is sampled), depends on the security parameter λ (that sets up the bilinear group). The value σ, instead, represents the parameter for statistical security that delivers the information theoretic security guarantee of the protocol. Finally, we recall that the handle $\texttt{bilin.group} = (q, \mathbb{G}_1, P_1, \mathbb{G}_2, P_2, \mathbb{G}_T, e)$ generated by $\mathsf{GlobalSetup}(\lambda)$, is available to all algorithms.

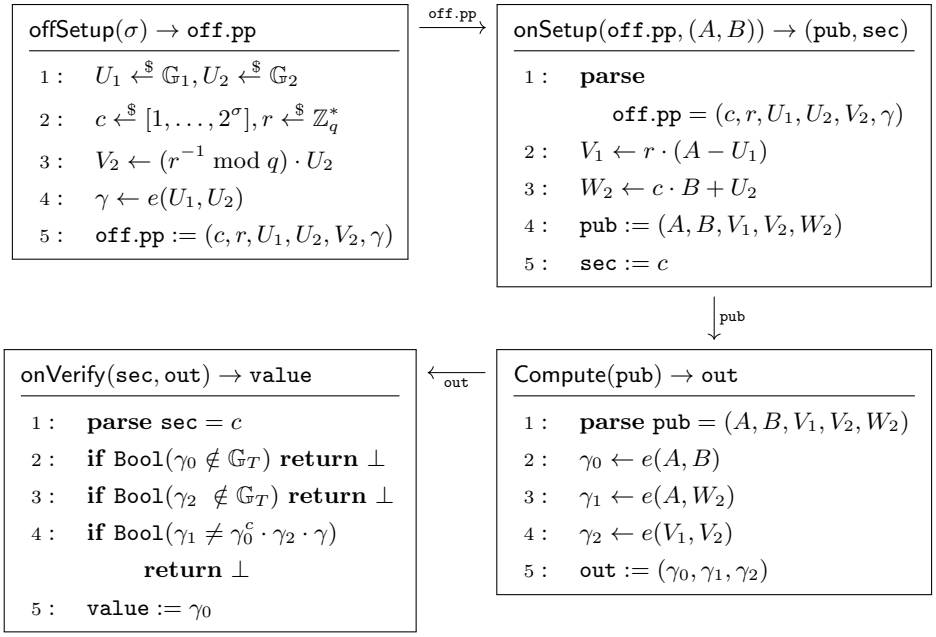

Fig. 2. Di Crescenzo et al.'s protocol for secure pairing delegation with online public inputs (see Section 3 in [21]). This description uses a different, more intuitive notation. The GlobalSetup is not included explicitely as it is trivial.

Correctness The correctness is trivial by inspection. By line 2 in Compute and line 5 in onVerify it follows that $\texttt{value} = \gamma_0 = e(A, B)$, since for correctness all parties are required not to deviate from the algorithms descriptions, and all communication happens via a perfect, noise-free channel.

Security The security essentially relies on the fact that an adversary (playing the role of a malicious server) cannot guess the challenge value c, except with probability $2^{-\sigma}$ (which is small by construction). We refer the reader to [21] for a detailed security proof.

Efficiency Regarding efficient online verification, we would need to estimate the client's computational cost in the online phase, i.e., $\mathsf{cost}(\mathsf{onSetup}) + \mathsf{cost}(\mathsf{onVerify})$ and compare it to the cost of computing the pairing $\mathsf{cost}(e(\cdot, \cdot))$. This is already done by [21] in an abstract way through a theoretical complexity analysis based on cost estimates extracted from Bos et al.'s work [16]. Interestingly, this efficiency analysis disregards the cost of membership testing in $\mathbb{G}_T$ which can be quite significant for some parameters [7]. In contrast, we aim to provide concrete efficiency analysis of complete algorithm executions (see Section 5). To this end, we implement the protocol in Figure 2, collect actual computational complexity and timings, and compare its performance against our LOVE variant (that we introduce in the next section, Figure 3).

4 Our Protocol for LOVE a Pairing

We first present our LOVE protocol in Figure 3, then we prove its security in the offline/online framework described in Section 3.1 and finally argue its online verification efficiency (deeper details on the latter in Section 5). Our LOVE protocol is obtained from few simple but clever twists on the original proposal of [21] presented in Figure 2. Concretely, LOVE's GlobalSetup, offSetup and onSetup are the same as in the previous proposal; the only changes are in Compute and onVerify, and we highlight them with a frame box in Figure 3.

