That's AmorE: Amortized Efficiency for Pairing Delegation

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

[1] Chalmers University of Technology and University of Gothenburg, Gothenburg, Sweden
{keilty,elenap}@chalmers.se
[2] Aarhus University, Aarhus, Denmark dfaranha@cs.au.dk
[3] Cryptography Research Centre, Technology Innovation Institute, UAE Computer Science Department, Cinvestav, México francisco@cs.cinvestav.mx

Abstract. Over two decades since their introduction in 2005, all major verifiable pairing delegation protocols for public inputs have been designed to ensure information-theoretic security. However, we note that a delegation protocol involving only ephemeral secret keys in the public view can achieve everlasting security, provided the server is unable to produce a pairing forgery *within* the protocol's execution time. Thus, computationally bounding the adversary's capabilities during the protocol's execution, rather than across its entire lifespan, may be more reasonable, especially when the goal is to achieve significant efficiency gains for the delegating party. This consideration is particularly relevant given the continuously evolving computational costs associated with pairing computations and their ancillary blocks, which creates an ever-changing landscape for what constitutes efficiency in pairing delegation protocols. With the goal of fulfilling both efficiency and everlasting security, we present AmorE, a protocol equipped with an adjustable security and efficiency parameter for sequential pairing delegation, which achieves state-of-the-art *amortized efficiency* in terms of the number of pairing computations. For example, delegating batches of 10 pairings on the BLS48-575 elliptic curve via our protocol costs to the client, on average, less than a single scalar multiplication in $\mathbb{G}_2$ per delegated pairing, while still ensuring at least 40 bits of statistical security.

1 Introduction

A cryptographic pairing is a function e that maps a pair $(A, B) \in \mathbb{G}_1 \times \mathbb{G}_2$ into an element $e(A, B) \in \mathbb{G}_T$. Here, $\mathbb{G}_1$, $\mathbb{G}_2$, and $\mathbb{G}_T$ represent groups of prime order q. The map e enjoys special properties such as being bilinear, non-degenerate, and efficiently computable. Constructive applications of bilinear pairings for cryptography appeared about 25 years ago by Sakai, Ohgishi, and Kasahara [51] and Joux [36]. Since then, bilinear pairings have proven to be remarkable in cryptography, driving the development of efficient solutions and enabling previously infeasible constructions, including three-party key agreement [36], the first cryptographically secure identity-based encryption scheme by Boneh and Franklin [12], aggregate signatures [13], short signatures [14], attribute-based encryption [50], and constant-size polynomial commitments [41]. Cryptocurrencies and blockchain technologies have caused a new surge in the development of pairing-based zero-knowledge Succinct Non-interactive ARguments of Knowledge (zk-SNARKs), as they provide exceptionally compact (short) proofs, e.g., [49, 30, 41, 31, 23]. Other recent applications include compact multi-signatures [11]; zk-SNARK aggregation [27] and threshold signatures with silent setup [28].

Pairing-based protocols typically rely on several ancillary building blocks, including membership tests, exponentiation of group elements in $\mathbb{G}_T$, scalar multiplication on the elliptic curves of $\mathbb{G}_1$ and $\mathbb{G}_2$, and the hash-to-point primitive, among others. To date, and depending on the pairing-friendly elliptic curve selected, computing a single pairing can be between eight to forty-seven times more expensive than performing scalar multiplications in $\mathbb{G}_1$ and about four times as expensive as scalar multiplications in $\mathbb{G}_2$ (cf. Table 2). Additionally, specialized pairing arithmetic libraries can considerably increase the overall software footprint of a cryptographic library.

Pairing delegation protocols offer a way to overcome this barrier [29, 16, 33, 22, 3]. In its simplest form, a pairing delegation protocol is a single round two-party interactive protocol that empowers an entity, referred to as client, to offload the evaluation of one or more pairings to a computationally more powerful but untrusted entity, the server. The interaction should be designed in such a way that the client can efficiently verify the correctness of the result. In contexts that require the computation of several pairings (referred

to as a *batch*), such as verifying BLS signatures or SNARKs, single pairing delegation is replaced by batch pairing delegation protocols [56, 45, 21, 47] with the goal of improving efficiency.

Table 1 offers a brief comparison between our work and the most relevant protocols in the same setting, i.e., *two-party* (client-server) protocols for delegating pairings computed on *public inputs* (i.e., the pairing arguments A, B in $e(A, B)$ are known to and possibly chosen by the adversary), where the server is untrusted (*malicious*). The top rows in Table 1 present single-pairing delegation protocols, while the bottom rows present batch delegation protocols. Negative gains indicate inefficient constructions for the client (the total client cost of running the delegation protocol is higher than computing the delegated operation locally).

Protocol	**Lower bounds on client cost**	**Gain**	**Security**
CDS14[16, Sec. 4.1]	$2N(\mathfrak{m}_1 + \mathfrak{m}_2 + \mathfrak{m}_T + \in_T)$	-40.1%	unc
CKKS20 [22]	$N(\mathfrak{p} + \mathfrak{m}_1 + \mathfrak{m}_2 + \bar{\mathfrak{m}}_2 + \bar{\mathfrak{m}}_T + 2 \in_T)$	-101.1%	unc
LOVE [3]	$N(\mathfrak{p} + \mathfrak{m}_1 + \mathfrak{m}_2 + \bar{\mathfrak{m}}_2 + \bar{\mathfrak{m}}_T + \in_T)$	-89.8%	unc
KC23 [42]	-	-	✗
AmorE (Fig. 2, $M = 1$)	$\mathfrak{m}_T + N(2\mathfrak{m}_1 + \mathfrak{m}_2 + \bar{\mathfrak{m}}_2 + \bar{\mathfrak{m}}_T + \in_T)$	32.1%	evr
MV19 [45, Alg. 4]	$N(\mathfrak{p} + M(\mathfrak{m}_1 + \mathfrak{m}_2 + \bar{\mathfrak{m}}_2 + \bar{\mathfrak{m}}_T + 2 \in_T))$	-3.7%	unc
CKC23 [21, Sec. 3.2]	$N(\mathfrak{p} + M(\mathfrak{m}_1 + \bar{\mathfrak{m}}_1 + \mathfrak{m}_2 + \bar{\mathfrak{m}}_T + \in_T))$	-6.5%	unc
KST23 [38, Alg. 1]	-	-	✗
AmorE (Fig. 2, $M > 1$)	$\mathfrak{m}_T + N(\mathfrak{m}_1 + 2\mathfrak{m}_2 + M(\bar{\mathfrak{m}}_1 + \bar{\mathfrak{m}}_T + \in_T))$	54.7%	evr

Table 1: Comparison of recent single- (top part) and batch (lower part) pairing delegation protocols for public and variable input in the single untrusted server model. The third column specifies the client's concrete efficiency gain compared to locally computing $N = 10$ delegations of a single pairing, or of batches of size $M = 3$ pairings on BLS12-381 (see Tables 3, 4 and 6 for further details). Notation: N = number of delegations; M = size of the batch; $\mathfrak{p}$ = pairing; $\mathfrak{m}_\iota(\bar{\mathfrak{m}}_\iota)$ = (short) scalar multiplication in $\mathbb{G}_\iota$; $\in_T$= membership test in $\mathbb{G}_T$; unc = unconditional; evr = everlasting; ✗ = proven broken in this paper.

All protocols in Table 1 rely on the fact that the client employs secret randomness to (statistically) verify the correctness of the result returned by the server. If implemented correctly, this approach makes a *one-shot* delegation information theoretically (aka unconditionally) sound, as discussed in, e.g., [20, 56]. By one-shot, we mean that no randomness can be re-used for securely delegating the computation of a second pairing, or a new batch of pairings. Simultaneously achieving security and efficiency is tricky, even for one-shot protocols. Two recent proposals (KC23 [42] for single, and KST23 [38] for batch) do not require the client to compute any element in the target group, if not as a combination of what is returned by the server. While this design choice is attractive for efficiency, we observe that it inevitably leads to verification equations that are malleable, and renders the protocols unsound. We generalize and formalize this observation in an impossibility result (Theorem 1).

Introducing Sequential Pairing Delegations and Amortized Efficiency

We argue that no prior work on public-input pairing delegation in the single untrusted server model is simultaneously secure and (significantly) efficient for the client: the client is required to compute either one pairing, e.g., [3, 22, 56, 45, 21], or multiple full-range exponentiations in the target group (each roughly as expensive as half a pairing), e.g., [20]; or the protocol is insecure, e.g., [42, 38]. Efficiency gains only happen when disregarding so-called *offline* client costs, e.g., CKKS20 [22] and LOVE [3] (see Table 3); or for *larger batch sizes*, e.g., MV19 [45], CKC23 [21] (see Table 4). This motivates the following question:

> *Can we design two-party delegation protocols for pairings with public variable input, that are secure against malicious adversaries and offer significant computational savings to the delegating party?*

This work answers this question affirmatively. We consider efficiency in the amortized sense: leveraging a one-time setup to *sequentially* delegate the computation of multiple (single or batch) pairings. This approach

provides a much-needed unified framework for two –otherwise artificially separated– lines of work (single vs. batch pairing delegation); and paves the way for pairing delegation protocols that are both secure and concretely efficient for the client: the client's computational cost of the one-time setup is *amortized* over several sequential delegations.

We notice that unconditional security seems to inherently hinder client efficiency. This is apparent in Table 1, and also motivated by the fact that, to argue information theoretic security, the client needs to sample some secrets from the full-range (e.g., $\mathbb{Z}_q^*$ or $\mathbb{G}_1$). Table 2 shows that a full-range exponentiation in the target group ($\mathfrak{m}_T$) is roughly as costly as half of a pairing ($\mathfrak{p}$), while a single full-range scalar multiplication in each of the base groups ($\mathfrak{m}_1 + \mathfrak{m}_2$) is approximately equivalent to one fourth of a pairing. To achieve significant computational savings for the client, we consider *everlasting security* (introduced in [4, 25]) rather than unconditional. Intuitively, this allows us to prove security against an adversary that is (time/computationally) bounded only during the delegation process, and may become unbounded after the protocol execution is completed. This setting is a relaxation of the hybrid bounded storage model introduced by Harnik and Naor in [34], as in our case, the adversary is allowed to store as much as it can compute.

Application Scenarios: Candidate Clients. Secure and efficient pairing delegation is well-suited for non-time-critical applications in which client devices have reliable communication channels with the server. Examples include IoT devices, constrained Electronic Control Units (ECUs) connected via CAN bus to a computationally more powerful unit in a car [54], and USB-C hardware wallets.

IoT devices are typically constrained by battery, processing power, and rely on communication protocols such as Wi-Fi, Bluetooth, and Zigbee. For devices that lack sufficient memory to store pairing libraries and/or the processing power to handle pairing computations, and where communication latency is not a critical concern, AmorE can offer significant advantages.

Hardware wallets represent another class of suitable clients. These devices are designed to securely store cryptocurrency private keys offline and usually connect to a computer or mobile device via a USB port, to sign transactions. Many of these wallets support anonymity-preserving cryptocurrencies like Zcash, which require computationally intensive zk-SNARK verifications. For instance, the popular Ledger Nano S Plus, as of 2025, uses an ST33K1M5 chip with a Cortex-M35P processor. Due to the inherent high computational cost of pairings, this processor may not be suitable for performing pairing computations locally.

Lastly, an interesting application scenario involves delegating pairing computations to a dedicated chip designed by an untrusted manufacturer. In this context, Campanelli and Gennaro [15] highlighted that defending against hardware-limited adversaries does not require the full scope of cryptographic protection. Instead, reasonable assumptions can be made about the computational power available in a given chip area, which can be used to design efficient solutions.

Application Scenarios: Pairing Provenance. Several pairing-based cryptographic schemes require the computation of a single pairing for each message decryption, e.g., the ID-based encryption scheme by Boneh and Franklin [12]; the public key encryption with keyword search [10]; Joux-based multi-party key agreement [36, 7]; identity-based onion routing [40]; trusted computing [56]; and, last but not least, the simpler-inner product argument in aPlonk [1]. In the aforementioned cases, the pairing arguments are progressively disclosed, making sequential (single) pairing delegation protocols a perfect candidate to widen the range of devices that can handle the required, but costly, pairing computation.

The vast majority of pairing-based schemes rely on products[4] of several pairings. With a cheap and fast connection to an untrusted server, sequential (batch) pairing delegation protocols can enable IoT devices, hardware wallets and constrained ECUs to efficiently handle complex pairing-based cryptosystems such as: BLS short signatures ($2\mathfrak{p}$) [14]; Groth16 ($3\mathfrak{p}$) [31]; aggregate signatures [13]; polynomial Commitments [41, 1]; SNARKs ($21\mathfrak{p}$) [8]; SnarkPack [27]; and performing secret leader election systems [17].

[4] The optimization of product of pairings where the whole computation requires the calculation of a single final exponentiation, is a well-known trick since the 2010's [52, 59].

1.1 Our Contributions

In this paper we present several improvements and novel ideas in the subject of pairing delegation. In summary, the main contributions are:

1. **Impossibility result**. We identify a fatal security flaw in the single pairing delegation protocol by Khodjaeva-di-Crescenzo [42] and in the batch pairing delegation protocol by Kalkar-Sertkaya-Tutdere [38]. This unfortunate discovery motivated us to distillate the cause of the attack, and led us to prove an *impossibility result* (Theorem 1) that serves as an indication to the hardness of achieving secure pairing delegation in the single untrusted server setting, if the client does not perform any preprocessing step in the target group.
2. **A Framework for Sequential Pairing Delegation**. As a minor contribution, we introduce a syntax that captures single, batch, one-shot, and sequential pairing delegation protocols. This syntax can be used to express all existing proposals in a unified way. Motivated by the limited efficiency gains of existing proposals, we set out to investigate how to amortize a one-time cost across several sequential delegations. To formally reason about this new setting, we introduce a security model for sequential delegation of pairing with public variable inputs in the single untrusted server model, and a definition of amortized security. We hope this framework will serve as reference for future work in this field.
3. **A Protocol for Sequential Pairing Delegation with Amortized Efficiency**. We propose AmorE (Figure 2) the first protocol in the sequential pairing delegation framework. AmorE has an efficiency parameter φ that enables the client to tweak the desired security level. Setting $\varphi = 2\lambda$, where λ is the computational security parameter for the bilinear group, yields an unconditionally secure scheme. This sets all client's randomness to be full-range. This design choice, however, does not lead to a significantly efficient solution, as discussed previously. We then present a specific setting of φ that renders AmorE concretely (amortized) efficient and provably everlasting secure in the hybrid bounded storage model, against an algebraic adversary whose computational power does not exceed the hashrate of the entire Bitcoin network. Notably, AmorE does not require the client to compute a pairing locally, removing the need for storing/loading specialized libraries for this task.
4. **A Novel Proof Technique**. As a result of independent interest we provide a novel proof technique for everlasting security of a pairing delegation protocol. To the best of our knowledge this is the first time that everlasting security is considered in this line of work. Interestingly, our proof draws connection to the original Diffie and Hellman meet-in-the-middle attack and combines known mathematical results that range from combinatorics to mathematical optimization/programming.
5. **Experimental Validation and Efficient Short Scalar Sampling**. We present benchmarking results using the cryptographic library RELIC and introduce a new technique for sampling and using short scalars more efficiently than previous work. Through careful analysis of various trade-offs and design choices, we have developed a software library capable of executing all the protocols presented in this paper with high computational time efficiency. Our software is freely available at: `https://github.com/relic-toolkit/relic/tree/amore`

Finally, as a byproduct of providing a fair experimental validation, we provide the first publicly available implementations for MV19 [45] and CKC23 [21].

1.2 Overview of our Techniques and Results

In what follows we provide an overview of our results and techniques. We try to keep it intuitive and minimize the use of notation (which can be found in subsection 2.2).

The **impossibility result** stems from observing that the verification equations of KC23 and KST23 are malleable. These are of the form $\gamma = \rho^r \cdot \xi^u$, where $\gamma, \rho, \xi \in \mathbb{G}_T$ are given by the server, the exponents r, u are known to the client only (client's secret randomness), and ρ is expected to be the evaluation of the delegated pairing. Theorem 1 states that any protocol for single- or batch pairing delegation in which the only verification checks are membership tests in $\mathbb{G}_T$ and equations like the above involving only $\mathbb{G}_T$ elements produced by the server, is malleable. The proof is constructive and presents a generic and successful

malleability attack: first compute the honest output (γ, ρ, ξ); then modify it by exponentiating all components to a common value $x \in \mathbb{Z}_q^* \setminus \{1\}$; and finally return $(\gamma^x, \rho^x, \xi^x)$ to the client. Clearly, $\gamma, \rho, \xi \in \mathbb{G}_T \Rightarrow \gamma^x, \rho^x, \xi^x \in \mathbb{G}_T$. Moreover, $\gamma = \rho^r \cdot \xi^u \Leftrightarrow \gamma^x = (\rho^x)^r \cdot (\xi^x)^u$, which would make the client accept the incorrect pairing value $\rho^x \neq \rho$.

The AmorE **protocol** follows the setting of, e.g., Chevallier et al. [20] and CDS14 [16], where the public parameters include a target group element $\gamma_T = e(P, Q)$. This design frees the client from computing pairings locally, and thus makes the storing of specialized pairing libraries unnecessary. The γ_T element is used to generate a long term secret $\xi = \gamma_T^s$ (with s uniformly random in $\mathbb{Z}_q^*$) used for all subsequent verifications. Since ξ is unknown to the server and appears in the verification equation, AmorE does not fall into the vulnerability class of our impossibility result. Therefore, at the overhead cost of just this one-time full-range exponentiation in $\mathbb{G}_T$, AmorE can verify the correctness of a sequence of N, single or batch, pairing delegations.

