Thesis for The Degree of Licentiate of Engineering

# Delegating Bilinear Pairings: Systematization, Amortized Efficiency, and Future Directions

Adrián Pérez Keilty


CHALMERS

Division of Computing Science
Department of Computer Science & Engineering
Chalmers University of Technology
Gothenburg, Sweden, 2026

**Delegating Bilinear Pairings: Systematization, Amortized Efficiency, and Future Directions**

Adrián Pérez Keilty



Department of Computer Science & Engineering
Division of Computing Science
Chalmers University of Technology
Gothenburg, Sweden

This thesis has been prepared using LaTeX.
Printed by Chalmers Reproservice,
Gothenburg, Sweden 2026.

*"If you like doing crypto but your workload is too high, try delegating it."*
*— A weak device*

# Abstract

Bilinear pairings are a fundamental tool in cryptography but computationally expensive when being run on resource-constrained devices, making delegation or outsourcing to a server a desirable alternative. However, designing a protocol that simultaneously *verifies* the server's output correctness and achieves *efficiency* over local computation has been a longstanding open problem. This thesis provides a systematization of existing work in this line of research, introduces the novel concepts of *amortized efficiency* and *sequential delegation*, and proposes new protocols that achieve significant and concrete efficiency gains for the first time in the literature.

**Keywords:** Bilinear Pairing, Verifiable Computation, Secure Outsourcing, Efficiency, IoT, Resource Constrained Environments

# Acknowledgments

First and foremost I'd like to thank my partner in crime, Nasima Samandari Sangari, who, a little over 3 years ago, inspired me to stick to mathematics and pursue this exciting journey in cryptography.

Second, I'd like to express my deepest gratitude to my PhD supervisor, Elena Pagnin, for her trust, support, fruitful discussions, and ultimately for accepting me as part of her growing team here at Chalmers.

Finally, I would like to express my sincere respect to all those who, in one way or another, contribute toward a freer world.

# Contents

# Notation and Glossary

### Parameters

| | |
|---|---|
| $\lambda$ | Security |
| $\varphi$ | Efficiency |
| $\sigma$ | Statistical security |
| $\tau$ | Protocol latency |
| $\kappa$ | Computational bound |

### Bilinear Groups

| | |
|---|---|
| $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e)$ | Bilinear groups with pairing $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ |
| $q$ | $2\lambda$-bit prime order of the groups $\mathbb{G}_1$, $\mathbb{G}_2$, and $\mathbb{G}_T$ |
| $P, Q$ | Public generators of $\mathbb{G}_1$ and $\mathbb{G}_2$ |
| $\gamma_T = e(P, Q)$ | Public generator in $\mathbb{G}_T$ |
| $\mathcal{O}$ | Identity element or point at infinity of $\mathbb{G}_1$ and $\mathbb{G}_2$ |
| $A, C, P, U, W, Y$ | Concrete group elements in $\mathbb{G}_1$ |
| $B, D, Q, V, X, Z$ | Concrete group elements in $\mathbb{G}_2$ |
| $\gamma, \eta, \rho, \xi$ | Concrete group elements in $\mathbb{G}_T$ |

## Integers and Scalars

| | |
|---|---|
| $i, j$ | Index variables |
| $r, t, \hat{t}, s$ | Secret scalar exponents in $\mathbb{Z}_q^*$ |
| $a = \log_P(A)$ | Discrete logarithm of $A$ with respect to $P$ |
| $[\![\mathsf{N}]\!]$ | Range notation denoting the set $\{1, \ldots, \mathsf{N}\}$ |

## Vectors and Matrices

| | |
|---|---|
| $\vec{A} = (A_1, \ldots, A_{\mathsf{M}})$ | Vector notation with right arrow, e.g., $\vec{A} \in \mathbb{G}_1^{\mathsf{M}}$ |
| $\overrightarrow{\mathbf{r}} = (r_{ij})_{ij}$ | Matrix notation as bold symbols with one over right arrow |

## Asymptotic Notation

| | |
|---|---|
| $\mathsf{negl}(\cdot)$ | Negligible function |
| $\mathsf{poly}(\cdot)$ | Polynomial function |
| $\Theta(\cdot)$ | Big Theta notation (if $f \in \Theta(g)$, then $f$ grows asymptotically as fast as $g$) |

## Probabilistic Notation

| | |
|---|---|
| $\mathcal{A}$ | An adversary or attacker |
| $\xleftarrow{\$}$ | Random sampling from a set |
| $\xleftarrow{\mathcal{D}}$ | Sampling according to a distribution $\mathcal{D}$ |
| $\leftarrow$ | Value assignment |
| $\mathsf{Dist}(X)$ | Set of all probability distributions on set $X$ |

## Bilinear Group Operations

| | |
|---|---|
| $\mathfrak{m}_\iota$ (resp.$\mathfrak{m}_\iota^{\mathsf{sh}}$) | Scalar multiplication (resp. short scalar multiplication) in $\mathbb{G}_\iota$ |
| $\mathfrak{m}_\iota^{\$}$ | Random sampling in $\mathbb{G}_\iota$ |
| $\mathfrak{g}_\iota$ | Group operation in $\mathbb{G}_\iota$ |
| $\in_\iota$ | Membership testing in $\mathbb{G}_\iota$ |
| $\mathfrak{p}$ | Pairing computation |

| | |
|---|---|
| $\mathsf{cost}(f)$ | Average computational cost of an algorithm $f$ over a fixed number of executions with uniformly sampled inputs |