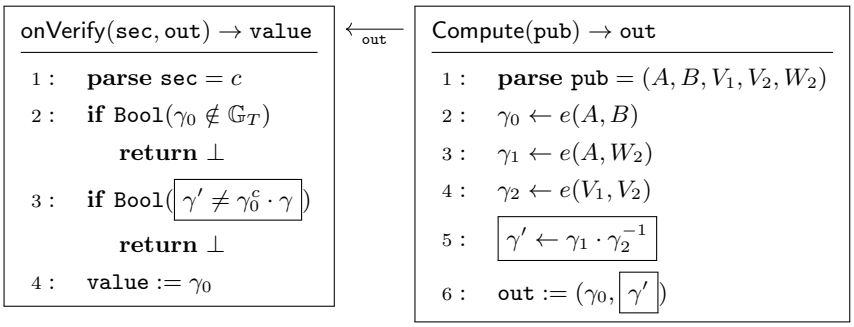

Fig. 3. LOVE: Lowering the cost of Outsourcing and Verifying Efficiently a pairing. The algorithms GlobalSetup, offSetup and onSetup are exactly as in Figure 2. For clarity, we frame the points in which LOVE differs from the previous proposal.

Correctness The correctness of our LOVE protocol (depicted in Figure 3) is evident by inspection: value $:= \gamma_0$ (line 4 in onVerify) and $\gamma_0 \leftarrow e(A, B)$ (line 2 in Compute).

Security The security proof for LOVE follows from the same arguments as the one for original protocol given in [21]. For completeness, we present below the full proof for our LOVE variant using the formalism of the offline/online framework introduced in Section 3.1.

In the security experiment $\mathbf{Exp}^{\mathsf{sec}}_{\mathcal{A}}(\lambda, \sigma)$, $\mathcal{A}$ chooses the pairing argument $(A, B) \in \mathbb{G}_1 \times \mathbb{G}_2$, and receives the string $\mathtt{pub} = (A, B, V_1, V_2, W_1)$. The adversary wins the game if she can forge an output $\mathtt{out}^* = (\gamma_0^*, \gamma'^*)$ on which onVerify returns $\mathtt{value} \notin \{\perp, e(A, B)\}$, i.e., the verification does not reject the forgery and returns a value different from the correct one.

Since we work with cyclic groups, each element has a unique representation as a multiple of a generator. For convenience let us describe the elements in pub in terms of their respective discrete logarithms (convention: lower case Latin letters denote the dlog of the corresponding capital case group element): $A = a \cdot P_1$, $B = b \cdot P_2$, $U_1 = u_1 \cdot P_1$, $V_1 = v_1 \cdot P_1$, $U_2 = u_2 \cdot P_2$, $V_2 = v_2 \cdot P_2$, $W_2 = w_2 \cdot P_2$. By construction we have:

$$\begin{cases} v_1 = ra - ru_1 \\ v_2 = r^{-1}u_2 \\ w_2 = cb + u_2 \end{cases} \tag{1}$$

where u_1, u_2 are uniform random variables (u.r.v.) on $\mathbb{Z}_q$, r is a u.r.v. on $\mathbb{Z}_q^*$, and c is a u.r.v. on $[1, \ldots, 2^\sigma]$. We make no assumptions on the distributions of a and b since these may be chosen by the adversary.

Our first step is to prove that pub leaks no information about c. We do so by showing that the distribution of (v_1, v_2, w_2), seen as the Cartesian product of the random variables obtained as the combination of (a, b, c, r, u_1, u_2) defined in System (1), is independent of the distribution of (a, b, c). Formally,

$$Prob[\{(v_1, v_2, w_2)\} | \{(a, b, c)\}] = Prob[\{(v_1, v_2, w_2)\}] + \mathsf{negl}(\lambda).$$

Proposition 1. *$Prob[\{(v_1, v_2, w_2)\}]$ is negligibly close to q^{-3} (u.r.v. on $\mathbb{Z}_q^3$).*

This is immediate since adding a u.r.v. defined on $\mathbb{Z}_q$ to any r.v. on any subset of $\mathbb{Z}_q$ yields a u.r.v. on $\mathbb{Z}_q$ (this is the argument for v_1 and w_2); and any r.v. on any subset of $\mathbb{Z}_q$ multiplied by a u.r.v. on $\mathbb{Z}_q$ yields a u.r.v. with overwhelming probability, i.e., except when either variable takes the value 0 (this is the argument for v_2). The latter event has probability q^{-1} which is negligible in the security parameter λ ($r \neq 0$ since it is invertible).

Our next goal is to show that the same statement holds even when conditioning the probability to a given event $\{(a, b, c)\} \in \mathbb{Z}_q \times \mathbb{Z}_q \times [1, \ldots, 2^\sigma]$.

Proposition 2. *Prob*$[\{(v_1, v_2, w_2)\}|\{(a, b, c)\}]$ *is negligibly close to* q^{-3}.