We begin by designing a protocol that is unconditionally secure by constructing n public values (besides the pairing inputs) involving $n + 1$ mutually masking secrets, thereby forming an underdetermined system of equations with a single degree of freedom. All variables come into play in the verification equation, to reconstruct the precomputed ξ.

We then look for efficient realizations that remain secure against realistic, time-bounded adversaries. Indeed, *once the delegation process is completed*, no harm can be done even if the client's secret randomness is leaked or revealed (after all the N rounds have concluded). To improve efficiency we adopt the well-known strategy of allowing the client to sample some randomness from a set of bounded size $[\![2^\varphi]\!] \subsetneq \mathbb{Z}_q^*$, see [45, 22, 3, 21], which allows for shorter scalar group exponentiations. Besides considering amortized efficiency in the sequential axis, we leverage the batch axis as well: our construction guarantees that the client cost scales only with the term $(\bar{\mathfrak{m}}_1 + \bar{\mathfrak{m}}_T + \in_T)$ as the batch size M grows (see Table 1), which already yields efficient asymptotics over the cost $\mathfrak{p}$ of computing a pairing.

Security proof technique After identifying the algebraic relation that characterizes successful forgeries against our protocol (Lemma 1), we establish that an adversarial view consisting of an underdetermined system of equations with one degree of freedom yields unconditional security (Theorem 2). However, additional constraints resulting from sampling two or more secrets from a bounded set, enable secret value discardment. Indeed, such secrets become *partially masking* ones, and evaluating carefully chosen public expressions in terms of them allows an adversary to rule out solutions by intersecting their corresponding generated sets. This fact is well described in subsection 6.2. **Everlasting security** is subsequently proven under the assumption that the adversary's computational resources are finite and proportional to the protocol's execution time (Theorem 3). Specifically, we limit the adversary in computing no more than 2^κ exponentiations in any of the bilinear groups within a pre-given timeout, and prove that setting the efficiency parameter φ to the value $\lceil (\sigma - 1)/2 + \kappa \rceil$ provides a statistical security of σ bits (see section 6, Lemma 2).

Concrete parameter instantiations We upperbound the adversary's resources to the current computational power of the *entire* Bitcoin network. As of 2025, the all-time high Bitcoin hashrate is $1,077.59$ EH/s, corresponding to nearly 2^{70} SHA256 hashes per second. Under the fair assumption that computing a single block header hash is less expensive than performing any bilinear group exponentiation, in this work (and at this point in time), we set $\kappa = 70 + \log(\tau)$, where τ represents the timeout of the protocol in seconds, i.e., we limit the adversary to performing no more than $2^{70} \cdot \tau$ scalar exponentiations across all three bilinear groups. Our use cases in section 7 –the costliest being sequentially delegating 10 batches of 100 pairings on the curve BLS48-575– take less than one second when run on our equipment, which is why we fixed $\tau = 1$.[5] In short, we set $\kappa = 70$ and $\varphi = 90$ when aiming for the standard $\sigma = 40$ bits of statistical security.

[5] Weak IoT devices dealing with server communication overhead may set a higher value for τ. In this case, however, the efficiency gain is only mildly degraded due to φ increasing logarithmically with τ.

Efficiency results We implement AmorE within the high-speed RELIC pairing library and, for the first time, achieve savings (in clock cycles) ranging from 22.7% to 45.4% (resp. 40.8% to 83.5%) for single (resp. batch) sequential delegation of pairings on BLS curves compared to computing them locally. Besides the protocol's design, a new method of efficiently sampling short scalars (see *Special-form challenges* in section 7) also contributes to the overall speedup. AmorE is thus the first pairing delegation protocol providing a substantial reduction in client computation.

1.3 Related Work

The concept of pairing delegation emerged two decades ago, in 2005, through three early works: Girault and Lefranc [29], which focuses on single-round server-aided verification (SAV) in general; Chevallier et al. [19] (published five years later in [20]), which addresses pairing delegation over several rounds; and Kang-Lee-Hong [39], which builds on [19] and targets pairing delegation with at least one secret input point.

The SAV framework proposed by Girault and Lefranc in [29] aims to offload the computationally expensive verification process of digital signatures to a powerful but untrusted server. For pairing-based signatures, such as ZSNS [60] and Boneh-Boyen [9], Girault and Lefranc demonstrate a secure delegation technique where the verifier's workload is reduced to only two exponentiations in the target group. While this approach was relevant at the time, due to the continuously evolving computational costs associated with pairing computations and related operations have since rendered it outdated. Moreover, in contrast to [19, 39, 20], the client in Girault-Lefranc's SAV protocols [29] does not recover the value of the outsourced pairing. Instead, the client probabilistically checks that the returned pairing satisfies the verification equation of the signature scheme. Follow-up work on SAV [58, 57, 48] improves the security model and combines SAV with delegation protocols for evaluating single pairings on public inputs.

Single pairing delegation with public inputs. This setting is relevant for signature verifications and has several applications as we discussed in section 1. We argue that no existing work achieves concrete efficiency for the client. For instance, Chevallier et al. [20] require the client to perform seven full-range $\mathbb{G}_T$ exponentiations, which is at least double the cost of computing a pairing using state-of-the-art techniques (see Table 6). To demonstrate client efficiency, Canard-Devigne-Sanders (CDS14) [16] split the client workload into an *offline* and an *online phase*. In a nutshell, the offline phase runs computations that are independent of the pairing argument, while the online phase uses the pairing input. Efficiency is claimed solely for the online phase cost, with the offline work disregarded. We implemented the public input protocol in CDS14 [16] and our test beds show that it is slightly inefficient for the client (see Tables 1 and 3), The current state-of-the-art pairing delegation protocol is LOVE, by Aranha et al. [3]. LOVE is a small improvement of Di Crescenzo et al.'s [22] pairing delegation protocol in the offline/online setting. In LOVE, verification is more efficient than in [22] and [16], however this claim holds only for the online phase (see Table 1). The latest work on this line of research is by Khodjaeva and Di Crescenzo (KD23) [42]. In this case, the client skips the offline phase altogether. This yields a malleable verification equation, as we show in section 3 that undermines the soundness of the delegation. Table 1 summarizes the state of the art for the most relevant work in single pairing delegation with public inputs and a single malicious server.

Batch pairing delegation with public inputs. A way to bypass the computational overhead of verification is to consider delegation of several pairings at once (batch). This idea appears in 2007 due to Tsang-Chow-Smith [56], together with a taxonomy of (batch) pairing delegation protocols based on the pairing arguments being a combination of public/secret and variable/constant input points. Again, we argue that existing work on batch pairing delegation with (variable) public input achieves limited (and unattractive) concrete efficiency for the client, especially when considering small batches ($N < 10$) on widely adopted curves. All protocols in [56] rely on costly exponentiations in the target group, and one pairing computation in the pre-processing phase that render them inefficient. Mefenza-Vergnaud (MV19) [45] introduced more efficient batch pairing delegation protocols by leveraging the endomorphism trick outlined in [33], along with reduced exponent sizes. MV19 requires the client to compute one pairing during pre-processing and our experimental results show that it is inefficient for small batches (see Table 4). The most recent works are from 2023, by Di

Crescenzo-Khodjaeva-Caro (CKC23) [21] and by Kalkar-Sertkaya-Tutdere (KST23) [38]. We implemented the "public online" protocol in CKC23 and found that its total client cost is consistently higher than in MV19 for all four BLS curves for batches of size $M < 10$ (see Tables 4 and 5). Moreover, despite claiming "unlimited client lifetime" and "after the offline phase, the number of delegation protocols executable by the resource-constrained client is an arbitrary polynomial", [21] lacks a formal model, and the security proof is only for one-shot protocols. Finally, similar to KD23, KST23 pursues efficiency by skipping the preprocessing and ends up with a verification equation that is malleable, which renders the scheme insecure against a malicious server.

Alternative lines of work Another line of work investigates delegation of pairings with at least one secret input, e.g., Kang-Lee-Hong [39] and Guillevic-Vergnaud [33]. Notably, Guillevic-Vergnaud's protocols are efficient and guarantee input secrecy, but are not verifiable, i.e., security is ensured against an honest but curious server, but not against a dishonest one. This setting has limited applications: identity based encryption schemes.

An alternative research line considers more servers via the (one-malicious) two-untrusted-program model [18, 37, 55]. We consider this setting is somewhat artificial: in real-world use cases, the client is resource-constrained (otherwise computing the pairing locally would not be a problem), and naturally aims to minimize costly wireless communications (more servers mean more communication channels to be established). Furthermore, the client may lack the necessary interfaces to support multiple servers, as is the case with hardware wallets that rely on USB-C connections for data transfer.

2 Background

2.1 A Swift Introduction to Bilinear Pairings

Let $\mathbb{G}_1$, $\mathbb{G}_2$ and $\mathbb{G}_T$ represent three possibly distinct groups of the same prime order q. In pairing-based cryptography, it is customary to describe $\mathbb{G}_1$, $\mathbb{G}_2$ using additive notation, while $\mathbb{G}_T$ is described using multiplicative notation. A cryptographic pairing, denoted as $e : \mathbb{G}_1 \times \mathbb{G}_2 \mapsto \mathbb{G}_T$, is a mapping characterized by being bilinear, i.e., $e(xA, yB) = e(yA, xB) = e(A, B)^{xy} \in \mathbb{G}_T$, where $A \in \mathbb{G}_1$, $B \in \mathbb{G}_2$ and $x, y \in \mathbb{Z}_q$; non-degenerate $e(A, B) \neq 1_{\mathbb{G}_T}$ if A, B are generators of the respective groups; and efficiently computable. In practical applications, cryptographic pairings are instantiated using pairing-friendly elliptic curves E defined over carefully chosen prime fields $\mathbb{F}_p$ and their extensions. For the bilinear pairings studied and implemented in this paper, the elements of the groups $\mathbb{G}_1$ and $\mathbb{G}_2$ are elliptic curve points in an order-q subgroup of $E(\mathbb{F}_p)$ and $E(\mathbb{F}_{p^k})$, respectively, where k is a small integer known as the embedding degree of the elliptic curve. The elements of $\mathbb{G}_T$ belong to an order-q subgroup in the finite field extension $\mathbb{F}^*_{p^k}$.

The bumpy road of implementing bilinear pairings at the 128-bit security level The first proof that bilinear pairings could be computed in polynomial time was provided by the seminal work of Victor Miller [46] in 2004. Later, the Barreto-Naehrig (BN) family of curves [6] stood as the preferred choice for pairing protocols, offering a 128-bit security level with primes slightly below 256 bits. However, a series of advancements in the Tower Number Field Sieve (TNFS) algorithm for attacking the discrete logarithm problem over prime finite field extensions, published between 2015 and 2016, diminished the security level of BN curves to less than 110 bits [43, 5, 32]. Presently, the most widely adopted family of pairing-friendly curves comprises the Barreto, Lynn, and Scott (BLS) curves, specifically instantiated as BLS12-381 and BLS12-383, utilizing 381- and 383-bit primes, respectively. The primary consequence of these nearly decade-old cryptanalysis advancements is that computing cryptographic pairings at the 128-bit security level now requires three times the computational effort compared to pre-attack scenarios.

2.2 Notation

– Lower case greek letters (except $\gamma, \eta, \rho, \xi, \phi$) denote **parameter** inputs: λ for security, φ for efficiency, σ for statistical security, τ for protocol duration, and κ for computational bound.

- In bilinear groups, P, Q, and $\gamma_T = e(P, Q)$ are the public generators of $\mathbb{G}_1$, $\mathbb{G}_2$, and $\mathbb{G}_T$, respectively.
- Concrete **group elements**: $A, C, P, U, W, Y \in \mathbb{G}_1$, $B, D, Q, V, X, Z \in \mathbb{G}_2$, and $\gamma, \eta, \rho, \xi \in \mathbb{G}_T$.
- Lowercase Latin letters denote **integers** such as indexes i, j; secret exponents r, s; and discrete logarithms of group elements, e.g., $a = \log_P(A)$, $b = \log_Q(B)$.
- "$[\![N]\!]$" denotes the range $\{1, \ldots, N\}$.
- **Vectors** are denoted by a right arrow above a symbol, e.g., $\overrightarrow{A} = (A_1, \ldots, A_N) \in \mathbb{G}_1^N$, while **Matrices** are denoted as bold symbols with one over right arrow, e.g., $\overrightarrow{\mathbf{r}} = (r_{ij})_{ij}$.
- "$\mathsf{negl}(\cdot)$" and "$\mathsf{poly}(\cdot)$" denote negligible and polynomial functions, respectively, while "$\Theta(\cdot)$" stands for Big Theta notation (recall that if $f \in \Theta(g)$, then f grows asymptotically as fast as g).
- $\mathcal{A}$ denotes the adversary, modeled as an algebraic algorithm, with specific runtime details provided later.
- "$\xleftarrow{\$}$" denotes random sampling from a set, and "$\leftarrow$" denotes value assignment.
- Bilinear group operations: $\mathfrak{m}_\iota$ (resp. $\bar{\mathfrak{m}}_\iota$) denotes scalar multiplication, $\mathfrak{o}_\iota$ a group operation, $\in_\iota$ membership testing, and $\mathfrak{p}$ a pairing computation.
- "$\mathsf{cost}(f)$" denotes the average computational cost of an algorithm f over a fixed number of executions with random inputs, measured, for example, by the number of group operations or the clock cycles required to execute the algorithm on a given device.
- $N \geq 1$ represents the number of rounds (delegations), and $M_i \geq 1$ is the batch size at round $i \in [\![N]\!]$.
- $L = \mathbf{1}^T \cdot \overrightarrow{M} = \sum_{i=1}^{N} M_i$ represents the total number of delegated pairings for fixed values N and $\overrightarrow{M}$.

2.3 Single & Batch Pairing Delegation

Models for single- and batch pairing delegation appear in a more or less formalized fashion in many works. Here, we provide a unifying syntax that additionally covers the sequential delegation setting introduced in this paper.

Generic Remarks Following established practice (e.g., [3, 38]), we assume that the communication channel between the client and the server is not vulnerable to integrity or replay attacks. Additionally, we assume that the client is always honest, while the server may be untrusted (malicious).

Syntax for Single Pairing Delegation Definition 1 recalls the framework for offline/online single pairing delegation in the single server setting introduced in [3, Definition 1], with the following minor adaptations to suit our work:

• For clarity, offSetup is renamed to OneTimeSetup (and `off.pp` to `sk`), onSetup to Setup, and onVerify to Verify (and `value` to `result`). (The distinction between online/offline phase is not as prominent when considering amortized efficiency).

• OneTimeSetup: instead of a single statistical security parameter σ, it takes as input tuple of auxiliary security parameters `aux.sec`.

• Verify: in addition to the one-time inputs (`sec` and `out`) it takes as input the client's long term `sk`. Everything else is syntactically equivalent to Definition 1 in [3], since that syntax provides a good foundation for any pairing delegation protocol.

With straightforward adaptations, the syntax in Definition 1 accommodates for describing also all public input pairing delegation protocols in [56, 20, 16, 45, 22, 42, 21, 38], see Appendix A for two concrete examples.

Definition 1 (Single Pairing Delegation Protocol). *A protocol* $\mathbf{\Pi}$ *for delegating pairing computations consists of five PPT algorithms* (GlobalSetup, OneTimeSetup, Setup, Compute, Verify) *with the following syntax:*

$\mathsf{GlobalSetup}(\lambda) \to \mathsf{pp}$ *is a randomized algorithm that takes as input the computational security parameter* λ *and returns the description of a bilinear map* $\mathsf{pp} := (q, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, P, Q, \gamma_T, e)$*, where* q *is a* (2λ)*-bit prime,* P*,* Q *and* γ_T *are generators of* $\mathbb{G}_1$*,* $\mathbb{G}_2$*, and* $\mathbb{G}_T$ *respectively, and* e *is a pairing. (We assume* pp *to be readily available to all other algorithms).*

$\mathsf{OneTimeSetup}(\mathtt{aux.sec}) \to \mathtt{sk}$ *is a randomized algorithm run by the* client *(once, during the offline phase). It takes as input a list of auxiliary security settings* aux.sec*, and generates long term secret material* sk *that may be accessed by other client-side algorithms. (We assume* aux.sec *to be readily available to all other algorithms).*

$\mathsf{Setup}(\mathtt{sk}, A, B) \to (\mathtt{pub}, \mathtt{sec})$ *is a randomized algorithm run by the* client *(possibly multiple times, in the online phase). It takes as input the client's long term key* sk *and the pairing arguments* $(A, B) \in \mathbb{G}_1 \times \mathbb{G}_2$*. It returns a public tuple* pub *for the* server*, and the client's secret randomness* sec *(possibly a vector) for the* client*.*

$\mathsf{Compute}(\mathtt{pub}) \to \mathtt{out}$ *is a deterministic algorithm run by the* server *(possibly multiple times, in the online phase). It takes as input the public tuple* pub*, and returns a tuple of public outputs* out*.*

$\mathsf{Verify}(\mathtt{sk}, \mathtt{sec}, \mathtt{out}) \to \mathtt{result}$ *is a deterministic algorithm run by the* client*. It takes as input the client's long term secret* sk*, the secret input* sec*, and the server's output* out*. It returns a value* $\mathtt{result} \in \{\mathbb{G}_T \cup \perp\}$*.*

Intended usage In all pairing delegation protocols, the GlobalSetup is run by a trusted party once to produce public parameters for all entities in the system. Protocols that do not consider online efficiency merge OneTimeSetup and Setup, e.g., [20, 16, 42]. The procedures Setup, Compute, and Verify are executed in this sequence to perform one pairing delegation. In [3, 22, 42] this triplet of algorithm identifies the so-called *online phase*, and efficiency is defined only for algorithms run in the online phase (the OneTimeSetup needs to run once per pairing delegation, but is not accounted for online efficiency estimates).