This is immediate for the same reasoning as Proposition 1. In detail, $w_2 = cb + u_2$ is uniformly distributed over $\mathbb{Z}_q$ since so is u_2, even conditioned to (b, c). Whenever $u_2 \neq 0$, $v_2 = r^{-1}u_2$ is uniformly distributed since so is r, and this holds independently of b, c and w_2. In detail, since we are in a prime order group, $u_2 \neq 0$ implies that u_2 admits an inverse $u_2^{-1} \mod q$. Therefore it is possible to write r as the product $r = r'(u_2^{-1} \mod q)$ for some $r' \in \mathbb{Z}_q^*$. Since r is uniformly random, so is r', and as a consequence also $v_2 = ru_2 = (r'(u_2^{-1} \mod q))u_2 \mod q = r'$. Finally, $v_1 = ra - ru_1$ is uniformly distributed since so is u_1 (and $r \neq 0$ since it is invertible by construction), and this holds independently of a, b, c and v_2, w_2.

Once established that pub leaks no information about c to $\mathcal{A}$ (except with a negligible probability in λ), we can move on to consider $\mathcal{A}$'s forgery attempts.

Proposition 3. *Given* $\gamma = e(U_1, U_2) \in \mathbb{G}_T$, *for any eligible forgery, i.e., for any* $(\gamma_0^*, \gamma'^*) \in \mathbb{G}_T^2$ *with* $\gamma_0^* \neq e(A, B)$, *there exists a unique value c, for which it holds that* $\gamma'^* = \gamma_0^{*c} \cdot \gamma$.

Because $\mathbb{G}_T$ is a multiplicative group, we can re-write the probabilistic verification check as $\gamma_0^{*c} = \gamma'^* \cdot \gamma^{-1}$. Since $\mathbb{G}_T$ is cyclic and of prime order, any element of $\mathbb{G}_T$ is a generator (except for its unit). Thus, $c = DLog_{\gamma_0^*}(\gamma_0^{*c}) = DLog_{\gamma_0^*}(\gamma'^* \cdot \gamma^{-1})$ is unique, modulus q (the group order).

Proposition 4. *For any* $\mathtt{out}^* = (\gamma_0{}^*, \gamma'^*) \in \mathbb{G}_T^2$ *such that* $\gamma_0{}^* \neq e(A, B)$, $\mathbf{Exp}_{\mathcal{A}}^{\mathsf{sec}}(\lambda, \sigma)$ *outputs 1 with probability at most* $2^{-\sigma} + \mathsf{negl}(\lambda)$.

By Propositions 1 and 2, the string pub does not leak any information about c. This implies that, for a malicious server, all values in $[1, \ldots, 2^\sigma]$ are still equally likely for c, even when conditioning over the $\mathcal{A}$'s view pub. By Proposition 3, the probability that any two values $(\gamma_0^*, \gamma'^*) \in \mathbb{G}_T$ satisfy the probabilistic test is one divided by the number of possible values c can take. Since to $\mathcal{A}$ all values of c are still equally likely, we get: $Prob[\mathbf{Exp}_{\mathcal{A}}^{\mathsf{sec}}(\lambda, \sigma) = 1] \leq 2^{-\sigma} + \mathsf{negl}(\lambda)$, which corresponds to $\mathcal{A}$ randomly guessing the value γ'^* that passes the verification equation (there are only 2^σ such values, given that $\gamma_0{}^*, \gamma \in \mathbb{G}_T$ and $c \in [1, \ldots 2^\sigma]$), or $\mathcal{A}$'s view leaking some information about c. □

Efficiency To show the efficient online verification of LOVE, we need to estimate the client's computational cost in the online phase, i.e., $\mathsf{cost}(\mathsf{onSetup}) + \mathsf{cost}(\mathsf{onVerify})$ and compare it to the cost of computing the pairing $\mathsf{cost}(e(\cdot, \cdot))$. The next section collects the actual computational complexity, timings and performance comparison against the original proposal of [21]. Here we provide only high-level arguments by counting the main operations of both protocols. Compared to the original protocol in Figure 2, the onVerify algorithm of LOVE saves one membership test for a $\mathbb{G}_T$ group element, and one multiplication in $\mathbb{G}_T$. Regarding communication, LOVE beats the original protocol by transmitting one less $\mathbb{G}_T$-element. Moreover, from the server side, LOVE's optimization also allows to compute γ' as a product of pairings and share the final exponentiation, which brings potential additional efficiency gains.

5 Implementation Results

We implemented LOVE and Di Crescenzo et al.'s protocol [21] using four different sets of parameters with the help of the RELIC cryptographic library [2]. The first choice is the legacy BN-254 curve previously used to set speed records [3] at the 128-bit security level, whose security guarantees have been degraded to a security level lying somewhere between 100 and 110 bits. The second choice is the curve BN-382, adjusted for new security levels. The third choice is BLS12-381 with embedding degree $k = 12$ and 255-bit prime-order subgroup popularized by the ZCash cryptocurrency [11]. The fourth choice is BLS12-383, a $\mathbb{G}_T$-strong curve generated by Scott [39,40] for applications where subgroup membership checking is

performance-critical.[4] The last choice is the BLS24-509 curve originally proposed by Costello [20] and recently suggested by Guillevic as promising at the 192-bit security [29]. RELIC provides dedicated Assembly acceleration for Intel 64-bit platforms for all these curves using a shared codebase, which means that finite field arithmetic is implemented using essentially the same techniques, which permits fair comparisons across different curves and protocols.[5] Given that our choices of λ range from 100 to 192, in order to improve protocol performance we selected a much lower statistical security level of $\sigma = 50$ bits in comparison to 128 used in [21].