Syntax for Batch Pairing Delegation The syntax for batch pairing delegation is a natural generalization of Definition 1 to handle inputs and outputs as vectors.

Definition 2 (Batch Pairing Delegation Protocol). *A protocol* $\mathbf{\Pi}$ *for delegating pairing computations consists of five PPT algorithms* (GlobalSetup, OneTimeSetup, Setup, Compute, Verify) *with the following syntax:*

$\mathsf{GlobalSetup}(\lambda) \to \mathtt{pp}$ *is as in Definition 1.*
$\mathsf{OneTimeSetup}(\mathtt{aux.sec}) \to \mathtt{sk}$ *is as in Definition 1.*
$\mathsf{Setup}(\mathtt{sk}, A, B) \to (\mathtt{pub}, \mathtt{sec})$ *is as in Definition 1 with vectors of pairing arguments* $\overrightarrow{A}, \overrightarrow{B} \in \mathbb{G}_1^n \times \mathbb{G}_2^n$*.*
$\mathsf{Compute}(\mathtt{pub}) \to \mathtt{out}$ *is as in Definition 1.*
$\mathsf{Verify}(\mathtt{sk}, \mathtt{sec}, \mathtt{out}) \to \mathtt{result}$ *is as in Definition 1, with* $\mathtt{result} \in \{\mathbb{G}_T^n \cup \perp\}$*.*

Security Models In the public input setting, pairing delegation protocols are required to be correct (aka complete [16]) and secure (aka verifiable [16], correct [56], ϵ_s-result secure [21]). Defining correctness is rather straightforward: if the server behaves honestly the result output by the verification process is the intended (vector of) pairing(s). Security, on the other hand, states that a malicious server should be able to convince the client of an incorrect pairing value only with a small and well-defined probability (statistical security). For completeness, Figure 5 (in Appendix subsection A.1) reports the security experiments for single and batch pairing delegations in our notation.

3 An impossibility result for secure pairing delegation without preprocessing

Delegating the computation of a single pairing, or a batch, to a single untrusted server in a way that is both secure and efficient is a very challenging task, even when considering the pairing arguments as public inputs. To improve efficiency, it might be tempting to remove any client pre-processing on $\mathbb{G}_T$ elements. If done naively, e.g., in KC23 [42, Section 4] and KST23 [38, Alg. 1], this leads to insecure constructions for both single and batch pairing delegation. Theorem 1 identifies sufficient and necessary conditions to launch malleability attacks against single and batch pairing delegation protocols.

Appendix A presents KC23 and KST23 in our notation and detailed attacks against KC23. We discuss KST23 as a warm up to our impossibility result. In KST23, the server outputs two n long vectors of $\mathbb{G}_T$

elements (n denotes the size of the batch), $\texttt{out} = (\overrightarrow{\gamma}, \overrightarrow{\rho})$. The client verifies the delegation by checking that all elements belong to the target group, and that $\gamma_i = \rho_i^{r_i}$, for all $i \in [\![n]\!]$, with r_i being the client's secret exponents.

Our attack against KST23: Instead of the honest tuple $(\overrightarrow{\gamma}, \overrightarrow{\rho})$, the malicious server returns $\texttt{out}^* = (\overrightarrow{\gamma}^*, \overrightarrow{\rho}^*) = (\overrightarrow{\gamma}^x, \overrightarrow{\rho}^x)$, where $x \in \mathbb{Z}_q \setminus \{0, 1\}$. As before, $\overrightarrow{\gamma}, \overrightarrow{\rho} \in \mathbb{G}_T^n$ implies $\overrightarrow{\gamma}^*, \overrightarrow{\rho}^* \in \mathbb{G}_T^n$ since $\mathbb{G}_T$ is closed under multiplication. Moreover, for every $i \in [\![n]\!]$, $\gamma_i = \rho_i^{r_i} \Leftrightarrow \gamma_i^* = (\rho_i^*)^{r_i}$ since $\gamma_i^* = \gamma_i^x = (\rho_i^{r_i})^x = (\rho_i^x)^{r_i} = (\rho_i^*)^{r_i}$. Hence, $\texttt{out}^*$ is not rejected by the verification, and the client would accept the pairing values $\overrightarrow{\rho}^* \neq \overrightarrow{\rho}$.

Theorem 1. *Let* $\mathbf{\Pi} = (\mathsf{GlobalSetup}, \mathsf{OneTimeSetup}, \mathsf{Setup}, \mathsf{Compute}, \mathsf{Verify})$ *be a protocol for delegating a single pairing or a batch, in the single untrusted server model,* $\overrightarrow{\gamma} = (\gamma_1, \ldots, \gamma_n)$ *the values returned by the server, and* $\overrightarrow{r}$ *a tuple of secret exponents in* $\mathbb{Z}_q \setminus \{0\}$ *generated by the client. If* Verify *solely consists of:*

1. *Membership tests* $\gamma_i \overset{?}{\in} \mathbb{G}_T$ *for* $i \in [\![n]\!]$*, and*
2. *Equality tests of the form* $\phi_{\overrightarrow{r}}^{(j)}(\overrightarrow{\gamma}) \overset{?}{=} 1$ *for* $j \in [\![m]\!]$*, where* $m \geq 1$ *and* $\phi_{\overrightarrow{r}}^{(j)} : \mathbb{G}_T^n \to \mathbb{G}_T$ *is an homomorphism involving secret exponentiations of elements in* $\overrightarrow{r}$ *on the values* $\overrightarrow{\gamma}$*,*

then $\mathbf{\Pi}$ *is malleable.*

Proof. Recall that a homomorphism ϕ in a multiplicative group satisfies the following property: $\phi(\gamma_1, \ldots, \gamma_n) \cdot \phi(\gamma'_1, \ldots, \gamma'_n) = \phi(\gamma_1 \cdot \gamma'_1, \ldots, \gamma_n \cdot \gamma'_n)$. In particular, for any $x \in \mathbb{Z}_q$, $\phi(\gamma_1^x, \ldots, \gamma_n^x) = \phi(\gamma_1, \ldots, \gamma_n)^x$.

A malicious server choosing any value $x \in \mathbb{Z}_q \setminus \{0, 1\}$ and returning $\texttt{out}^* = (\gamma_1^*, \ldots, \gamma_n^*) = (\gamma_1^x, \ldots, \gamma_n^x) \neq \texttt{out}$ bypasses the Verify phase. Indeed, the elements of $\texttt{out}^*$ belong to $\mathbb{G}_T$ due to the group closure property, and for every $j \in [\![m]\!]$, it holds that

$$\phi_{\overrightarrow{r}}^{(j)}(\gamma_1^*, \ldots, \gamma_n^*) = \phi_{\overrightarrow{r}}^{(j)}(\gamma_1^x, \ldots, \gamma_n^x) = \phi_{\overrightarrow{r}}^{(j)}(\gamma_1, \ldots, \gamma_n)^x = 1^x = 1\,.$$

□

4 Modeling Sequential Pairing Delegation and Amortized Efficiency

Sequential pairing delegation can be expressed using the syntax introduced in Definition 1, though its intended usage changes: OneTimeSetup is meant to be run one time throughout the life span of a protocol instance, and the loop Setup, Compute, Verify can be run for a finite number $N \geq 1$ of times, using the output of OneTimeSetup. Sequential delegations allow us to consider amortized efficiency: the computational cost of OneTimeSetup should be amortized over several sequential (single or batch) pairing delegations. Naturally, the i-th delegation round is considered to be a *single* one when $M_i = 1$, and *batch* whenever $M \geq 1$.

In the remainder of this section, we present definitions of correctness, concrete amortized efficiency, and security for sequential pairing delegation protocols, covering both the single and batch settings.

4.1 Defining Correctness

The correctness property for a sequential pairing delegation protocol essentially states that: as long as all the algorithms are run honestly, for each delegation (round) $i \in [\![N]\!]$ the verification procedure should return $\texttt{result}_{ij} = e(A_{ij}, B_{ij})$, where $(A_{ij}, B_{ij}) \in \mathbb{G}_1 \times \mathbb{G}_2$ denote the j-th pairing arguments used for the i-th delegation (the size of the i-th batch, M_i, may vary at every delegation round).

Definition 3 (Correctness). *A protocol* $\mathbf{\Pi}$ *for sequential pairing delegation is said to be* L*-correct if, for any number of rounds (delegations)* N*, and batch sizes* $\overrightarrow{M} = \{M_i\}_{i=1}^N$ *satisfying* $\mathbf{1}^T \cdot \overrightarrow{M} \leq L$*, for any choice of parameters* λ *(computational security),* $\texttt{aux.sec}$ *(e.g., efficiency, statistical security, timeout); for any* pp *output by* $\mathsf{GlobalSetup}(\lambda)$*, and for any* sk *produced by* $\mathsf{OneTimeSetup}(\texttt{aux.sec})$*, it holds that:*

$$\Pr\left[\texttt{result} = \{e(A_{ij}, B_{ij})\}_{j \in [\![M_i]\!]} \;\middle|\; \begin{array}{c}(\texttt{pub}, \texttt{sec}) \leftarrow \mathsf{Setup}(\texttt{sk}, \overrightarrow{A}_i, \overrightarrow{B}_i) \\ \texttt{out} \leftarrow \mathsf{Compute}(\texttt{pub}) \\ \texttt{result} \leftarrow \mathsf{Verify}(\texttt{sk}, \texttt{sec}, \texttt{out})\end{array}\right] = 1 \qquad \forall i \in [\![N]\!].$$

4.2 Defining Amortized Efficiency

In the following, we refer to the client's computational cost as $\mathsf{cost}(\mathsf{client})$, which is a shorthand for the *total computational cost* of running all client-side algorithm of a protocol $\mathbf{\Pi}$. Specifically, this includes $\mathsf{cost}(\mathsf{OneTimeSetup})$; plus the cost of N delegations with batch sizes $\overrightarrow{M}$, i.e., the N-term sum of the cost of Setup and Verify, where the input to Verify is computed by evaluating $\mathsf{Compute}$ on the output of Setup, and vectors have size M_i for $i \in [\![N]\!]$.

Intuitively, amortized efficiency states that the protocol allows the client amortize the cost of $\mathsf{OneTimeSetup}$ over a large enough number of delegated pairings, hence providing concrete efficiency for the client. We remark that for a fixed number L of delegated pairings, $\mathsf{cost}(\mathsf{client})$ may vary depending on the number of delegation rounds N, and the respective batch sizes $\overrightarrow{M}$. As long as running $\mathbf{\Pi}$ with a specific choice $(N, \overrightarrow{M})$ (satisfying $L = \mathbf{1}^T \cdot \overrightarrow{M}$) for which $\mathsf{cost}(\mathsf{client})$ is less than the cost of evaluating L pairings locally, we say that $\mathbf{\Pi}$ is amortized efficient.

Definition 4 (Amortized Efficiency). *Let $\mathbf{\Pi}$ be a protocol for sequential pairing delegation, and $L \geq 1$ a positive integer. $\mathbf{\Pi}$ is said to be L-amortized efficient if there exists a tuple $(N, \overrightarrow{M})$ for delegating L pairings and a real value $\epsilon \in]0, 1[$ such that:*

$$\frac{\mathsf{cost}(\mathsf{client})}{L \cdot \mathfrak{p}} \leq \epsilon \,.$$

where $\mathsf{cost}(\mathsf{client})$ represents the client-side cost of running $\mathbf{\Pi}$ for delegating L pairings.

From Definition 4 we can draw the notion of concrete amortized efficiency for a fixed batch size $M \geq 1$, and a target efficiency ratio $\epsilon \in]0, 1[$ by finding (if it exists) the smallest integer $N \geq 1$ such that $\frac{\mathsf{cost}(\mathsf{client})}{M \cdot N \cdot \mathfrak{p}} \leq \epsilon$.

We remark that for one-shot pairing delegation protocols the ratio ϵ is constant since $\mathsf{cost}(\mathsf{client})$ is linear in $N \cdot M$ (the $\mathsf{OneTimeSetup}$ needs to be run at every new delegation round).

4.3 Security Model

We now present the security experiment for sequential pairing delegation. Our experiment, depicted in Figure 1, generalizes the one for one-shot pairing delegation defined, e.g., in [3, Definition 3] and reported in subsection A.1. Intuitively, the adversary $\mathcal{A}$ plays the role a malicious server which controls the pairing inputs at each delegation round.

In detail, $\mathbf{Exp}^{\mathsf{seq}}_{\mathbf{\Pi},\mathcal{A}}$ sets up an instance of a sequential pairing delegation protocol $\mathbf{\Pi}$ for a fixed number L of pairings; and for every delegation round, $\mathcal{A}$ selects the pairing arguments $(\overrightarrow{A}, \overrightarrow{B})$ adaptively, which are used as inputs in Setup. The adversary wins the experiment if

1. At a given delegation round i, a forgery is produced, i.e., an $\mathtt{out}^*$ for which the verification procedure Verify returns a tuple in $\mathbb{G}_T$ not matching $\{e(A_{ij}, B_{ij})\}_{j=1}^{M_i}$.
2. All outputs returned by the adversary prior to the forgery are honest.
3. The number of delegated pairings up that point doesn't exceed L.

This experiment aims to capture the behavior of an adversary attempting to gain knowledge during the initial rounds of the protocol and subsequently leveraging this information to launch an attack at a later stage of the protocol. As expected, setting $L = 1$ yields the same security experiment found in [3, Section 3] (for a one-shot single pairing delegation).

We remark that a more typical and stronger security experiment can be defined by allowing the adversary to return non successful forgeries $\mathtt{out}^*$ without automatically losing the game, simply by removing line 14 in $\mathbf{Exp}^{\mathsf{seq}}_{\mathbf{\Pi},\mathcal{A}}$. However, this extra requirement is not realistic in our framework: if the client rejects the server output due to being bogus, it can as well decide to interact with another server for future delegations, and/or refresh its secret randomness by re-running $\mathsf{OneTimeSetup}$. We believe these are easy and common fixes in the application scenarios we consider and should be leveraged, rather than discouraged.

$\mathbf{Exp}^{\mathsf{seq}}_{\mathbf{\Pi},\mathcal{A}}(\lambda, \mathtt{aux.sec}, L)$

1 : $\mathtt{pp} \leftarrow \mathsf{GlobalSetup}(\lambda)$
2 : $\mathtt{sk} \leftarrow \mathsf{OneTimeSetup}\,(\mathtt{aux.sec})$
3 : $\mathsf{ctr} \leftarrow 0$
4 : $\mathtt{st}_{\mathcal{A}} \leftarrow (\lambda, \mathtt{aux.sec}, \mathtt{pp})$
5 : **while** true **:**
6 : $\quad(\overrightarrow{A}, \overrightarrow{B}, \mathtt{st}_{\mathcal{A}}) \leftarrow \mathcal{A}(\mathtt{st}_{\mathcal{A}})$
7 : $\quad\mathsf{ctr} \leftarrow \mathsf{ctr} + \mathtt{len}(\overrightarrow{A})$
8 : $\quad$**if** $\mathsf{ctr} \geq L$ **: return** 0
9 : $\quad(\mathtt{pub}, \mathtt{sec}) \leftarrow \mathsf{Setup}(\mathtt{sk}, \overrightarrow{A}, \overrightarrow{B})$
10 : $\quad(\mathtt{out}^*, \mathtt{st}_{\mathcal{A}}) \leftarrow \mathcal{A}\,(\mathtt{pub}, \mathtt{st}_{\mathcal{A}})$
11 : $\quad\mathtt{result}^* \leftarrow \mathsf{Verify}(\mathtt{sk}, \mathtt{sec}, \mathtt{out}^*)$
12 : $\quad$**if** $\mathtt{result}^* \notin \{\perp, e(\overrightarrow{A}, \overrightarrow{B})\}$ **:**
13 : $\quad\quad$**return** 1
14 : $\quad$**if** $\mathtt{result}^* = \perp$ **: return** 0
15 : **return** 0

Fig. 1: Security experiment for sequential pairing delegation.

Definition 5 (Security). *For a given positive integer $L \geq 1$, a computational security parameter λ, and a tuple of auxiliary security parameters* aux.sec*, a sequential delegation protocol $\mathbf{\Pi}$ is said to be secure against a given family of adversaries if, for any $\mathcal{A}$ in the family, it holds that:*

$$\mathcal{P}\left[\mathbf{Exp}^{\mathsf{seq}}_{\mathbf{\Pi},\mathcal{A}}(\lambda, \mathtt{aux.sec}, L) = 1\right] \leq \mathsf{negl}(\lambda) + \sum_{\delta \in \mathtt{aux.sec}} \mathsf{negl}(\delta)$$

where $\mathbf{Exp}^{\mathsf{seq}}_{\mathbf{\Pi},\mathcal{A}}$ is depicted in Fig. 1.

5 The AmorE protocol

In this section we present AmorE, the first pairing delegation protocol that can be run sequentially with (concrete) amortized efficiency.

AmorE is instantiated with the following auxiliary security parameters: (1) a timeout parameter τ, which upperbounds the duration of the protocol in seconds, and (2) an efficient parameter φ, which balances the trade-off between (statistical) security and efficiency.