	BN curves: $k = 12$	**BLS12 curves:** $k = 12$	**BLS12 curves:** $k = 12$
$p(z)$	$36z^4 + 36z^3 + 24z^2 + 6z + 1$	$(z-1)^2(z^4 - z^2 + 1)/3 + z$	$(z-1)^2(z^8 - z^4 + 1)/3 + z$
$q(z)$	$36z^4 + 36z^3 + 18z^2 + 6z + 1$	$z^4 - z^2 + 1$	$z^8 - z^4 + 1$
$t(z)$	$6z^2 + 1$	$z + 1$	$z + 1$
$h(z)$	1	$(z-1)^2/3$	$(z-1)^2/3$

E	b	z_0	$\lceil \log_2 p \rceil$	$\lceil \log_2 q \rceil$	$\lceil \log_2 h \rceil$
BN-254	2	$-(2^{62} + 2^{55} + 1)$	254	254	1
BN-382	2	$-(2^{94} + 2^{78} + 2^{67} + 2^{64} + 2^{48} + 1)$	382	382	1
BLS12-381	4	$-(2^{63} + 2^{62} + 2^{60} + 2^{57} + 2^{48} + 2^{16})$	381	255	126
BLS12-383	4	$2^{64} + 2^{51} + 2^{24} + 2^{12} + 2^9$	383	256	126
BLS24-509	1	$-2^{51} - 2^{28} + 2^{11} - 1$	509	408	100

Table 1. Parametrization and concrete parameters for the BN, BLS12 and BLS24 pairing-friendly curves used in our implementation. For the specified seed choice z_0, the curve BN-254 provides around 100 bits of security; and the curves BN-382, BLS12-381 and BLS12-383 provide a conjectured 128-bit security level. The curve BLS24-509 yields a conjectured security level of 192-bits.

Table 1 summarizes the main parameters corresponding to the BN [10], BLS12 and BLS24 [8] families of elliptic curves. Note that all of these curves are parameterized by an integer z, and they are defined by an equation of the form $Y^2 = X^3 + b$, and have a twist of degree $d = 6$. Table 1 also reports the salient parameters of the BN, BLS12 and BLS24 curve instantiations using a concrete choice of seed z_0, suitable for implementing pairing-based protocols at the 128- and 192-bit security level (this last security level is only achieved by the curve BLS24-509). The requirements for these security levels are in good agreement with the recommendations recently given in [29,30].

Membership Testing in $\mathbb{G}_T$. The traditional way of performing a subgroup membership test for a group element g, i.e., to explicitly verify whether or not $g \in \mathbb{G}_T$, is to exponentiate g by the group order q to check whether g^q is equal to the identity. An alternative way is first checking if g belongs to the cyclotomic subgroup of order $\phi_k(p)$. Thanks to the Frobenius endomorphism, this is an inexpensive operation (see below). If this test is passed, the second check consists of raising g to a power given by the cofactor $\phi_k(p)/q$, such that the final result lies in the right subgroup. For $\mathbb{G}_T$, the first strategy is usually more efficient because the cofactor $\phi_k(p)/q$ is typically considerably large, having, for the curve families considered in this paper, a bitlength at least three times larger than that of q.

For the specific case of prime-order BN curves, we know that $q = p + 1 - t$, so testing for membership can be done by checking that $g^q = g^{p+1-t} \stackrel{?}{=} 1_T$, or $g^p \stackrel{?}{=} g^{6z^2}$, which costs an efficient Frobenius map and an exponentiation by the short exponent $6z^2$ [39]. The exponentiation can be performed after checking that g is in the cyclotomic subgroup of order $p^4 - p^2 + 1$ through the equation $g \cdot g^{p^4} \stackrel{?}{=} g^{p^2}$, which only requires a few applications of powers of the Frobenius and one multiplication. In the cyclotomic subgroup, faster [27] and compressed squarings [3] are available and are favored due to the low Hamming weight of the exponent.

[4] Following the definition given in [39], a curve is said to be $\mathbb{G}_T$-strong, if $\phi_k(p)/q$ does not have small factors, where $\phi_k(\cdot)$ denotes the k-th cyclotomic polynomial, p is the base field prime and q is the order of the group $\mathbb{G}_T$, respectively.

[5] The resulting code is available in the library repository for reproducibility.

The case for BLS12 curves is split into the two options, but we start by checking for cyclotomic subgroup membership in both. The BLS12-383 curve is $\mathbb{G}_T$-strong, so further checks can be omitted. For BLS12-381, the situation is more complicated, as the cofactor is known to be composite but hard to factor. A conservative way involves exploiting $g^q = g^{(p+1-t)/h} \stackrel{?}{=} 1_T$ to check $g^p \stackrel{?}{=} g^z$ and $g^h \neq 1_T$, as implemented in the MIRACL library[6]. A faster way consists of following the recommendation in [7] to perform the exponentiation by the group order with the 4-GLS method using the Frobenius as an efficient endomorphism in $\mathbb{G}_T$ [24]. The 4-dimensional decomposition is fixed and sparse for the group order, such that the exponentiation requires only an exponentiation by sparse z, two multiplications and two applications of the Frobenius.

For the BLS24-509 curve, we first check for membership in the cyclotomic subgroup of order p^8-p^4+1 and then proceed with the same conservative and fast strategies as in the BLS12 curve, namely, exploiting the group order equation or optimizing the 8-GLS exponentiation. The latter approach only involves an exponentiation by z followed by four multiplications and four Frobenius.

5.1 Timings for Operations in Pairing Groups

We implemented the conservative and fast membership testing in $\mathbb{G}_T$ as described in the previous section, and benchmarked the other pairings group operations on a high-end Intel Core i7-6700K Skylake processor running at 4.0GHz, with HyperThreading (HT) and TurboBoost (TB) turned off to reduce measurement noise. RELIC was built for each curve using the available configuration presets with GCC 11.0.1 on a Fedora 34 operating system.

The target platform is obviously not representative of an embedded system, but to keep comparisons fair we do not make usage of any memory-heavy operation that would benefit either the pairing computation or the additional protocol operations in one platform or another. In particular, the protocols we implemented do not require fixed-base scalar multiplications or exponentiations that could benefit from large precomputed tables in any of the groups.

Timings can be found in Table 2, for scalar multiplication in the unknown point case for $\mathbb{G}_1$ and $\mathbb{G}_2$ using endomorphisms and a left-to-right w-NAF algorithm with $w = 4$. Exponentiation of a variable base in $\mathbb{G}_T$ does not rely on precomputation and uses cyclotomic squarings and GLS endomorphisms with a simple NAF algorithm, since inversion in a cyclotomic subgroup is just conjugation. We also include timings for operations with short scalar/exponents using a simple NAF approach to show savings for shorter 50-bit challenges. We hope these results can update the figures from [16] with current parameters, and note that the rate at which the cost of performing operations increases from $\mathbb{G}_1$ to $\mathbb{G}_2$, and to $\mathbb{G}_T$ is lower than [16], indicating that we employ a more efficient implementation of extension field arithmetic.

Operation \ **Curve**	**BN-254**	**BN-382**	**BLS12-381**	**BLS12-383**	**BLS24-509**
$[r]P$ in $\mathbb{G}_1$	214	587	402	404	969
$[c]P$ in $\mathbb{G}_1$, short c	72	133	134	134	210
$[r]Q$ in $\mathbb{G}_2$	381	1268	836	879	5231
$[c]Q$ in $\mathbb{G}_2$, short c	139	305	322	322	1631
g^r in $\mathbb{G}_T$	601	1952	1317	1318	8323
g^c in $\mathbb{G}_T$, short c	282	633	634	634	2487
Cons. Test in $\mathbb{G}_T$	262	895	683	–	2483
Fast test in $\mathbb{G}_T$	–	–	382	–	1660
$e(P, Q)$	1086	3664	3255	3187	16730
Miller Loop	641	2183	1469	1446	5924
Final Exp	445	1481	1786	1741	10806

Table 2. Timings of pairing group operations implemented in RELIC reported in 10^3 cycles in a Skylake processor, averaged over 10^4 executions (HT and TB disabled). The operations are scalar multiplication or exponentiation by a random integer $r \xleftarrow{\$} \mathbb{Z}_q^*$ or a short 50-bit scalar c, and membership testing in $\mathbb{G}_T$ (both conservative and fast variants). The pairing computation is split between Miller loop and Final exponentiation.