Introducing τ stems from the observation that a malicious server cannot develop a winning strategy before receiving the first public value from the client, or after the protocol has concluded. Hence, if it holds that the adversary's view pub only encodes ephemeral secret coins (which become useless, if disclosed *after* the delegation is over), it is reasonable to bound the adversary's running time in the duration of the protocol. In particular, we assume that the number of delegated pairings is linearly dependent on τ, i.e., $L \in \Theta(\tau)$. On the other hand, φ allows the client to adjust the security and efficiency of the construction, ranging from information-theoretic (but not necessarily efficient) to everlasting security against a bounded algebraic adversary, and significantly efficient.

AmorE is presented as pseudocode in Figure 2. In subsection 5.1, we describe the code flow while in subsection 5.2 we prove the correctness property of Definition 3. Two security results are subsequently presented in section 6: Theorem 2, where information-theoretic security is proven for the efficiency parameter

$\varphi = 2 \cdot \lambda$, any time parameter τ, and any number $L \in \mathsf{poly}(\lambda)$ of pairing delegations; and Theorem 3 where $\varphi \ll 2 \cdot \lambda$ allows AmorE achieve amortized efficiency, and is proven to be everlasting secure against adversaries operating under the Algebraic Group Model [26] that are computationally bounded during the execution time τ of the protocol.

5.1 Protocol Description

We describe AmorE in two blocks: the one time procedures and the ones for sequential delegation.

Protocol description (one-time procedures) The GlobalSetup is run once to generate the global public parameters pp, which describe a bilinear group and its generators. Specifically, pp is comprised of the field size q, a bilinear map e, the groups $\mathbb{G}_\iota$ for $\iota \in \{1, 2, T\}$, and their corresponding generators $P, Q, \gamma_T = e(P, Q)$. The global public parameters are publicly available to all algorithms. The OneTimeSetup is run once by the client, and takes as input the auxiliary security parameters $\mathtt{aux.sec} = (\tau, \varphi)$, which are set by the client. The time parameter τ will be used as a termination flag, while the efficiency one, φ, will tune efficiency and security (as discussed later in this section). A *long term secret* scalar s is uniformly sampled and the element $\xi = \gamma_T^{-s}$ is computed. The start time $\mathsf{t}_{\mathsf{start}}$ of the protocol is initialized, enabling time lapse checking against the time parameter τ at the verification phase of each delegation.

Protocol description (sequential delegations) The procedures Setup, Compute, Verify can be run multiple times in this sequence.

Setup is run by the client to initiate a new pairing delegation for the arguments $(\overrightarrow{A}, \overrightarrow{B})$. After checking that the inputs have consistent sizes and do not contain the infinity point, a field element u is uniformly sampled, generating the points $U \in \mathbb{G}_1$ and $V \in \mathbb{G}_2$ (lines 7-9), linked by the relation $e(U, V) = \xi$. We remark that these computations are independent of the pairing inputs, and hence can be precomputed. Next, whether the batch $(\overrightarrow{A}, \overrightarrow{B})$ consists of a single pair or multiple, determines the follow up logic, but essentially, a secret scalar r is uniformly sampled from $[\![\min\{2^\varphi, q-1\}]\!]$ for each pairs of inputs (A, B) in $(\overrightarrow{A}, \overrightarrow{B})$ and additional group elements are computed and checked against the infinity point in order to later verify the correctness of the server's ouput at the end of the interaction round. The public tuple $\left(\overrightarrow{A}, \overrightarrow{B}, \overrightarrow{C}, D, X, Y\right)$ is then sent to the server, while the secrets $\overrightarrow{r}$ are kept by the client.

Compute is the only algorithm run by the server. It takes pub as input and returns to the client the delegated pairing(s) $\rho_j = e(A_j, B_j)$ for $j \in [\![M]\!]$ along with the check value γ.

Verify is run by the client to conclude the delegation. It first ensures the protocol has not timed out (line 2) and then checks that the expected evaluated pairings $\overrightarrow{\rho}$ are $\mathbb{G}_T$ elements (in order to avoid small subgroup attacks, as in [53, Section 8.3]). The last check verifies the identity $\xi \stackrel{?}{=} \left(\prod_{j=1}^{M} \rho_j^{r_j}\right) \cdot \gamma$ (line 8). If all checks pass, the algorithm returns $\overrightarrow{\rho}$, else, it returns $\perp$.

$M = 1$ vs $M > 1$ delegation approach The public view is constructed differently depending on whether the delegation round consists of a single pairing or a batch of them, and the underlying reason is solely for leveraging efficiency (see Table 2). Indeed, for $M = 1$, it is more efficient to perform a short scalar multiplication on the $\mathbb{G}_2$ point D and a full one to the $\mathbb{G}_1$ point C. On the other hand, for $M > 1$, our verification formula allows us to perform M short multiplications on $\mathbb{G}_1$ and a single full one on a $\mathbb{G}_2$ point.

5.2 Protocol Correctness

This follows by inspection of the pseudo code in Figure 2. In line 4 of Compute, the server computes $\overrightarrow{\rho} = \{e(A_j, B_j)\}_{j=1}^{M}$ which is then returned in line 9 of Verify. It remains to show that the verification equation $\xi \stackrel{?}{=} \left(\prod_{j=1}^{M} \rho_j^{r_j}\right) \cdot \gamma$ in line 8 of Verify, is satisfied when all algorithms are run honestly. We have two cases:

$\mathsf{GlobalSetup}(\lambda) \to \mathtt{pp}$

// Bilinear Group parameters

$\mathtt{pp} \leftarrow (q, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, P, Q, \gamma_T, e)$

$\mathsf{Setup}(\mathtt{sk}, \overrightarrow{A}, \overrightarrow{B}) \to (\mathtt{pub}, \mathtt{sec})$

parse $\mathtt{sk} = (s, \xi, \cdot, \varphi, \cdot)$
if $\mathtt{len}(\overrightarrow{A}) \neq \mathtt{len}(\overrightarrow{B})$**:**
  return $\perp$
if $\mathcal{O} \in \overrightarrow{A} \cup \overrightarrow{B}$**:**
  return $\perp$
$M \leftarrow \mathtt{len}(\overrightarrow{A})$
$u \xleftarrow{\$} \mathbb{Z}_q^*$
$U \leftarrow [u]P$
$V \leftarrow [s \cdot u^{-1}]Q$
if $M = 1$**:**
  $r \xleftarrow{\$} [\![\min\{2^\varphi, q-1\}]\!]$
  $C \leftarrow \left[-s \cdot u^{-1}\right](U + A)$
  $D \leftarrow V - [r]B$
  $(X, Y) \leftarrow (\perp, \perp)$
else :
  $W \xleftarrow{\$} \mathbb{G}_1 \setminus \{\mathcal{O}\}$
  $D \leftarrow \sum_{j=1}^{M} B_j$
  $Y \leftarrow W - U$
  $X \leftarrow [u](D - V)$
  for $j \in [\![M]\!]$**:**
    $r_j \xleftarrow{\$} [\![\min\{2^\varphi, q-1\}]\!]$
    $C_j \leftarrow [r_j] A_j + W$
if $\mathcal{O} \in \{\overrightarrow{C}, D, X, Y\}$**:**
  return $\perp$
$\mathtt{pub} \leftarrow \left(\overrightarrow{A}, \overrightarrow{B}, \overrightarrow{C}, D, X, Y\right)$
$\mathtt{sec} \leftarrow \overrightarrow{r}$

$\mathsf{OneTimeSetup}(\mathtt{aux.sec}) \to \mathtt{sk}$

parse $\mathtt{aux.sec} = (\tau, \varphi)$
$s \xleftarrow{\$} \mathbb{Z}_q^*$
$\xi \leftarrow \gamma_T^{-s}$
$\mathtt{t}_{\mathsf{start}} \leftarrow \mathtt{time.now}()$
$\mathtt{sk} \leftarrow (s, \xi, \tau, \varphi, \mathtt{t}_{\mathsf{start}})$

$\mathsf{Compute}(\mathtt{pub}) \to \mathtt{out}$

parse $\mathtt{pub} = \left(\overrightarrow{A}, \overrightarrow{B}, \overrightarrow{C}, D, X, Y\right)$
$M \leftarrow \mathtt{len}(\overrightarrow{A})$
for $j \in [\![M]\!]$**:**
  $\rho_j = e(A_j, B_j)$
if $M = 1$**:**
  $\gamma \leftarrow e(A, D) \cdot e(C, Q)$
else :
  $\gamma \leftarrow \left(\prod_{j=1}^{M} e(C_j, -B_j)\right) \cdot e(Y, D) \cdot e(P, X)$
$\mathtt{out} \leftarrow (\gamma, \overrightarrow{\rho})$

$\mathsf{Verify}(\mathtt{sk}, \mathtt{sec}, \mathtt{out}) \to \mathtt{result}$

parse $\mathtt{sk} = (\cdot, \xi, \tau, \cdot, \mathtt{t}_{\mathsf{start}})$
if $\mathtt{time.now}() - \mathtt{t}_{\mathsf{start}} > \tau$**:**
  return $\perp$
parse $\mathtt{sec} = \overrightarrow{r}$
parse $\mathtt{out} = (\gamma, \overrightarrow{\rho})$
$M \leftarrow \mathtt{len}(\overrightarrow{r})$
if $\bigvee_{j \in [\![M]\!]} (\rho_j \notin \mathbb{G}_T)$**:** **return** $\perp$
if $\left(\xi = \left(\prod_{j=1}^{M} \rho_j^{r_j}\right) \cdot \gamma\right)$**:**
  return $\overrightarrow{\rho}$
return $\perp$

Fig. 2: Pseudocode for the AmorE protocol. The lines in grey are not input-dependent and can be pre-computed to prioritize online efficiency.

1. $M = 1$:

$$\begin{aligned}
\rho^r \cdot \gamma &= e(A,B)^r \cdot (e(A,D) \cdot e(C,Q)) \\
&= e(A,[r]B) \cdot e(A, V - [r]B) \cdot e\left([-s \cdot u^{-1}](U+A), Q\right) \\
&= e(A,V) \cdot e(U+A, [-s \cdot u^{-1}]Q) \\
&= e(A,V) \cdot e(U+A, -V) \\
&= e(U,-V) = e(P,Q)^{u \cdot (-s) \cdot u^{-1}} = \gamma_T^{-s} = \xi
\end{aligned}$$

2. $M > 1$:

$$\begin{aligned}
\left(\prod_{j=1}^{M} \rho_j^{r_j}\right) \cdot \gamma &= \left(\prod_{j=1}^{M} e(A_j, B_j)^{r_j}\right) \cdot \left(\prod_{j=1}^{M} e(C_j, -B_j)\right) \cdot e(Y,D) \cdot e(P,X) \\
&= \left(\prod_{j=1}^{M} e(A_j, B_j)^{r_j}\right) \cdot \left(\prod_{j=1}^{M} e([r_j]\, A_j + W, -B_j)\right) \cdot e(Y,D) \cdot e(P,X) \\
&= \left(\prod_{j=1}^{M} e(A_j, B_j)^{r_j}\right) \cdot \left(\prod_{j=1}^{M} e(A_j, B_j)^{-r_j}\right) \cdot \prod_{j=1}^{M} e(W, B_j)^{-1} \cdot e(Y,D) \cdot e(P,X) \\
&= e\left(W, \sum_{j=1}^{M} B_j\right)^{-1} \cdot e\left(W - U, \sum_{j=1}^{M} B_j\right) \cdot e\left(P, [u]\left(\sum_{j=1}^{M} B_j - V\right)\right) \\
&= e\left(-U, \sum_{j=1}^{M} B_j\right) \cdot e\left(U, \sum_{j=1}^{M} B_j - V\right) \\
&= e(U,-V) = e([u]P, [-su^{-1}]Q) = \xi
\end{aligned}$$

Hence, for a given number $L \in \Theta(\tau)$ of pairing delegations, and for any choice of delegation rounds N and corresponding batch sizes $\overrightarrow{M}$ satisfying $L = \mathbf{1}^T \cdot \overrightarrow{M}$, if all algorithms are run honestly, AmorE is L-correct as per Definition 3.

6 Security

Proof technique overview:

- In Lemma 1 (subsection 6.1), we prove that an algebraic adversary $\mathcal{A}$ attempting to produce a forgery against AmorE's verification equation must produce, in the simplest case, a pair (η, η^{r_j}) for some $\eta \in \mathbb{G}_T \setminus \{1\}$ and $j \in [\![M]\!]$.
- Next, in subsection 6.2 we show that in the efficient setting $\varphi < 2 \cdot \lambda$, extracting a secret scalar r_j from the public view pub reduces to finding the intersection of at least two sets of size 2^{φ} in one of the bilinear groups $\mathbb{G}_\iota$.
- Moreover, in subsection 6.3 we prove that when $\mathcal{A}$ is limited to computing 2^{κ} many φ-bit group operations for a time-dependent parameter $\kappa \in \Theta(\log(\tau))$,[6] setting $\varphi = \frac{\sigma-1}{2} + \kappa$ allows us to upperbound the success probability of finding this intersection by $2^{-\sigma}$, where σ is the statistical security parameter.

[6] In practice, $\kappa = 70 + \log(\tau)$ (τ in seconds), which adjusts to the all-time highest hashrate of the Bitcoin network as of 2025. This can be re-adjusted in the future.

– Finally, in subsection 6.4 and subsection 6.5, we provide an unconditional security result stated in Theorem 2 and an everlasting security one stated in Theorem 3.

6.1 Bypassing the verification equation

The following lemma identifies necessary and sufficient conditions to produce a valid forgery against any single- or batch-, one-shot or sequential pairing delegation protocol that employs a specific type of verification checks.

Lemma 1 (Identifying forgeries). *Let λ be a computational security parameter and $\mathbf{\Pi}$ a pairing delegation protocol over the bilinear group parameters $(q, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, P, Q, \gamma_T, e) \leftarrow \mathsf{GlobalSetup}(\lambda)$. If the* Verify *procedure of $\mathbf{\Pi}$ consists solely of checks of the form:*

$$\left(\xi = \prod_{i=1}^{M} \rho_i^{r_i} \cdot \gamma\right) \wedge (\rho_i \in \mathbb{G}_T)_{i \in [\![M]\!]} \tag{1}$$

for some $M \geq 1$, with $\xi \in \mathbb{G}_T$ and $\overrightarrow{r}$ being the client's secret values, and $\mathtt{out} = (\overrightarrow{\rho}, \gamma) \in \mathbb{G}_T^M \times \mathbb{G}_T$ the output of an honest server during a delegation; then, for any $\mathtt{out}^* \neq \mathtt{out}$ satisfying (1), there exists a non-empty subset of indexes $J \subset [\![M]\!]$, and elements $\eta_j \in \mathbb{G}_T \setminus \{1\}$ for $j \in J$ such that $\mathtt{out}^* = (\overrightarrow{\rho}^*, \gamma^*)$ with:

$$\rho_i^* = \begin{cases} \rho_i & \text{if } i \in [\![M]\!] \setminus J \\ \rho_i \cdot \eta_i & \text{if } i \in J \end{cases} \qquad \text{and} \qquad \gamma^* = \gamma \cdot \prod_{i \in J} \eta_i^{-r_i}.$$

Proof. By the closure property of the multiplicative group $\mathbb{G}_T$, any (forgery) vector $\mathtt{out}^* = \left(\overrightarrow{\rho}^*, \gamma^*\right) \neq \mathtt{out}$ can be written as a deviation from the honest output: $\mathtt{out} \cdot \overrightarrow{\eta}$, for some $\overrightarrow{\eta} \in \mathbb{G}_T^M \times \mathbb{G}_T$. Since $\mathtt{out}^* \neq \mathtt{out}$, at least two entries of $\overrightarrow{\eta}$ must be different from 1. Let $J \subset [\![M]\!]$ denote the non-empty subset of indexes of elements of $\mathtt{out}^*$ of the form $\rho_j^* = \rho_j \cdot \eta_j$, with $\eta_j \neq 1$. It remains to prove that $\eta_{M+1} = \prod_{j \in J} \eta_j^{-r_j}$. This fact is immediate, since any other value of η_{M+1} would lead to $\mathtt{out}^*$ not satisfying the verification check in (1), indeed by the equality in (1):

$$\xi = \prod_{i=1}^{M} (\rho_i^*)^{r_i} \cdot \gamma^* = \left(\prod_{i=1}^{M} \rho_i^{r_i} \cdot \gamma\right) \cdot \left(\prod_{i=1}^{M} \eta_i^{r_i} \cdot \eta_{M+1}\right) = \xi \cdot \left(\prod_{i=1}^{M} \eta_i^{r_i} \cdot \eta_{M+1}\right)$$

which implies $\eta_{M+1} = \prod_{i=1}^{M} \eta_i^{-r_i}$, hence $\gamma^* = \gamma \cdot \prod_{i=1}^{M} \eta_i^{-r_i}$. □

6.2 Extracting partially-masking secret variables

In this section we illustrate how an otherwise information-theoretic secure protocol becomes susceptible to an intersection attack when at least two of its secret variables are sampled from a reduced set, instead of the full-range $\mathbb{Z}_q^*$. In the specific case of AmorE, the reduced set is $[\![2^\varphi]\!] \subset \mathbb{Z}_q^*$ with $\varphi < 2 \cdot \lambda$ (see lines 11 and 21 in Setup in Figure 2).