[6] `https://github.com/miracl/MIRACL/`

5.2 Timings for Delegated Pairing Computation

We implemented the original protocol due to Di Crescenzo et al. and our LOVE variant in the same benchmarking machine, and collected the timings in Table 3. We implemented both the public and the private input versions for completeness, and adopted the fast membership check for a best-case scenario. The protocol operations include preco (corresponding to the client's offSetup), the server-side portion of the computation server (Compute) and client-side online algorithms client (onSetup and onVerify). We first note that the offline setup of both protocols is the same, so no significant performance difference is observed in that step. Compared to [21], LOVE has significant improvements for the client in all curves, except for BLS12-383 because the main savings come from skipping one subgroup membership checking. In the public inputs case, the LOVE's improvements range from 18.0% to 29.7%; while in the private inputs case, they decrease to around 10.3% and 14.9%. From the server's point of view, the savings are between 20.2% and 24.9% for public inputs; and 15.1% to 18.7% for private inputs. These extra savings come from interleaving products of pairings inside Compute for the LOVE protocol. We do not take the communication latency in consideration for our performance estimates, but a simple analysis of how many bytes are transmitted points out that LOVE saves 23-24% communication cost depending on the choice of parameters by reducing by one the number of $\mathbb{G}_T$ elements transmitted.

Now considering the cost of computing a pairing, we observe performance improvements of LOVE in comparison with local computation ranging from 15.7% to 56.2%. The speedup is higher for the curves BLS12-383 and BLS24-509 because of the $\mathbb{G}_T$-strong property. LOVE provides speedups even in the BN-254 and BN-382 curves, where [21] underperforms. Neither protocol is efficient in the private inputs case. The significantly lower performance of Di Crescenzo et al.'s protocol, even in the favourable setting when $\sigma = 50$ reduces the impact of $\mathbb{G}_T$ exponentiations, directly contradicts the estimates given in [21]. We attribute this effect to the lack of membership checks in the performance estimates and an inaccurate extrapolation from [16] to new security levels.

Protocol \ **Curve**	**BN-254**	**BN-382**	**BLS12-381**	**BLS12-383**	**BLS24-509**
cost$(e(P,Q))$	1086	3664	3255	3187	16730
[21] (preco)	2055	6520	5207	5225	27659
[21] (client)	1183	**3459** (5.6%)	**2167** (33.4%)	**1472** (53.8%)	**8928** (46.6%)
[21] (server)	3284	11070	9889	9710	50363
LOVE (preco)	2050	6516	5199	5217	27657
LOVE (client)	**916** (15.7%)	**2433** (33.6%)	**1768** (45.7%)	**1397** (56.1%)	**7322** (56.2%)
LOVE (server)	2595	8829	7600	7442	37800
Priv-[21] (preco)	4892	15607	12100	12219	65852
Priv-[21] (client)	2452	7071	4406	3358	18459
Priv-[21] (server)	4404	14800	13237	12991	67179
Priv-LOVE (preco)	4892	15619	12090	12219	65845
Priv-LOVE (client)	2130	6017	3953	3304	**16298** (2.6%)
Priv-LOVE (server)	3704	12560	10887	10701	54591

Table 3. Timings from running the pairing delegation protocols implemented in RELIC reported in 10^3 cycles in a Skylake processor, averaged over 10^4 executions (HT and TB disabled). For all protocols the statistical security parameter is set to $\sigma = 50$. The label preco refers to the offline precomputation (offSetup), client to the client-side online computation (onSetup and onVerify), and server to server-side online computation (Compute). We mark in bold the combination of parameter and setting that provides a performance improvement over computing the pairing locally. In these cases, we display between parenthesis the corresponding efficiency gain computed as $(1 - \mathsf{cost}(\mathsf{client})/\mathsf{cost}(e(P,Q))$. Higher percentage values imply larger efficiency gains.

6 Conclusions

In this paper, we introduced LOVE: the most efficient protocol to date for secure offline/online delegation of a pairing computation. While developing and analyzing LOVE we identified interesting questions that stem out of our research.

For instance, is there a secure way to leverage the first pairing delegation to efficiency advantage of delegating one-more pairing? In other words, can 'batch delegation' of n pairings be secure and more efficient than just repeating LOVE n times? An orthogonal direction would be to investigate if one can securely delegate other building blocks of the verification, such as hash-to-point or membership tests.

Also, protocol-tailored solutions might be interesting. For instance, in the context of Groth's zk-SNARK [28], the verifier needs to compute l scalar multiplications in $\mathbb{G}_1$, 3 executions of the Miller's loop and 1 computation of the final exponentiation (here l is a parameter of the zk-SNARK protocol). In this setting, can we design a secure and efficient delegation protocol for the computation of the three Miller loops and the final exponentiation? These are all components needed for the computation of a pairing, but we are not aware of works that outsource these components, instead of the whole pairing.

Finally, we identified the need for efficient and reliable $\mathbb{G}_T$-membership testing. Since the BLS12-381 curve is being considered for standardization[7], we suggest starting a computational effort to find out the integer factorization of the $\mathbb{G}_T$ cofactor of this curve or bounds on its prime factors to better understand its subgroup security.

Acknowledgments. This work was partly funded by: the strategic research area ELLIIT. The first author is also affiliated to the DIGIT Centre for Digitalisation, Big Data and Data Analytics; and the Concordium Blockchain Research Center at Aarhus University.