A toy example: Let $(\mathbb{G} = \langle P \rangle, +)$ be a cyclic group of prime order q, and (r_1, r_2, u) a tuple of secret values such that $r_1, r_2, u \xleftarrow{\$} \mathbb{Z}_q^*$. Let $U = [u]P \in \mathbb{G}$, and consider the public view

$$\mathtt{pub} = \begin{cases} C = [r_1]U + X \\ D = [r_2]U + Y \end{cases}, \tag{2}$$

where $X, Y \in \mathbb{G}$ are public and $X \neq C, Y \neq D$. Note that r_1, r_2, u work as one time pads, effectively masking each other. Specifically, by denoting in lowercase the discrete logarithm with respect to P of group elements, we obtain the following system of equations:

$$\begin{cases} c = r_1 \cdot u + x \mod q \\ d = r_2 \cdot u + y \mod q \end{cases} \quad \Rightarrow \quad \begin{cases} r_1 = u^{-1} \cdot (c - x) \mod q \\ r_2 = u^{-1} \cdot (d - y) \mod q \end{cases}$$

which yields the parametrization

$$\big(r_1 = u^{-1} \cdot (c - x), r_2 = u^{-1} \cdot (d - y), u\big)$$

in $\mathbb{Z}_q$. This parametrization produces $q - 1$ equiprobable secret tuples (r_1, r_2, u) that yield `pub`. Hence, any protocol whose public view is `pub` and whose security relies on the masking of these secret variables, is information-theoretic secure.

Now, if one of the variables were sampled from a small subset, e.g, $r_1 \xleftarrow{\$} S \subset \mathbb{Z}_q^*$, parameterizing by r_1 would produce $|S|$ equiprobable secret tuples instead.

However, if $r_1, r_2 \xleftarrow{\$} S \subset \mathbb{Z}_q^*$, values for u can now be discarded according to the new constraint

$$u \in \mathcal{I} = \big\{[t_1^{-1}](c - x) \colon t_1 \in S\big\} \cap \big\{[t_2^{-1}](d - y) \colon t_2 \in S\big\} \ .$$

For a computationally bounded algebraic adversary, attempting to learn the secret tuple from (2) reduces to picking an element

$$U^* \in \mathcal{I} = \big\{[t_1^{-1}](C - X) \colon t_1 \in S\big\} \cap \big\{[t_2^{-1}](D - Y) \colon t_2 \in S\big\} \tag{3}$$

and its corresponding indexes r_1^*, r_2^*, which takes up to $2 \cdot |S|$ computations. [7] In the worst case (best case for $\mathcal{A}$), when $|\mathcal{I}| = 1$, finding (3) is analogous to breaking a 2-key-expansion of a block cipher (e.g., Double DES). In such case, for a plaintext-ciphertext pair (m, c), $\mathcal{A}$ would need to compute

$$\big\{\mathsf{DEC}(t_2, c) \colon t_2 \in \{0,1\}^k\big\} \cap \big\{\mathsf{ENC}(t_1, m) \colon t_1 \in \{0,1\}^k\big\} \ ,$$

which is most efficiently done via Diffie-Hellman's *meet-in-the-middle* attack in [24], taking up to 2^{k+1} operations.

In general: Consider n public elements belonging to a cyclic group $\mathbb{G}$ of prime order q, constructed from $n+1$ secret scalars that, if sampled uniformly from $\mathbb{Z}_q^*$, would work as one time pads, effectively masking each other. Suppose that at least two of these secret scalars are sampled from a small subset $S \subset \mathbb{Z}_q^*$, turning them into *partially-masking* variables. Then, it can be verified by inspection of the public view in question, that the most time-efficient strategy for a computationally bounded algebraic adversary, attempting to extract a secret scalar solely from the public view, is an intersection attack based on the bound constraints taking up to $2 \cdot |S|$ scalar computations in $\mathbb{G}$. This also holds when the public values belong to the bilinear groups $\mathbb{G}_1$ and $\mathbb{G}_2$ and they contain partially-masking variables. This is simply because the bilinearity of e merely allows an adversary to produce pairs (η, η^t) in $\mathbb{G}_T \times \mathbb{G}_T$ where t is a product or addition obtained from the discrete logarithms of the public values with respect to P and Q. In other words, while e may enable an intersection attack in $\mathbb{G}_T$, it does not give any additional advantage to $\mathcal{A}$ in producing a pair (η, η^r) for a secret scalar r as long as the public view is underdetermined.

6.3 Winning probability

The advantage of $\mathcal{A}$ in extracting a secret value from a public view consisting of a constrained underdetermined system of equations in the bilinear groups $\mathbb{G}_1$ and $\mathbb{G}_2$, can therefore be estimated by upperbounding the probability of the event that $\mathcal{A}$ succeeds in finding an intersection of the type (3) containing one of the client's secret values, without exceeding 2^κ group operations.

In our *toy example* from subsection 6.2, this event is equivalent to $\mathcal{A}$ selecting subsets S_1 and S_2 of $[\![2^\varphi]\!]$ that satisfy the budget limitation $|S_1| + |S_2| \leq 2^\kappa$, with each subset containing the secret values r_1 and r_2 respectively. This is required to ensure that

$$U \in \mathcal{I} = \big\{[t^{-1}](C - X) \colon t \in S_1\big\} \cap \big\{[t^{-1}](D - Y) \colon t \in S_2\big\} \ ,$$

[7] Observe that if $\mathcal{A}$ could extract discrete logarithms, it can simply run a membership test with a single iterator $t \in S$, by checking $t \cdot (d - y) \cdot (c - x)^{-1} \mod q \overset{?}{\in} S$ which takes up to $|S|$ tries.

so that $\mathcal{A}$ can choose r_i from a narrower set of possible secret tuples. Indeed,

$$\mathcal{P}\left[\{U \in \mathcal{I}\}\right] = \mathcal{P}[\{r_1 \in S_1\} \wedge \{r_2 \in S_2\}]\,.$$

The following lemma upperbounds the probability of this event in the general case where the public view is an underdetermined system involving L secret values r_i, sampled from the subset $[\![2^{\varphi}]\!] \subset \mathbb{Z}_q^*$.

Lemma 2 (Intersection success probability). *Let $\varphi, \kappa, \sigma, M$ be positive integers such that*

$$\varphi = \frac{\sigma - 1}{2} + \kappa\,.$$

For all $i \in [\![M]\!]$, let $r_i \xleftarrow{\$} [\![2^{\varphi}]\!]$. Then, for a collection of subsets $S_i \subset [\![2^{\varphi}]\!]$ selected without previous knowledge of the values $\overrightarrow{r}$, and satisfying $\sum_{i=1}^{M} |S_i| \leq 2^{\kappa}$, the probability of at least two of them, say S_j and S_k, respectively containing the values r_j and r_k for $j, k \in [\![M]\!]$, is negligible in σ. In particular,

$$\mathcal{P}\left[\bigvee_{k=2}^{M}\bigvee_{j=1}^{k-1}\{r_j \in S_j\} \wedge \{r_k \in S_k\}\right] \leq 2^{-\sigma}\,.$$

Proof. Let $c, n \in \mathbb{N}$ and $g : \mathbb{R}^n \to \mathbb{R}$ defined as $g(\overrightarrow{x}) = \sum_{j=1}^{n} x_i - c$. Consider a tuple $\{x_j\}_{j=1}^{n} \subset \mathbb{R}^+$ fulfilling the constraint $g(\overrightarrow{x}) = 0$. Then, it holds that

$$\left(\sum_{j=1}^{n} x_j\right)^2 = \sum_{j=1}^{n} x_j^2 + 2 \cdot \sum_{k=2}^{n}\sum_{j=1}^{k-1} x_j \cdot x_k$$

$$\Leftrightarrow \quad \sum_{1 \leq j < k \leq n} x_j \cdot x_k = \frac{1}{2} \cdot \left(\left(\sum_{j=1}^{n} x_j\right)^2 - \sum_{j=1}^{n} x_j^2\right) = \frac{1}{2} \cdot \left((g(\overrightarrow{x}) + c)^2 - \sum_{j=1}^{n} x_j^2\right) = \frac{1}{2} \cdot \left(c^2 - \sum_{j=1}^{n} x_j^2\right)\,,$$

from which we can establish that

$$\underset{\{x_j\}_{j=1}^{n}}{\arg\max}\left(\sum_{1 \leq j < k \leq n} x_j \cdot x_k\right)\Bigg|_{g(\overrightarrow{x})=0} = \underset{\{x_j\}_{j=1}^{n}}{\arg\min}\left(\sum_{j=1}^{n} x_j^2\right)\Bigg|_{g(\overrightarrow{x})=0} = \left\{\frac{c}{n}\right\}_{j \in [\![n]\!]}\,,$$

where the last equality follows from using a Lagrange multiplier to include the constraint $g(\overrightarrow{x}) = 0$ in the minimization. Under the new assignment $x_i \leftarrow |S_i|$ and $(c, n) \leftarrow (2^{\kappa}, N)$, [8] we can write

$$\sum_{1 \leq j < k \leq M} |S_j| \cdot |S_k| \leq \binom{M}{2} \cdot \left(\frac{2^{\kappa}}{M}\right)^2 = \frac{M \cdot (M-1)}{2} \cdot 2^{2\kappa} \cdot M^{-2} < 2^{2\kappa - 1}\,.$$

Now,

$$\mathcal{P}\left[\bigvee_{1 \leq j < k < M}\{r_j \in S_j\} \wedge \{r_k \in S_k\}\right] \leq \sum_{1 \leq j < k < M} \mathcal{P}\left[\{r_j \in S_j\}\right] \cdot \mathcal{P}\left[\{r_k \in S_k\}\right]$$

$$= \sum_{1 \leq j < k < M} \frac{\binom{2^{\varphi} - 1}{|S_j| - 1}}{\binom{2^{\varphi}}{|S_j|}} \cdot \frac{\binom{2^{\varphi} - 1}{|S_k| - 1}}{\binom{2^{\varphi}}{|S_k|}} = 2^{-2\varphi} \cdot \sum_{1 \leq j < k < M} |S_j| \cdot |S_k| < 2^{-2\varphi + 2\kappa - 1} = 2^{-\sigma}\,.$$

□

[8] $\sum_{i=1}^{n} x_i - c = 0 \Leftrightarrow \sum_{i=1}^{M} |S_i| = 2^{\kappa}$.

In the next subsection, to indicate that a certain value relates to the i-th round and the j-th pairing at round i, we will use the subscripts ij, e.g., A_{ij} is the j-th $\mathbb{G}_1$ pairing input at round i of the protocol. For values that are only round dependent, we will use a single subscript, e.g., the public value X_i, or the input A_i in the case that $M_i = 1$.

6.4 Unconditional security

AmorE achieves unconditional security when the efficient parameter φ is set to $2 \cdot \lambda$. This setting simply implies sampling r from $\mathbb{Z}_q^*$ in lines 11 and 21 of Setup in Figure 2.

Theorem 2. *For any λ, let q denote the $(2 \cdot \lambda)$-bit prime order of the bilinear groups $\mathbb{G}_\iota$ output by any execution of* $\mathsf{GlobalSetup}(\lambda)$. *Let* $\mathtt{aux.sec} = (\varphi, \tau)$, *with $\varphi = 2 \cdot \lambda$, $\tau > 0$ and $L := \mathsf{poly}(\lambda)$. Then,* AmorE *(Figure 2) instantiated in this way is information-theoretic secure. In particular, for any algebraic adversary $\mathcal{A}$, it holds that:*

$$\mathcal{P}\left[\mathbf{Exp}^{\mathsf{seq}}_{\mathbf{\Pi},\mathcal{A}}\left(\lambda, \mathtt{aux.sec} = (2 \cdot \lambda, \tau), L\right) = 1\right] \leq \frac{1}{q - 1 - 3 \cdot L},$$

where $\mathbf{Exp}^{\mathsf{seq}}_{\mathbf{\Pi},\mathcal{A}}$ is depicted in Figure 1.

Proof. Consider an adversary $\mathcal{A}$ entering the delegation round $i \in [\![N]\!]$ of the experiment. From Lemma 1, due to AmorE's verification equation and membership tests (lines 7 and 8 of Verify in Figure 2), $\mathcal{A}$ wins the experiment if and only if, it suceeds in producing in the simplest case, a triplet $(\eta, \eta^r, j) \in \mathbb{G}_T^2 \times [\![M_i]\!]$ for some secret exponent r used to verify the element ρ_{ij}. By inspecting the protocol, in addition to the pairing inputs $(\overrightarrow{\mathbf{A}}, \overrightarrow{\mathbf{B}})$, the public view up to round i consists of the following values:

$$\begin{cases} C_{kj} &= [r_{kj} a_{kj} + w_k]\, P \\ Y_k &= [w_k - u_k] P \\ X_k &= \left[u_k \cdot \sum_{j=1}^{M_k} b_{kj} - s\right] Q, \end{cases} \qquad \begin{cases} C_\ell &= \left[-s \cdot \left(1 + u_\ell^{-1} \cdot a_\ell\right)\right] P \\ D_\ell &= \left[s \cdot u_\ell^{-1} - r_\ell \cdot b_\ell\right] Q \end{cases} \tag{4}$$

where ℓ ranges over a subset of indexes $\mathcal{L} \subset [\![i]\!]$ identifying the delegation rounds consisting of a single pairing delegation ($M_\ell = 1$), and $(k, j) \in \{[\![i]\!] \setminus \mathcal{L}\} \times [\![M_k]\!]$. Now, (4) can be transformed into the following parametrization in $\mathbb{Z}_q$ (modulo q):

$$\begin{cases} u_k &= (x_k + s) \cdot \left(\sum_{j=1}^{M_k} b_{kj}\right)^{-1} \\ w_k &= y_k + (x_k + s) \cdot \left(\sum_{j=1}^{M_k} b_{kj}\right)^{-1} \\ r_{kj} &= \left(c_{kj} - y_k - (x_k + s) \cdot \left(\sum_{j=1}^{M_k} b_{kj}\right)^{-1}\right) \cdot a_{kj}^{-1} \end{cases} \qquad \begin{cases} u_\ell &= \left(-c_\ell \cdot s^{-1} - 1\right)^{-1} \cdot a_\ell \\ r_\ell &= \left((-c_\ell - s) \cdot a_\ell^{-1} - d_\ell\right) \cdot b_\ell^{-1}. \end{cases}^{9} \tag{5}$$

In (5), each value of s yields a secret tuple $(s, \overrightarrow{W}, \overrightarrow{\mathbf{r}}, \overrightarrow{u})$ that produces the view (4), as long as $\mathcal{O} \notin \overrightarrow{W}$ and $0 \notin \{\overrightarrow{\mathbf{r}}, \overrightarrow{u}\}$ [10]. Hence, for any $j \in M_i$, there are at least $q - 1 - (\sum_{k=1}^{i} M_k + 2 \cdot i - |\mathcal{L}|)$ possible and equiprobable secret values r_{ij} that could have been sampled according to $\mathcal{A}$'s view (recall that $|\overrightarrow{W}| = i - |\mathcal{L}|$). Since this number decreases as the number of delegated pairings increases, we can upperbound it by $q - 1 - 3 \cdot L$. □

[9] Recall that AmorE rejects points at infinity (see lines 5 and 24 in Setup); this implies that in order not to lose in the security experiment, $\mathcal{A}$ has to pick inputs $\overrightarrow{\mathbf{A}}$ and $\overrightarrow{\mathbf{B}}$ that do not contain points at infinity and such that $\sum_{j=1}^{M_k} B_{kj} \neq \mathcal{O}$ (hence all a_{kj} and $\sum_{j=1}^{M_k} b_{kj}$ are in $\mathbb{Z}_q^*$ and therefore invertible).

[10] The setting $\varphi = 2 \cdot \lambda$ implies that $r_{kj} \xleftarrow{\$} \mathbb{Z}_q^*$ in lines 11 and 21 of Setup, so no non-zero values can be discarded in (5) for the secrets $(\overrightarrow{w}, \overrightarrow{\mathbf{r}}, \overrightarrow{u})$.

6.5 Everlasting security

Security for AmorE in the efficient setting $\varphi < 2 \cdot \lambda$, can be proven against an adversary operating under the Algebraic Group Model [26], and with limited computational power during the execution of the sequential pairing delegation protocol (recall that $\mathcal{A}$ can only win $\mathbf{Exp}^{\mathsf{seq}}_{\mathbf{\Pi},\mathcal{A}}$ *between the start and the end of the protocol*). In particular, the protocol timeout parameter τ, determines an additional computational parameter $\kappa \in \Theta(\log(\tau))$, such that $\mathcal{A}$ is limited to computing 2^κ elements in any group $\mathbb{G}_\iota$ for $\iota \in \{1, 2, T\}$.

Theorem 3. *For any λ, let q denote the prime order of the groups output by any execution of* $\mathsf{GlobalSetup}(\lambda)$. *Let τ, σ be positive integers, and $\kappa \in \Theta(\log(\tau))$ satisfying $\varphi = \frac{\sigma-1}{2} + \kappa < 2 \cdot \lambda$. Then, for any positive integer $L \in \Theta(\tau)$,* AmorE *(Figure 2) is secure according to $\mathbf{Exp}^{\mathsf{seq}}_{\mathbf{\Pi},\mathcal{A}}$ against an algebraic adversary $\mathcal{A}$ limited to computing 2^κ elements in any group $\mathbb{G}_\iota$ for $\iota \in \{1, 2, T\}$. In particular,*

$$\mathcal{P}\left[\mathbf{Exp}^{\mathsf{seq}}_{\mathbf{\Pi},\mathcal{A}} = \left(\lambda, \mathsf{aux.sec} = \left(\varphi = \frac{\sigma-1}{2} + \kappa, \tau\right), L\right) = 1\right] \leq 2^{-\sigma}\,.$$

Proof. Given a pair $(N, \overrightarrow{M})$ such that $L = \mathbf{1}^T \cdot \overrightarrow{M}$, assume $\mathcal{A}$ enters the delegation round $i \in [\![N]\!]$ of $\mathbf{Exp}^{\mathsf{seq}}_{\mathsf{AmorE},\mathcal{A}}$. Recall from Lemma 1 that $\mathcal{A}$ needs to output at least a pair $(\eta, \eta^r) \in \mathbb{G}_T^2$ for some secret scalar r used in the verification equation at round i in order to win the experiment. Let $\mathcal{L} \subset [\![i]\!]$ indicate the rounds up to i which consist of delegations of a single pairing. In the AGM, given the view (4), $\mathcal{A}$ is restricted to computing elliptic curve points $G_\iota \in \mathbb{G}_\iota$ for $\iota = 1, 2$, of the general form

$$G_1 = \sum_{k,j} \left[z^{(1)}_{kj}\right] A_{kj} + \sum_{\ell \in \mathcal{L}} \left[z^{(2)}_{\ell}\right] C_\ell + \sum_{k \in [\![i]\!] \setminus \mathcal{L},\, j} \left[z^{(3)}_{kj}\right] C_{kj} + \sum_{k \in [\![i]\!] \setminus \mathcal{L}} \left[z^{(4)}_{k}\right] Y_k + \left[z^{(5)}\right] P \in \mathbb{G}_1$$

$$G_2 = \sum_{k,j} \left[\bar{z}^{(1)}_{kj}\right] B_{kj} + \sum_{\ell \in \mathcal{L}} \left[\bar{z}^{(2)}_{\ell}\right] D_\ell + \sum_{k \in [\![i]\!] \setminus \mathcal{L}} \left[\bar{z}^{(3)}_{k}\right] X_k + \left[\bar{z}^{(4)}\right] Q \in \mathbb{G}_2\,,$$

for adversarially chosen scalars $\overrightarrow{z}, \overrightarrow{\bar{z}} \subset \mathbb{Z}_q^*$. Additionally, these points can be used as inputs for the bilinear map e. Now, since $\mathcal{A}$'s view (4) is based on an underdetermined system of equations with one degree of freedom, even if the secret scalars r_{kj} are now sampled from $[\![2^\varphi]\!] \subset \mathbb{Z}_q^*$, $\mathcal{A}$ cannot construct a pair $(\eta, \eta^{r_{kj}}) \in (\mathbb{G}_T)^2$ with non-negligible probability from just e and a specific choice of scalars $\overrightarrow{z^*}, \overrightarrow{\bar{z}^*} \subset \mathbb{Z}_q^*$. As described in subsection 6.2, leveraging the constraints $r_{kj} \in [\![2^\varphi]\!]$ in this scenario, implies value discarding via an intersection on sets containing a common secret value. In particular, for minimal adversarial cost, these sets need to be generated by expressions that isolate the bounded secrets from any other source of randomness to allow a *meet-in-the-middle* attack. In the public view (4), the aforementioned constraints manifest themselves in several relations, e.g.,

$$\xi = \gamma_T^s \in \bigcap_{\ell \in \mathcal{L}} \left\{\rho_\ell^{t_\ell} \cdot \gamma_\ell \colon t_\ell \in [\![2^\varphi]\!]\right\} \subset \mathbb{G}_T\,, \tag{6}$$

where $\rho_\ell = e\,(A_\ell, B_\ell)$ and $\gamma_\ell = e\,(A_\ell, D_\ell) \cdot e\,(C_\ell, Q)$ for $\ell \in \mathcal{L}$.

Alternatively, general intersections can be established by removing the common source of randomness u_ℓ from C_ℓ and D_ℓ in (4) by choosing scalars $(z, \bar{z}, \hat{z}) \in (\mathbb{Z}_q^*)^3$ fulfilling $\bar{z} = z \cdot a_\ell$, and computing

$$\begin{aligned} e([z]C_\ell, [\hat{z}]Q) \cdot e([\hat{z}]P, [\bar{z}]D_\ell) &= e(\hat{z}P, Q)^{z \cdot c_\ell} \cdot e(P, \hat{z}Q)^{\bar{z} \cdot d_\ell} \\ &= e(\hat{z}P, Q)^{z \cdot \left(-s \cdot \left(1 + u_\ell^{-1} \cdot a_\ell\right)\right) + \bar{z} \cdot \left(s \cdot u_\ell^{-1} - r_\ell \cdot b_\ell\right)} \\ &= e(\hat{z}P, Q)^{-z \cdot s - \bar{z} \cdot r_\ell \cdot b_\ell} \\ &= \eta^{-z \cdot s - \bar{z} \cdot r_\ell \cdot b_\ell} \end{aligned}$$

to finally leverage the relation

$$\gamma_T^{z \cdot \hat{z} \cdot s} \in \bigcap_{\ell \in \mathcal{L}} \{\eta^{z \cdot s - \bar{z} \cdot r_\ell \cdot b_\ell} \cdot \eta^{\bar{z} \cdot t \cdot b_\ell} \colon t \in [\![2^\varphi]\!]\} \subset \mathbb{G}_T\,. \tag{7}$$

However, due to the sequential design of AmorE, $\mathcal{A}$ cannot cheat on rounds that have already passed, hence, these approaches are only useful for $\mathcal{A}$ if the current round i consists of a single pairing (i.e., $i \in \mathcal{L}$). In that case, $\mathcal{A}$ can narrow down values for r_i by finding subsets of $[\![2^{\varphi}]\!]$ that lead to non-empty intersections of the form (6) or (7), and win $\mathbf{Exp}^{\mathsf{seq}}_{\mathsf{AmorE},\mathcal{A}}$ with non negligible probability whenever $\varphi \ll 2 \cdot \lambda$. If we assume that $\mathcal{L} = \emptyset$, the only place where the relevant bounded secret exponents r_{ij} appear in the view (4), is in the expression $C_{ij} = [r_{ij}a_{ij} + w_i]\,P$, which in turn enables the intersection

$$W_i \in \bigcap_{j=1}^{M} \{C_{ij} - [t_j]A_{ij} \colon t_j \in [\![2^{\varphi}]\!]\} \subset \mathbb{G}_1\,. \tag{8}$$

Now, as expected from the analysis in subsection 6.2, in all intersection attacks, the underlying sets are obtained by ranging over the sampling space of the bounded secret values, i.e., $[\![2^{\varphi}]\!]$. Let $\mathcal{I}$ denote any intersection leading to value discardment of a bounded scalar r. Since $|\mathcal{I}|$ is the number of equiprobable possible values for r, the probability for an unbounded $\mathcal{A}$ of winning $\mathbf{Exp}^{\mathsf{seq}}_{\mathsf{AmorE},\mathcal{A}}$ equals $|\mathcal{I}|^{-1}$. However, for the worst case $|\mathcal{I}| = 1$, the winning probability of an adversary limited to computing 2^{κ} operations in any of the bilinear groups, is upperbounded by the adversary finding at least a pair of subsets $(S, \bar{S}) \subset [\![2^{\varphi}]\!] \times [\![2^{\varphi}]\!]$ such that $(r \in S) \wedge (\bar{r} \in \bar{S})$, for some other bounded scalar $\bar{r}$, and $|S| + |\bar{S}| \leq 2^{\kappa}$. For example, succeeding in finding such sets fulfilling $(r_{ij} \in S) \wedge (r_{ik} \in \bar{S})$ for $j, k \in [\![M_i]\!]$ in (8) yields in the worst case (best case for $\mathcal{A}$)

$$\{W_i\} = \{C_{ij} - [t]A_{ij} \colon t \in S\} \cap \left\{C_{ik} - [t]A_{ik} \colon t \in \bar{S}\right\}\,,$$

after which $\mathcal{A}$ extracts both r_{ij} and r_{ik}. For the setting $\varphi = \frac{\sigma-1}{2} + \kappa$, Lemma 2, upperbounds the probability of this event by $2^{-\sigma}$ for any fixed subset selection by $\mathcal{A}$. □

7 Implementation Results

We implemented our protocols using four BLS curves at different security levels in a private fork of the RELIC 7.0 cryptographic library, the same used to establish the previous state of the art [3]. Our code is currently available at a branch[11]. We reused the implementation of LOVE already included in the library and extended the codebase to include AmorE. In comparison to that work, we removed the curves BN-254 and BN-382 as parameter choices because BN curves are now considered obsolete, since the BN-254 curve does not reach 128 bits of security, and BN-382 is slower than other curves at the same security level. At 128-bit security, we adopted the widely-deployed BLS12-381 curve with embedding degree $k = 12$ and the $\mathbb{G}_T$-strong curve BLS12-383 that removes the need for subgroup membership testing in $\mathbb{G}_T$. We chose the BLS24-509 curves with embedding degree $k = 24$ and the curve BLS48-575 with embedding degree $k = 48$ for the 192- and 256-bit security levels, respectively [2, 44]. A brief summary of the curve families and concrete parameters we use can be found in Table 6 in Appendix B.

RELIC is a strategic choice for benchmarking our protocols because it implements all the aforementioned parameter sets with similar levels of optimization, such as assembly acceleration for Intel 64-bit platforms and the best-known formulas for arithmetic in pairing groups, allowing for a comprehensive and fair comparison. In the last few years, the library evolved to include slightly faster code for BLS12-381, faster membership testing with better formulas in the literature [35], and different approaches for scalar multiplication and exponentiation in the various groups. It was also not possible to obtain access to the same server-grade Intel Core i7-6700K Skylake processor used in LOVE, so we settled down instead on a similar but slightly less performant Core i7-7700 processor. For this reason, we independently benchmarked the existing LOVE implementation from scratch, noting that the timings do not deviate significantly from the original. Furthermore, we chose $\sigma = 40$ for the statistical security parameter across all protocols (instead of 50 in [3]), because it is a more standard choice in the literature, while noting that the performance impact of this change is minor. We admit that the target platform is clearly not representative of the resource-constrained devices

[11] `https://github.com/relic-toolkit/relic/tree/amore`

envisioned in this work, but we believe it is the best choice to illustrate the performance improvement of AmorE with respect to related work while allowing fair future comparisons.

Special-form challenges. An optimization we introduce with our AmorE implementation is a way to sample short challenges c more efficiently than previous work [3], which samples σ-bit challenges simply as random σ-bit scalars. By observing that $\mathbb{G}_2$ and $\mathbb{G}_T$ are equipped with efficient p-power Frobenius endomorphisms, we can exploit this property and sample c as $\Phi(k)$ subscalars c_i with $\lceil \frac{\sigma}{\Phi(k)} \rceil$ bits as $c = \sum_{0 \leq i < \Phi(k)} c_i p^i \bmod q$, with k the curve's embedding degree, and $\Phi(k)$ the dimension of the base-p scalar decompositions given by the curve family. Because $p \equiv z \bmod q$ in BLS curves, for the integer seed z defining the curve, the scalars will be computed in as $c = \sum_{0 \leq i < \Phi(k)} c_i z^i$ and no reductions modulo q are necessary. We can then use the Frobenius map ψ to evaluate a scalar multiplication in $\mathbb{G}_2$ as $[c]Q = \sum_{0 \leq i < \Phi(k)} [c_i]\psi^i(Q)$ in interleaved fashion, or the analogue in $\mathbb{G}_T$ for exponentiation, which reduces the complexity by a $\Phi(k)$-factor. This technique does not apply as efficiently to $\mathbb{G}_1$, due to the endomorphisms available in the group. The same idea can also be ported to other pairing-friendly curves, but it will not be as efficient as BLS due to the special relationship between p, z and q.

7.1 Timings for Operations in Pairing Groups

Timings for operations in pairing groups can be found at in Table 2, benchmarked on an Intel Core i7-7700 Kaby Lake running at 3.6GHz with HyperThreading (HT) and TurboBoost (TB) disabled for reducing measurement variability. We built the RELIC library by using the presets that come with the library and GCC version 14.2.1 with flags `-O3 -march=native -mtune=native -fomit-frame-pointer -finline-small-functions`. We included all the main operations in pairing groups required for the various protocols, and some optimized variants for shorter scalars. Variable-based scalar multiplication in $\mathbb{G}_1$ and $\mathbb{G}_2$ was implemented using the GLV/GLS method by an interleaved w-NAF approach with window size $w = 5$. Fixed-based scalar multiplication of generators P, Q in those groups was implemented using the comb method with 32 points, resulting in approximately half the cost of a variable-based scalar multiplication. Exponentiation in $\mathbb{G}_T$ exploits the p-power Frobenius as a GLS endomorphism in a windowed NAF approach. For shorter scalars, we benchmarked both the standard NAF approach in $\mathbb{G}_2$ and $\mathbb{G}_T$ with σ-bit scalars and the special-form challenges discussed previously. It is clear that the special-form challenges provide performance improvements across all parameter choices, ranging from 15% for exponentiation in $\mathbb{G}_T$ in BLS48-575 to 63% for scalar multiplication in $\mathbb{G}_2$ in BLS12-381.

7.2 Timings for Delegated Single Pairing Computation

Table 3 collects timings for the related work and our proposed AmorE protocol on the same benchmarking machine. The protocol operations include (preco) for the client precomputation, (onetime) for the client's OneTimeSetup, (online) for the client's Setup and Verify, and (server) for the server-side part of the computation. We make a distinction between (preco) and (onetime) because the latter is amortized across multiple protocol executions. We also include the total client latency (preco or amortized OneTimeSetup, together with Setup and Verify) as client for ease of comparison. We benchmark individual pairing delegations ($M = 1$) for LOVE and CDS14 [16], and $N = 10$ individual pairing delegations for AmorE, reporting the averaged latency to illustrate the amortized efficiency.

From the table, it is clear that CDS14 does not provide any performance gains for the delegating client, which motivated the introduction of a precomputation step to split the costs more favorably in offline/online phases. While LOVE was able to achieve speedups up to 75.5% from evaluating the online phase *only* in comparison with computing the pairing locally, AmorE is the first protocol able to claim speedups between 22.7% and 45.4% for the *entire* client computation with respect to computing the pairing locally. This is a much better performance metric to consider, for which we mark the corresponding figures with bold in the table, since each pairing delegation needs at least one execution of the offline step and an online phase. Note that this was essentially impossible to achieve for LOVE, since it required a pairing evaluation during the precomputation phase; and it comes at the cost of slightly more expensive online phase for the client and

Cost \ Curve	**BLS12-381**	**BLS12-383**	**BLS24-509**	**BLS48-575**
$\mathfrak{m}_1$	373	428	1,008	1,468
$\bar{\mathfrak{m}}_1$	161	185	287	340
$\mathfrak{m}_2$	718	748	4,187	18,052
$\bar{\mathfrak{m}}_2$	412	425	1,812	6,125
$\bar{\mathfrak{m}}_2$, special scalar	153	163	685	3,078
$\mathfrak{m}_T$	1,074	1,160	6,478	30,547
$\bar{\mathfrak{m}}_T$,	427	459	1,844	7,066
$\bar{\mathfrak{m}}_T$, special scalar	252	277	1,139	6,008
$\in_T$	361	35	1,294	4,461
Miller Loop	1,441	1,482	5,420	12,273
Final Exp	1,750	1,734	9,648	56,550
$\mathfrak{p}$ (ML+FE)	3,194	3,216	15,068	68,823

Table 2: Timings from the latest release of RELIC for operations in groups $\mathbb{G}_1, \mathbb{G}_2$ and $\mathbb{G}_T$, as per the notation in Table 1. Figures are reported in 10^3 cycles in a Kaby Lake processor, averaged over 10^4 executions (HT/TB disabled). Full-range scalar multiplication or exponentiation take a random scalar in $\mathbb{Z}_q^*$, while the short versions sample a random σ-bit scalar with $\sigma = 40$. Performance for longer but still short scalars, i.e. with $\varphi > \sigma$ bits, can be estimated by linearly scaling the figures, same for special-form scalars exploiting Frobenius. The cost $\mathfrak{p}$ for pairing computation is split in the Miller Loop and Final Exponentiation.

server in AmorE. Moreover, LOVE originally benefited more (20%) from a $\mathbb{G}_T$-strong curve, but subgroup membership testing is currently efficient enough to not make that much of a difference.

7.3 Timings for Delegated Batch Pairing Computation

We implemented the AmorE protocol for delegating a sequence of pairings. The baseline for AmorE is computing a batch of M pairings locally, such that their product can also be recovered by just multiplying the results in $\mathbb{G}_T$. Table 4 presents the timings. We note again that BLS12-383 does not provide a slight advantage anymore, due to faster membership testing; and that MV19 performs generally better than CKC23 in the online phase. The latter can be seen by closely inspecting the performance trade-offs in in the original paper or Table 1. AmorE shows consistent performance improvement between 40.8% and 73.2% in comparison to local computation, with M ranging from 2 to 10^2 pairings. It also approaches a 2-factor speedup in comparison to related works when the number of pairings increase, both in the online phase and total client time. Table 5 presents the timings for AmorE in BLS24-509 and BLS48-575, with larger speedups ranging from 45.9% to 83.5%, and in comparison to related work.

8 Conclusions and Remarks

This paper presents positive and negative results around pairing delegation protocols in the single untrusted server model.

On the negative side, we presented an impossibility result for single- and batch-pairing delegation protocols in the single untrusted server model. This result affects existing work [42, 38], and we hope will serve as a guideline for future work: while skipping the client-side preprocessing has attractive efficiency gains, it also has drastic security consequences.