References

1. D. F. Aranha, P. S. L. M. Barreto, P. Longa, and J. E. Ricardini. The realm of the pairings. In *SAC*, volume 8282 of *LNCS*, pages 3–25. Springer, 2013.
2. D. F. Aranha, C. P. L. Gouvêa, T. Markmann, R. S. Wahby, and K. Liao. RELIC is an Efficient LIbrary for Cryptography. `https://github.com/relic-toolkit/relic`.
3. D. F. Aranha, K. Karabina, P. Longa, C. H. Gebotys, and J. C. López-Hernández. Faster explicit formulas for computing pairings over ordinary curves. In *EUROCRYPT*, volume 6632 of *LNCS*, pages 48–68. Springer, 2011.
4. D. F. Aranha and E. Pagnin. The simplest multi-key linearly homomorphic signature scheme. In *LATINCRYPT*, volume 11774 of *LNCS*, pages 280–300. Springer, 2019.
5. D. F. Aranha, E. Pagnin, and F. Rodríguez-Henríquez. Love a pairing. In *LATINCRYPT*, LNCS. Springer, 2021.
6. R. Barbulescu and S. Duquesne. Updating key size estimations for pairings. *J. Cryptol.*, 32(4):1298–1336, 2019.
7. P. S. L. M. Barreto, C. Costello, R. Misoczki, M. Naehrig, G. C. C. F. Pereira, and G. Zanon. Subgroup security in pairing-based cryptography. In *LATINCRYPT*, volume 9230 of *LNCS*, pages 245–265. Springer, 2015.
8. P. S. L. M. Barreto, B. Lynn, and M. Scott. Constructing elliptic curves with prescribed embedding degrees. In *SCN*, volume 2576 of *LNCS*, pages 257–267. Springer, 2002.
9. P. S. L. M. Barreto, B. Lynn, and M. Scott. Efficient implementation of pairing-based cryptosystems. *J. Cryptol.*, 17(4):321–334, 2004.
10. P. S. L. M. Barreto and M. Naehrig. Pairing-friendly elliptic curves of prime order. In *SAC*, volume 3897 of *LNCS*, pages 319–331. Springer, 2005.
11. E. Ben-Sasson, A. Chiesa, C. Garman, M. Green, I. Miers, E. Tromer, and M. Virza. Zerocash: Decentralized anonymous payments from bitcoin. In *IEEE Symposium on Security and Privacy*, pages 459–474. IEEE Computer Society, 2014.
12. D. Boneh, J. Bonneau, B. Bünz, and B. Fisch. Verifiable delay functions. In *CRYPTO (1)*, volume 10991 of *LNCS*, pages 757–788. Springer, 2018.
13. D. Boneh and M. K. Franklin. Identity-Based Encryption from the Weil Pairing. In *CRYPTO*, volume 2139 of *LNCS*, pages 213–229. Springer, 2001.
14. D. Boneh, C. Gentry, B. Lynn, and H. Shacham. Aggregate and verifiably encrypted signatures from bilinear maps. In *EUROCRYPT*, volume 2656 of *LNCS*, pages 416–432. Springer, 2003.
15. D. Boneh, B. Lynn, and H. Shacham. Short Signatures from the Weil Pairing. *J. Cryptol.*, 17(4):297–319, 2004.
16. J. W. Bos, C. Costello, and M. Naehrig. Exponentiating in pairing groups. In *SAC*, volume 8282 of *LNCS*, pages 438–455. Springer, 2013.