On the positive side, we present AmorE a novel protocol for delegating multiple pairings on public arguments, in a sequential manner. This is a small tweak to the single pairing delegation setting and allows us to claim the first solution that is truly efficient for the client in an amortized sense, where amortization is achieved in two directions, one as the number of delegations grows, and the other as the size of the batch

Prot.\Curve		**BLS12-381**	**BLS12-383**	**BLS24-509**	**BLS48-575**
$\mathfrak{p}$		3,194	3,216	15,068	68,823
CDS14 [16]	(client)	4,477	4,564	24,237	104,398
	(server)	8,105	8,367	38,069	156,590
LOVE [3]	(preco)	4,624	4,808	23,687	105,552
	(online)	1,440	1,224	5,362	16,863
	(client)	6,064	6,032	29,049	122,415
	(server)	7,198	7,385	34,178	147,563
AmorE	(onetime)	1,085	1,154	6,597	30,742
	(online)	2,060	2,372	9,010	34,055
	(client)	**2,168**	**2,487**	**9,770**	**37,602**
		(32.1%)	*(22.7%)*	*(35.2%)*	*(45.4%)*
	(server)	7,184	7,410	34,297	148,116

Table 3: Timings for running various pairing delegation protocols. Figures are reported in 10^3 cycles in a Kaby Lake processor, averaged over 10^4 executions (HT/TB disabled). In the case of AmorE, we execute the protocol 10 times and take the average latency to illustrate the amortized efficiency property. For all protocols, the statistical security parameter is set to $\sigma = 40$. AmorE has additional parameter $\kappa = 70$, such that $\varphi = 90$. We highlight with bold the protocol instances where the total client time is faster than computing the pairing locally. In such cases cases, we display between parenthesis the corresponding efficiency gain computed as (1 - cost(client)/cost($\mathfrak{p}$)). Higher percentage values imply larger efficiency gains.

grows. Our experimental results show that average total client cost for single pairing delegations of AmorE is 22.7%-45.4% more efficient than computing the pairing locally, for reasonable parameter settings. This may be relevant in identity-based onion routing systems, e.g., [40]; Joux' three-party key agreement [36]; the inner-product argument of the simpler inner product argument in aPlonk [1], and in cases where the pairing arguments are progressively disclosed. For example, Boneh and Franklin discuss in [12, §1.1.2] a scenario where their ID-based encryption protocol could be adapted for the *Delegation of Duties* use case. In this scenario, encrypted messages m_i with a subject s_i can only be decrypted by entities possessing the corresponding private key from the authority, enabling decryption of messages with subject s_i, while preventing decryption of messages with any other subject $s_j \neq s_i$. It is conceivable that a constrained client wishing to implement this variant of the ID-based encryption, will need to use our AmorE protocol for the sequential delegation of individual pairings.

AmorE is also the most efficient and secure protocol to date for delegating multiple pairings on (potentially different) public arguments simultaneously. Our experimental results show that average total client cost for one round of AmorE is 40.8%-83.5% more efficient than computing the pairing locally, for reasonable parameter settings. Hence, AmorE finally enables IoT devices and hardware wallets to handle complex pairing-based cryptosystems provided a cheap and fast connection to an untrusted server. Applications include, but are not limited to, efficiently verify BLS signatures [14], polynomial commitments [41, 1], SNARKs [8, 31], SnarkPack [27], and performing secret leader election systems [17].

Our work rises the following open questions: Is restricting the adversary to an algebraic one a necessary condition to prove security for non-unconditionally secure pairing delegation schemes? Can everlasting security be leveraged to achieved public verifiability of the pairing computation? And finally, it would be worth to test the feasibility of our results for boasting existing services on off-the-shelf commodity devices.

References

1. Ambrona, M., Beunardeau, M., Schmitt, A.L., Toledo, R.R.: aPlonK: Aggregated PlonK from multi-polynomial commitment schemes. In: Shikata, J., Kuzuno, H. (eds.) IWSEC 23: 18th International Workshop on Secu-

Protocol/**Curve**		$M=2$	$M=3$	$M=10$	$M=10^2$
BLS12-381	$M\cdot\mathfrak{p}$	6,388	9,582	31,940	319,400
MV19 [45]	(preco)	2,005	2,704	7,762	72,819
	(online)	5,985	7,232	16,359	128,473
	(client)	7,990	9,936	**24,121**	**201,292**
	(speedup)	-	-	*(24.5%)*	*(37.0%)*
	(server)	9,404	18,948	63,203	628,331
CKC23 [21]	(preco)	3,909	3,900	3,876	3,901
	(online)	4,343	6,309	19,563	176,413
	(client)	8,243	10,209	**23,439**	**180,314**
	(speedup)	-	-	*(26.6%)*	*(43.5%)*
	(server)	11,323	15,400	43,893	409,179
AmorE	(onetime)	1,089	1,078	1,080	1,080
	(online)	3,410	4,234	10.103	85,325
	(client)	**3,528**	**4,341**	**10,211**	**85,433**
	(speedup)	*(44.8%)*	*(54.7%)*	*(68.0%)*	*(73.2%)*
	(server)	14,444	18,539	46,885	411,630
BLS12-383	$M\cdot\mathfrak{p}$	6,432	9,648	32,160	321,600
MV19 [45]	(preco)	2,128	2,878	8,256	77,256
	(online)	5,554	6,608	14,003	103,516
	(client)	7,682	**9,486**	**22,259**	**180,772**
	(speedup)	-	*(1.7%)*	*(30.3%)*	*(43.4%)*
	(server)	12,946	19,462	64,848	648,900
CKC23 [21]	(preco)	4,060	4,053	4,052	4,054
	(online)	3,764	5,514	17,521	160,458
	(client)	7,824	**9,567**	**21,673**	**164,512**
	(speedup)	-	*(0.8%)*	*(32.1%)*	*(48.9%)*
	(server)	11,639	15,839	45,404	423,867
AmorE	(onetime)	1,155	1,158	1,158	1.153
	(online)	3,869	4,789	11,498	95,843
	(client)	**3,804**	**4,904**	**11,613**	**95,958**
	(speedup)	*(40.8%)*	*(49.2%)*	*(63.9%)*	*(70.2%)*
	(server)	14,737	19,084	47,911	421,313

Table 4: Timings for delegation protocols for batches of M pairings averaged over $N = 10$ delegation rounds. All protocols are set with the statistical parameter $\sigma = 40$. AmorE has additional parameter $\kappa = 70$, such that $\varphi = 90$. Figures are reported in 10^3 cycles in a Kaby Lake processor, averaged over 10^4 executions (HT/TB disabled). Bold figures indicate instances where the total client time is smaller than computing the batch of pairings locally. For protocols with precomputation, the total client cost is computed as preco+online. For AmorE with amortized one-time setup, the total client is computed as online+onetime/10. Concrete efficiency gains are in parenthesis.

rity, Advances in Information and Computer Security. Lecture Notes in Computer Science, vol. 14128, pp. 195–213. Springer, Cham, Switzerland, Yokohama, Japan (August 29–31, 2023). https://doi.org/10.1007/978-3-031-41326-1_11

2. Aranha, D.F., Fotiadis, G., Guillevic, A.: A short-list of pairing-friendly curves resistant to the special TNFS algorithm at the 192-bit security level. IACR Commun. Cryptol. **1**(3), 3 (2024)
3. Aranha, D.F., Pagnin, E., Rodríguez-Henríquez, F.: LOVE a pairing. In: LATINCRYPT. Lecture Notes in Computer Science, vol. 12912, pp. 320–340. Springer (2021)
4. Aumann, Y., Ding, Y.Z., Rabin, M.: Everlasting security in the bounded storage model. IEEE Transactions on Information Theory **48**(6), 1668–1680 (2002). https://doi.org/10.1109/TIT.2002.1003845
5. Barbulescu, R., Duquesne, S.: Updating key size estimations for pairings. J. Cryptol. **32**(4), 1298–1336 (2019)

Protocol/**Curve**		$M=2$	$M=3$	$M=10$	$M=10^2$
BLS24-509	$M\cdot\mathtt{p}$	30,136	45,204	150,680	1,506,800
MV19 [45]	(preco)	12,198	16,387	45,812	424,197
	(online)	25,816	30,937	66,176	509,078
	(client)	38,014	47,324	**111,988**	**933,275**
	(speedup)	-	-	*(25.7%)*	*(38.0%)*
	(server)	60,740	91,210	303,981	3,036,994
CKC23 [21]	(preco)	19,531	19,503	19,493	19,487
	(online)	25,939	34,265	93,9908	817,400
	(client)	45,470	53,768	**113,401**	**836,887**
	(speedup)	-	-	*(24.7%)*	*(44.5%)*
	(server)	53,324	72,295	205,503	1,919,976
AmorE	(onetime)	6,556	6,554	6,560	6,555
	(online)	15,637	18,634	40,003	330,055
	(client)	**16,292**	**19,289**	**40,659**	**330,710**
	(speedup)	*(45.9%)*	*(57.3%)*	*(73.0%)*	*(78.0%)*
	(server)	68,500	87,546	220,964	1,937,533
BLS48-575	$M\cdot\mathtt{p}$	137,646	206,469	688,230	6,882,300
MV19 [45]	(preco)	53,680	72,371	200,027	1,848,335
	(online)	105,300	123,074	241,780	1,770,177
	(client)	158,980	**195,445**	**441,807**	**3,618,512**
	(speedup)	-	*(5.3%)*	*(35.8%)*	*(47.4%)*
	(server)	279,320	420,172	1,399,017	13,994,544
CKC23 [21]	(preco)	87,820	88,060	87,888	87,978
	(online)	101,407	132,988	359,678	3,041,980
	(client)	189,227	221,048	**447,566**	**3,129,958**
	(speedup)	-	-	*(35.0%)*	*(54.5%)*
	(server)	227,171	305,308	855,601	7,918,196
AmorE	(onetime)	30,883	30,849	30,871	30,852
	(online)	60,893	69.641	147,476	1,134,413
	(client)	**63,981**	**72,725**	**150,563**	**1,137,498**
	(speedup)	*(53.5%)*	*(64.8%)*	*(78.1%)*	*(83.5%)*
	(server)	297,007	375,632	924,288	7,990,114

Table 5: Timings on BLS24-509 and BLS48-575 following the same setting and format as in Table 4.

6. Barreto, P.S.L.M., Naehrig, M.: Pairing-friendly elliptic curves of prime order. In: SAC. LNCS, vol. 3897, pp. 319–331. Springer (2005)
7. Barua, R., Dutta, R., Sarkar, P.: Extending joux's protocol to multi party key agreement (extended abstract). In: Johansson, T., Maitra, S. (eds.) Progress in Cryptology - INDOCRYPT 2003, 4th International Conference on Cryptology in India, New Delhi, India, December 8-10, 2003, Proceedings. Lecture Notes in Computer Science, vol. 2904, pp. 205–217. Springer (2003)
8. Ben-Sasson, E., Chiesa, A., Genkin, D., Tromer, E., Virza, M.: SNARKs for C: Verifying program executions succinctly and in zero knowledge. In: Canetti, R., Garay, J.A. (eds.) Advances in Cryptology – CRYPTO 2013, Part II. Lecture Notes in Computer Science, vol. 8043, pp. 90–108. Springer Berlin Heidelberg, Germany, Santa Barbara, CA, USA (August 18–22, 2013). https://doi.org/10.1007/978-3-642-40084-1_6
9. Boneh, D., Boyen, X.: Short signatures without random oracles. In: Cachin, C., Camenisch, J. (eds.) Advances in Cryptology – EUROCRYPT 2004. Lecture Notes in Computer Science, vol. 3027, pp. 56–73. Springer Berlin Heidelberg, Germany, Interlaken, Switzerland (May 2–6, 2004). https://doi.org/10.1007/978-3-540-24676-3_4

10. Boneh, D., Crescenzo, G.D., Ostrovsky, R., Persiano, G.: Public key encryption with keyword search. In: Cachin, C., Camenisch, J. (eds.) Advances in Cryptology - EUROCRYPT 2004, International Conference on the Theory and Applications of Cryptographic Techniques. Lecture Notes in Computer Science, vol. 3027, pp. 506–522. Springer (2004)
11. Boneh, D., Drijvers, M., Neven, G.: Compact multi-signatures for smaller blockchains. In: International Conference on the Theory and Application of Cryptology and Information Security. pp. 435–464. Springer (2018)
12. Boneh, D., Franklin, M.K.: Identity-Based Encryption from the Weil Pairing. In: CRYPTO. LNCS, vol. 2139, pp. 213–229. Springer (2001)
13. Boneh, D., Gentry, C., Lynn, B., Shacham, H.: Aggregate and verifiably encrypted signatures from bilinear maps. In: Biham, E. (ed.) Advances in Cryptology — EUROCRYPT 2003. pp. 416–432. Springer Berlin Heidelberg, Berlin, Heidelberg (2003)
14. Boneh, D., Lynn, B., Shacham, H.: Short Signatures from the Weil Pairing. J. Cryptol. **17**(4), 297–319 (2004)
15. Campanelli, M., Gennaro, R.: Fine-grained secure computation. In: Beimel, A., Dziembowski, S. (eds.) TCC 2018: 16th Theory of Cryptography Conference, Part II. Lecture Notes in Computer Science, vol. 11240, pp. 66–97. Springer, Cham, Switzerland, Panaji, India (November 11–14, 2018). https://doi.org/10.1007/978-3-030-03810-6_3
16. Canard, S., Devigne, J., Sanders, O.: Delegating a pairing can be both secure and efficient. In: ACNS. LNCS, vol. 8479, pp. 549–565. Springer (2014)
17. Catalano, D., Fiore, D., Giunta, E.: Efficient and universally composable single secret leader election from pairings. In: Boldyreva, A., Kolesnikov, V. (eds.) PKC 2023: 26th International Conference on Theory and Practice of Public Key Cryptography, Part I. Lecture Notes in Computer Science, vol. 13940, pp. 471–499. Springer, Cham, Switzerland, Atlanta, GA, USA (May 7–10, 2023). https://doi.org/10.1007/978-3-031-31368-4_17
18. Chen, X., Susilo, W., Li, J., Wong, D.S., Ma, J., Tang, S., Tang, Q.: Efficient algorithms for secure outsourcing of bilinear pairings. Theoretical Computer Science **562**, 112–121 (2015). https://doi.org/https://doi.org/10.1016/j.tcs.2014.09.038, https://www.sciencedirect.com/science/article/pii/S0304397514007282
19. Chevallier-Mames, B., Coron, J., McCullagh, N., Naccache, D., Scott, M.: Secure delegation of elliptic-curve pairing. IACR Cryptol. ePrint Arch. p. 150 (2005)
20. Chevallier-Mames, B., Coron, J., McCullagh, N., Naccache, D., Scott, M.: Secure delegation of elliptic-curve pairing. In: CARDIS. LNCS, vol. 6035, pp. 24–35. Springer (2010)
21. Crescenzo, G.D., Khodjaeva, M., Caro, D.D.M.: Single-server batch delegation of variable-input pairings with unbounded client lifetime. In: European Symposium on Research in Computer Security. pp. 233–255. Springer (2023)
22. Crescenzo, G.D., Khodjaeva, M., Kahrobaei, D., Shpilrain, V.: Secure and efficient delegation of pairings with online inputs. In: CARDIS. LNCS, vol. 12609, pp. 84–99. Springer (2020)
23. Daira Emma Hopwood, Sean Bowe, T.H., Wilcox, N.: Zcash protocol specification. https://github.com/zcash/zips/blob/main/protocol/protocol.pdf (December 2023)
24. Diffie, W., Hellman, M.E.: Exhaustive cryptanalysis of the NBS data encryption standard. COMPUTER pp. 74–84 (1977)
25. Ding, Y.Z., Rabin, M.O.: Hyper-encryption and everlasting security. In: Alt, H., Ferreira, A. (eds.) STACS 2002. pp. 1–26. Springer Berlin Heidelberg, Berlin, Heidelberg (2002)
26. Fuchsbauer, G., Kiltz, E., Loss, J.: The algebraic group model and its applications. In: Shacham, H., Boldyreva, A. (eds.) Advances in Cryptology – CRYPTO 2018, Part II. Lecture Notes in Computer Science, vol. 10992, pp. 33–62. Springer, Cham, Switzerland, Santa Barbara, CA, USA (August 19–23, 2018). https://doi.org/10.1007/978-3-319-96881-0_2
27. Gailly, N., Maller, M., Nitulescu, A.: SnarkPack: Practical SNARK aggregation. In: Eyal, I., Garay, J.A. (eds.) FC 2022: 26th International Conference on Financial Cryptography and Data Security. Lecture Notes in Computer Science, vol. 13411, pp. 203–229. Springer, Cham, Switzerland, Grenada (May 2–6, 2022). https://doi.org/10.1007/978-3-031-18283-9_10
28. Garg, S., Jain, A., Mukherjee, P., Sinha, R., Wang, M., Zhang, Y.: hints: Threshold signatures with silent setup. In: 2024 IEEE Symposium on Security and Privacy (SP). pp. 3034–3052. IEEE (2024)
29. Girault, M., Lefranc, D.: Server-aided verification: Theory and practice. In: ASIACRYPT. LNCS, vol. 3788, pp. 605–623. Springer (2005)
30. Groth, J.: Short pairing-based non-interactive zero-knowledge arguments. In: Abe, M. (ed.) Advances in Cryptology - ASIACRYPT 2010 - 16th International Conference on the Theory and Application of Cryptology and Information Security, Singapore, December 5-9, 2010. Proceedings. Lecture Notes in Computer Science, vol. 6477, pp. 321–340. Springer (2010)

31. Groth, J.: On the size of pairing-based non-interactive arguments. In: EUROCRYPT (2). LNCS, vol. 9666, pp. 305–326. Springer (2016)
32. Guillevic, A.: A short-list of pairing-friendly curves resistant to special TNFS at the 128-bit security level. In: PKC (2). LNCS, vol. 12111, pp. 535–564. Springer (2020)
33. Guillevic, A., Vergnaud, D.: Algorithms for outsourcing pairing computation. In: CARDIS. LNCS, vol. 8968, pp. 193–211. Springer (2014)
34. Harnik, D., Naor, M.: On everlasting security in the hybrid bounded storage model. In: Bugliesi, M., Preneel, B., Sassone, V., Wegener, I. (eds.) ICALP 2006: 33rd International Colloquium on Automata, Languages and Programming, Part II. Lecture Notes in Computer Science, vol. 4052, pp. 192–203. Springer Berlin Heidelberg, Germany, Venice, Italy (July 10–14, 2006). https://doi.org/10.1007/11787006_17
35. Housni, Y.E., Guillevic, A., Piellard, T.: Co-factor clearing and subgroup membership testing on pairing-friendly curves. In: AFRICACRYPT. Lecture Notes in Computer Science, vol. 13503, pp. 518–536. Springer Nature Switzerland (2022)
36. Joux, A.: A one round protocol for tripartite Diffie-Hellman. In: Bosma, W. (ed.) Algorithmic Number Theory – ANTS IV. pp. 385–394. No. 1838 in LNCS, Springer (2000)
37. Kalkar, Ö., Kiraz, M.S., Sertkaya, İ., Uzunkol, O.: A more efficient 1–checkable secure outsourcing algorithm for bilinear maps. In: Information Security Theory and Practice: 11th IFIP WG 11.2 International Conference, WISTP 2017, Heraklion, Crete, Greece, September 28–29, 2017, Proceedings 11. pp. 155–164. Springer (2018)
38. Kalkar, O., Sertkaya, I., Tutdere, S.: On the batch outsourcing of pairing computations. The Computer Journal **66**(10), 2437–2446 (2023)
39. Kang, B.G., Lee, M.S., Park, J.H.: Efficient delegation of pairing computation. IACR Cryptol. ePrint Arch. **2005**, 259 (2005)
40. Kate, A., Zaverucha, G.M., Goldberg, I.: Pairing-based onion routing. In: Borisov, N., Golle, P. (eds.) PET 2007: 7th International Symposium on Privacy Enhancing Technologies. Lecture Notes in Computer Science, vol. 4776, pp. 95–112. Springer Berlin Heidelberg, Germany, Ottawa, Canada (June 20–22, 2007). https://doi.org/10.1007/978-3-540-75551-7_7
41. Kate, A., Zaverucha, G.M., Goldberg, I.: Constant-size commitments to polynomials and their applications. In: Abe, M. (ed.) Advances in Cryptology - ASIACRYPT 2010 - 16th International Conference on the Theory and Application of Cryptology and Information Security, Singapore, December 5-9, 2010. Proceedings. Lecture Notes in Computer Science, vol. 6477, pp. 177–194. Springer (2010)
42. Khodjaeva., M., Di Crescenzo., G.: On single-server delegation without precomputation. In: Proceedings of the 20th International Conference on Security and Cryptography - SECRYPT. pp. 540–547. INSTICC, SciTePress (2023). https://doi.org/10.5220/0012140100003555
43. Kim, T., Barbulescu, R.: Extended tower number field sieve: A new complexity for the medium prime case. In: CRYPTO (1). LNCS, vol. 9814, pp. 543–571. Springer (2016)
44. Mbang, N.M., Aranha, D.F., Fouotsa, E.: 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)
45. Mefenza, T., Vergnaud, D.: Verifiable outsourcing of pairing computations. https://tinyurl.com/Batch-Mefenza-Pairings (2018)
46. Miller, V.S.: The Weil Pairing, and Its Efficient Calculation. J. Cryptol. **17**(4), 235–261 (2004)
47. Novakovic, A., Eagen, L.: On proving pairings. Cryptology ePrint Archive (2024)
48. Pagnin, E., Mitrokotsa, A., Tanaka, K.: Anonymous single-round server-aided verification. In: LATINCRYPT. LNCS, vol. 11368, pp. 23–43. Springer (2017)
49. Parno, B., Howell, J., Gentry, C., Raykova, M.: Pinocchio: nearly practical verifiable computation. Commun. ACM **59**(2), 103–112 (2016)
50. Sahai, A., Waters, B.: Fuzzy identity-based encryption. In: Boneh, D. (ed.) Advances in Cryptology - EUROCRYPT 2005. Lecture Notes in Computer Science, vol. 3494, pp. 457–473. Springer (2005)
51. Sakai, R., Oghishi, K., Kasahara, M.: Cryptosystems based on pairing. In: 2000 Symposium on Cryptography and Information Security (SCIS2000), Okinawa, Japan. pp. 26–28 (Jan 2000)
52. Scott, M.: On the efficient implementation of pairing-based protocols. In: Chen, L. (ed.) Cryptography and Coding - 13th IMA International Conference, IMACC 2011, Oxford, UK, December 12-15, 2011. Proceedings. Lecture Notes in Computer Science, vol. 7089, pp. 296–308. Springer (2011)
53. Scott, M.: Unbalancing pairing-based key exchange protocols. Cryptology ePrint Archive, Paper 2013/688 (2013), https://eprint.iacr.org/2013/688
54. Shao, J., Wei, G.: Secure outsourced computation in connected vehicular cloud computing. IEEE Network **32**(3), 36–41 (2018)

55. Tian, H., Zhang, F., Ren, K.: Secure bilinear pairing outsourcing made more efficient and flexible. In: Bao, F., Miller, S., Zhou, J., Ahn, G.J. (eds.) ASIACCS 15: 10th ACM Symposium on Information, Computer and Communications Security. pp. 417–426. ACM Press, Singapore (April 14–17, 2015). https://doi.org/10.1145/2714576.2714615
56. Tsang, P.P., Chow, S.S.M., Smith, S.W.: Batch pairing delegation. In: IWSEC. LNCS, vol. 4752, pp. 74–90. Springer (2007)
57. Wang, Z.: A new construction of the server-aided verification signature scheme. Mathematical and computer modelling **55**(1-2), 97–101 (2012)
58. Wu, W., Mu, Y., Susilo, W., Huang, X.: Provably secure server-aided verification signatures. Computers & Mathematics with Applications **61**(7), 1705–1723 (2011)
59. Zavattoni, E., Perez, L.J.D., Mitsunari, S., Sánchez-Ramírez, A.H., Teruya, T., Rodríguez-Henríquez, F.: Software implementation of an attribute-based encryption scheme. IEEE Trans. Computers **64**(5), 1429–1441 (2015)
60. Zhang, F., Safavi-Naini, R., Susilo, W.: An efficient signature scheme from bilinear pairings and its applications. In: Bao, F., Deng, R., Zhou, J. (eds.) PKC 2004: 7th International Workshop on Theory and Practice in Public Key Cryptography. Lecture Notes in Computer Science, vol. 2947, pp. 277–290. Springer Berlin Heidelberg, Germany, Singapore (March 1–4, 2004). https://doi.org/10.1007/978-3-540-24632-9_20

Appendix

A Descriptions of KC23, KST23 and Concrete Attacks

Here we collect pseudo-code descriptions of KC23 [42, Section 4] and KST23 [38, Alg. 1] in our notation and present an explicit attacks.

In KC23, the server outputs three elements $\mathtt{out} = (\rho, \gamma_1, \gamma_2)$, and the client verifies the delegation by checking that ρ and γ_1 are in $\mathbb{G}_T$, and that $\gamma_2 = \rho^r \cdot \gamma_1^u$, where r, u are the client's secret randomness (and r is a short exponent). We argue that a malicious server can produce a successful forgery against KC23 with probability 1, exploiting the malleability of the verification equation. Notably, the security proof in [42] makes three claims that do not take into consideration the malleability of the verification equation.

$\mathsf{Setup}(A, B) \to (\mathtt{pub}, \mathtt{sec})$	$\mathsf{Compute}(\mathtt{pub}) \to \mathtt{out}$	$\mathsf{Verify}(\mathtt{sec}, \mathtt{out}) \to \mathtt{result}$
$r \xleftarrow{\$} [\![2^\sigma]\!]$	**parse** $\mathtt{pub} = (A, B, W, Y)$	**parse** $\mathtt{sec} = r$
$u \xleftarrow{\$} \mathbb{Z}_q^*$, $U \xleftarrow{\$} \mathbb{G}_1$	$\rho \leftarrow e(A, B)$	**if** $(\rho \notin \mathbb{G}_T$ OR $\gamma_1 \notin \mathbb{G}_T)$
$W \leftarrow [u^{-1}]U$	$\gamma_1 \leftarrow e(W, B)$	**return** $\perp$
$Y \leftarrow [r]A + U$	$\gamma_2 \leftarrow e(Y, B)$	**if** $(\gamma_2 \neq \rho^r \cdot \gamma_1^u)$
$\mathtt{pub} \leftarrow (A, B, W, Y)$	$\mathtt{out} \leftarrow (\rho, \gamma_1, \gamma_2)$	**return** $\perp$
$\mathtt{sec} \leftarrow r$		$\mathtt{result} \leftarrow \rho$

Fig. 3: Khodjaeva and Di Crescenzo's Protocol (KdC23) in [42, Sec. 4]. The GlobalSetup procedure generates a bilinear group, and there is no OneTimeSetup.

Our attack: Let $\mathtt{out} = (\rho, \gamma_1, \gamma_2)$ denote the honest server's output in KC23. Instead of $\mathtt{out}$, a malicious server can return $\mathtt{out}^* = (\rho^*, \gamma_1^*, \gamma_2^*) = (\rho^x, \gamma_1^x, \gamma_2^x)$ for any $x \in \mathbb{Z}_q \setminus \{0, 1\}$. Clearly $\mathtt{out} \neq \mathtt{out}^*$. It is easy to see that $\mathtt{out}^*$ is a valid forgery since: ρ^x and γ_1^x trivially belong to $\mathbb{G}_T$ and

$$\gamma_2^* = \gamma_2^x = (\rho^r \cdot \gamma_1^u)^x = \rho^{x \cdot r} \cdot \gamma_1^{x \cdot u} = (\rho^*)^r \cdot (\gamma_1^*)^u \,.$$

$\mathsf{Setup}(\overrightarrow{A}, \overrightarrow{B}) \to (\mathtt{pub}, \mathtt{sec})$	$\mathsf{Compute}(\mathtt{pub}) \to \mathtt{out}$	$\mathsf{Verify}(\mathtt{sec}, \mathtt{out}) \to \mathtt{result}$
for $i \in [\![\mathtt{len}(\overrightarrow{A})]\!]$**:**	**parse** $\mathtt{pub} = (\overrightarrow{A}, \overrightarrow{B}, \overrightarrow{C})$	**for** $i \in [\![\mathtt{len}(\overrightarrow{A})]\!]$**:**
$r_i \xleftarrow{\$} [\![2^{2\lambda-1}]\!]$	**for** $i \in [\![\mathtt{len}(\overrightarrow{A})]\!]$**:**	**if** $(\rho_i, \gamma_i \notin \mathbb{G}_T)$**:**
$C \leftarrow [r_i]A_i$	$\rho_i \leftarrow e(A_i, B_i)$	**return** $\perp$
$\mathtt{pub} \leftarrow (\overrightarrow{A}, \overrightarrow{B}, \overrightarrow{C})$	$\gamma_i \leftarrow e(C_i, B_i)$	**if** $(\gamma_i \neq \rho_i^{r_i})$**:**
$\mathtt{sec} \leftarrow (\overrightarrow{r})$	$\mathtt{out} \leftarrow (\overrightarrow{\rho}, \overrightarrow{\gamma})$	**return** $\perp$
		$\mathtt{result} \leftarrow \overrightarrow{\rho}$

Fig. 4: Kalkar, Sertkaya, and Tutdere batch pairing delegation protocol (KST23) with public variable inputs [38, Algorithm 1]. The GlobalSetup procedure generates a bilinear group, and there is no OneTimeSetup.

Definition 6 (Batch Pairing Delegation Protocol). *A protocol* $\mathbf{\Pi}$ *for delegating pairing computations consists of five PPT algorithms* (GlobalSetup, OneTimeSetup, Setup, Compute, Verify) *with the following syntax:*

$\mathsf{GlobalSetup}(\lambda) \rightarrow \mathtt{pp}$ *is as in Definition 1.*
$\mathsf{OneTimeSetup}(\mathtt{aux.sec}) \rightarrow \mathtt{sk}$ *is as in Definition 1.*
$\mathsf{Setup}(\mathtt{sk}, A, B) \rightarrow (\mathtt{pub}, \mathtt{sec})$ *is as in Definition 1 with vectors of pairing arguments* $\overrightarrow{A}, \overrightarrow{B} \in \mathbb{G}_1^n \times \mathbb{G}_2^n$.
$\mathsf{Compute}(\mathtt{pub}) \rightarrow \mathtt{out}$ *is as in Definition 1.*
$\mathsf{Verify}(\mathtt{sk}, \mathtt{sec}, \mathtt{out}) \rightarrow \mathtt{result}$ *is as in Definition 1, with* $\mathtt{result} \in \{\mathbb{G}_T^n \cup \perp\}$.

A.1 Security Models for Single and Batch Pairing Delegation

$\mathbf{Exp}^{\mathsf{single}}_{\mathbf{\Pi},\mathcal{A}}(\lambda, \mathtt{aux.sec})$

1 : $\mathtt{pp} \leftarrow \mathsf{GlobalSetup}(\lambda)$
2 : $\mathtt{sk} \leftarrow \mathsf{OneTimeSetup}(\mathtt{aux.sec})$
3 : $\mathtt{st}_{\mathcal{A}} \leftarrow (\lambda, \mathtt{pp}, \mathtt{aux.sec})$
4 : $(A, B, \mathtt{st}_{\mathcal{A}}) \leftarrow \mathcal{A}(\mathtt{st}_{\mathcal{A}})$
5 : $(\mathtt{pub}, \mathtt{sec}) \leftarrow \mathsf{Setup}(\mathtt{sk}, A, B)$
6 : $(\mathtt{out}^*, \mathtt{st}_{\mathcal{A}}) \leftarrow \mathcal{A}(\mathtt{pub}, \mathtt{st}_{\mathcal{A}})$
7 : $\mathtt{result}^* \leftarrow \mathsf{Verify}(\mathtt{sk}, \mathtt{sec}, \mathtt{out}^*)$
8 : **if** $\mathtt{result}^* \notin \{\perp, e(A, B)\}$ **:**
9 : **return** 1
10 : **return** 0

$\mathbf{Exp}^{\mathsf{batch}}_{\mathbf{\Pi},\mathcal{A}}(\lambda, \mathtt{aux.sec}, N)$

1 : $\mathtt{pp} \leftarrow \mathsf{GlobalSetup}(\lambda)$
2 : $\mathtt{sk} \leftarrow \mathsf{OneTimeSetup}(\mathtt{aux.sec})$
3 : $\mathtt{st}_{\mathcal{A}} \leftarrow (\lambda, \mathtt{pp}, \mathtt{aux.sec})$
4 : $\left(\overrightarrow{A}, \overrightarrow{B}, \mathtt{st}_{\mathcal{A}}\right) \leftarrow \mathcal{A}(\mathtt{st}_{\mathcal{A}})$
5 : $(\mathtt{pub}, \mathtt{sec}) \leftarrow \mathsf{Setup}(\mathtt{sk}, \overrightarrow{A}, \overrightarrow{B})$
6 : $\mathtt{out}^* x \leftarrow \mathcal{A}(\mathtt{pub}, \mathtt{st}_{\mathcal{A}})$
7 : $\mathtt{result}^* \leftarrow \mathsf{Verify}(\mathtt{sk}, \mathtt{sec}, \mathtt{out}^*)$
8 : **if** $\mathtt{result}^* \notin \left\{\perp, \{e(A_i, B_i)\}_{i=1}^N\right\}$ **:**
9 : **return** 1
10 : **return** 0

Fig. 5: Security experiments for single ($\mathbf{Exp}^{\mathsf{single}}_{\mathbf{\Pi},\mathcal{A}}$) and batch ($\mathbf{Exp}^{\mathsf{batch}}_{\mathbf{\Pi},\mathcal{A}}$) pairing delegation.

B Concrete Parameters

	BLS12 curves		**BLS24 curves**		**BLS48 curves**
$p(z)$	$(z-1)^2(z^4-z^2+1)/3+z$		$(z-1)^2(z^8-z^4+1)/3+z$		$(z-1)^2(z^{16}-z^8+1)/3+z$
$q(z)$	z^4-z^2+1		z^8-z^4+1		$z^{16}-z^8+1$
$t(z)$	$z+1$		$z+1$		$z+1$
$h(z)$	$(z-1)^2/3$		$(z-1)^2/3$		$(z-1)^2/3$
Curve	b	Seed z_0	$\lceil \log_2 p \rceil$	$\lceil \log_2 q \rceil$	k
BLS12-381	4	$-(2^{63}+2^{62}+2^{60}+2^{57}+2^{48}+2^{16})$	381	255	12
BLS12-383	4	$2^{64}+2^{51}+2^{24}+2^{12}+2^9$	383	256	12
BLS24-509	1	$-2^{51}-2^{28}+2^{11}-1$	509	408	24
BLS48-575	4	$2^{32}-2^{18}-2^{10}-2^4$	575	512	48

Table 6: Families of BLS pairing-friendly curves with various embedding degree k and concrete parameters used in our implementation. The curves BLS12-381 and BLS12-383 provide a conjectured 128-bit security level, while BLS24-509 and BLS48-575 provide 192- and 256-bit security.