[7] `https://datatracker.ietf.org/doc/html/draft-irtf-cfrg-bls-signature-04`

17. S. Canard, J. Devigne, and O. Sanders. Delegating a pairing can be both secure and efficient. In *ACNS*, volume 8479 of *LNCS*, pages 549–565. Springer, 2014.
18. B. Chevallier-Mames, J. Coron, N. McCullagh, D. Naccache, and M. Scott. Secure delegation of elliptic-curve pairing. In *CARDIS*, volume 6035 of *LNCS*, pages 24–35. Springer, 2010.
19. S. S. Chow, M. H. Au, and W. Susilo. Server-aided signatures verification secure against collusion attack. *Information Security Technical Report*, 17(3):46–57, 2013.
20. C. Costello, K. E. Lauter, and M. Naehrig. Attractive subfamilies of BLS curves for implementing high-security pairings. In *INDOCRYPT*, volume 7107 of *LNCS*, pages 320–342. Springer, 2011.
21. G. D. Crescenzo, M. Khodjaeva, D. Kahrobaei, and V. Shpilrain. Secure and efficient delegation of pairings with online inputs. In *CARDIS*, volume 12609 of *LNCS*, pages 84–99. Springer, 2020.
22. L. D. Feo, S. Masson, C. Petit, and A. Sanso. Verifiable delay functions from supersingular isogenies and pairings. In *ASIACRYPT (1)*, volume 11921 of *LNCS*, pages 248–277. Springer, 2019.
23. D. Freeman, M. Scott, and E. Teske. A taxonomy of pairing-friendly elliptic curves. *J. Cryptol.*, 23(2):224–280, 2010.
24. S. D. Galbraith and M. Scott. Exponentiation in pairing-friendly groups using homomorphisms. In *Pairing*, volume 5209 of *LNCS*, pages 211–224. Springer, 2008.
25. R. Gennaro, C. Gentry, and B. Parno. Non-interactive verifiable computing: Outsourcing computation to untrusted workers. In *CRYPTO*, volume 6223 of *Lecture Notes in Computer Science*, pages 465–482. Springer, 2010.
26. M. Girault and D. Lefranc. Server-aided verification: Theory and practice. In *ASIACRYPT*, volume 3788 of *LNCS*, pages 605–623. Springer, 2005.
27. R. Granger and M. Scott. Faster squaring in the cyclotomic subgroup of sixth degree extensions. In *PKC*, volume 6056 of *LNCS*, pages 209–223. Springer, 2010.
28. J. Groth. On the size of pairing-based non-interactive arguments. In *EUROCRYPT (2)*, volume 9666 of *LNCS*, pages 305–326. Springer, 2016.
29. A. Guillevic. A short-list of pairing-friendly curves resistant to special TNFS at the 128-bit security level. In *PKC (2)*, volume 12111 of *LNCS*, pages 535–564. Springer, 2020.
30. A. Guillevic, S. Masson, and E. Thomé. Cocks-pinch curves of embedding degrees five to eight and optimal ate pairing computation. *Des. Codes Cryptogr.*, 88(6):1047–1081, 2020.
31. A. Guillevic and D. Vergnaud. Algorithms for outsourcing pairing computation. In *CARDIS*, volume 8968 of *LNCS*, pages 193–211. Springer, 2014.
32. B. G. Kang, M. S. Lee, and J. H. Park. Efficient delegation of pairing computation. *IACR Cryptol. ePrint Arch.*, 2005:259, 2005.
33. T. Kim and R. Barbulescu. Extended tower number field sieve: A new complexity for the medium prime case. In *CRYPTO (1)*, volume 9814 of *LNCS*, pages 543–571. Springer, 2016.
34. N. M. Mbang, D. F. Aranha, and E. Fouotsa. Computing the Optimal Ate pairing over elliptic curves with embedding degrees 54 and 48 at the 256-bit security level. *Int. J. Appl. Cryptogr.*, 4(1):45–59, 2020.
35. T. Mefenza and D. Vergnaud. Verifiable outsourcing of pairing computations. Technical report.
36. V. S. Miller. The Weil Pairing, and Its Efficient Calculation. *J. Cryptol.*, 17(4):235–261, 2004.
37. N. E. Mrabet and M. Joye, editors. *Guide to Pairing-Based Cryptography.* Chapman and Hall/CRC, London, 2017.
38. E. Pagnin, A. Mitrokotsa, and K. Tanaka. Anonymous single-round server-aided verification. In *LATINCRYPT*, volume 11368 of *LNCS*, pages 23–43. Springer, 2017.
39. M. Scott. Unbalancing pairing-based key exchange protocols. Cryptology ePrint Archive, Report 2013/688, 2013. `http://eprint.iacr.org/`.
40. M. Scott. Pairing implementation revisited. Cryptology ePrint Archive, Report 2019/077, 2019. `https://eprint.iacr.org/2019/077`.
41. P. P. Tsang, S. S. M. Chow, and S. W. Smith. Batch pairing delegation. In *IWSEC*, volume 4752 of *LNCS*, pages 74–90. Springer, 2007.
42. F. Vercauteren. Optimal pairings. *IEEE Trans. Inf. Theory*, 56(1):455–461, 2010.
43. Z. Wang. A new construction of the server-aided verification signature scheme. *Mathematical and computer modelling*, 55(1-2):97–101, 2012.
44. W. Wu, Y. Mu, W. Susilo, and X. Huang. Server-aided verification signatures: Definitions and new constructions. In *ProvSec*, volume 5324 of *LNCS*, pages 141–155. Springer, 2008.
45. W. Wu, Y. Mu, W. Susilo, and X. Huang. Provably secure server-aided verification signatures. *Computers & Mathematics with Applications*, 61(7):1705–1723, 2011.
46. G. Zanon, M. A. S. Jr., G. C. C. F. Pereira, J. Doliskani, and P. S. L. M. Barreto. Faster key compression for isogeny-based cryptosystems. *IEEE Trans. Computers*, 68(5):688–701, 2019.
47. E. Zavattoni, L. J. D. Perez, S. Mitsunari, A. H. Sánchez-Ramírez, T. Teruya, and F. Rodríguez-Henríquez. Software implementation of an attribute-based encryption scheme. *IEEE Trans. Computers*, 64(5):1429–1441, 2015.