## Pairing Delegation Notation

| | |
|---|---|
| $\mathsf{N} \geq 1$ | Number of rounds (delegations) |
| $\mathsf{M}_i \geq 1$ | Batch size at round $i \in [\![\mathsf{N}]\!]$ |
| $L = \mathbf{1}^T \cdot \vec{\mathsf{M}} = \sum_{i=1}^{\mathsf{N}} M_i$ | Total number of delegated pairings |
| $A_{ij}$ | The $j$-th $\mathbb{G}_1$ pairing input at round $i$ of the protocol |
| Subscript convention | Double subscripts $ij$ indicate values related to the $i$-th round and $j$-th pairing; single subscripts indicate round-only dependence |

## Protocol Naming Convention

Throughout this thesis, pairing delegation protocols are referenced using the initials of the authors concatenated with the year of publication. For example, a protocol by Kang, Lee, and Park in [63], which was published in 2005 is denoted as KLP05. If a specific protocol of a paper is referenced, it is indicated in brackets after the protocol name, e.g., MV19 [73, Alg. 2].

# Introduction

Modern cryptography underpins the security of digital communications, e-commerce, cloud computing, and countless other applications in an increasingly connected world. As cryptographic protocols become more sophisticated the computational demands on end devices grow correspondingly. This tension between cryptographic design sophistication and computational constraints has motivated a fundamental paradigm in cryptography: *secure outsourcing*, where resource-constrained devices delegate expensive cryptographic operations to more powerful servers while maintaining security guarantees. The challenge of secure outsourcing is particularly acute for *bilinear pairings*, a powerful tool that has become a cornerstone of modern public-key cryptography.

**Bilinear Pairings.** A cryptographic pairing is a bilinear and non-degenerate map

$$e\colon \mathbb{G}_1 \times \mathbb{G}_2 \mapsto \mathbb{G}_T$$

that maps pairs of group elements to a target group. The crucial property is *bilinearity*: for group elements $P \in \mathbb{G}_1$, $Q \in \mathbb{G}_2$, and scalars $a, b$, the pairing satisfies

$$e([a]P, [b]Q) = e(P, Q)^{ab} = e([b]P, [a]Q) = e(P, [b]Q)^{a} = e([a]P, Q)^{b}.$$

This seemingly simple algebraic property enables remarkably powerful cryptographic constructions that would be infeasible with traditional discrete logarithm-based cryptography.

**Bilinearity Relevance.** As an illustration, consider Joux's tripartite Diffie-Hellman [59], a one-round key agreement protocol between three parties Alice, Bob, and Charlie. The protocol uses a symmetric pairing ($\mathbb{G}_1 = \mathbb{G}_2 = \mathbb{G}$) and each party holds a secret scalar: Alice has $a$, Bob has $b$, and Charlie has $c$. Using an agreed-upon generator $P \in \mathbb{G}$, each party broadcasts a single group element: Alice broadcasts $[a]P$, Bob broadcasts $[b]P$, and Charlie broadcasts $[c]P$. Through the bilinearity of the pairing, each party can independently compute the shared secret key

$$K = e(P,P)^{abc} = e([b]P,[c]P)^{a} = e([a]P,[c]P)^{b} = e([a]P,[b]P)^{c}$$
$$\uparrow \qquad\qquad \uparrow \qquad\qquad \uparrow$$
$$\text{Alice} \qquad\qquad \text{Bob} \qquad\qquad \text{Charlie}$$

without any additional interaction. This construction is impossible with traditional Diffie-Hellman groups, where no such computational "shortcut" exists for combining three parties' contributions in a single round. Since their introduction to cryptography, pairings have enabled breakthrough constructions across diverse cryptographic domains: identity-based encryption (where email addresses can serve as public keys) [21], short signatures (achieving constant-size signatures regardless of message length) [24], attribute-based encryption (enabling fine-grained access control based on user attributes) [53], and succinct zero-knowledge proofs (allowing efficient verification of complex computations) [54, 13], among many others. These applications have transformed theoretical possibilities into practical systems deployed in real-world settings, from blockchain technologies to privacy-preserving authentication schemes.

**Computational Cost.** However in practice, pairings come at a significant computational cost. They are generally instantiated over an elliptic curve in which $\mathbb{G}_1$ and $\mathbb{G}_2$ are cyclic subgroups of the same prime order. The computation of pairings is based on Miller's algorithm [74], which evaluates a rational function through a double-and-add procedure analogous to scalar multiplication. For example, evaluating the optimal Ate pairing on

the widely used curve BLS24-509 on a 13th Gen Intel Core i5 processor, takes on average $7,591,268$ clock cycles, whereas a scalar multiplication in $\mathbb{G}_1$ takes only $373,290$. This computational asymmetry creates a critical barrier for resource-constrained devices such as IoT sensors, smart cards, and especially hardware wallets, whose processing power directly prohibits pairing computations. As pairing-based cryptography continues to proliferate in bandwidth-constrained and energy-limited environments, the need for efficient delegation mechanisms becomes increasingly pressing.

## Pairing Delegation

The problem of *pairing delegation* addresses this computational bottleneck by allowing a resource-constrained client to outsource the evaluation of bilinear pairings to a more powerful but potentially untrusted server. The fundamental challenge is to design delegation protocols that simultaneously achieve four critical properties:

1. **Verifiability** (also referred to as *security* or *soundness*): The client must be able to detect with high probability when the server returns an incorrect result, even when facing a malicious server that actively attempts to provide false outputs.

2. **Efficiency**: The client's computational workload—including preprocessing, communication, and verification—must be substantially lower than the cost of local pairing evaluation. Without concrete efficiency gains, delegation offers no practical advantage.

3. **Input privacy**: For applications handling sensitive data, the server should learn no information about the client's pairing inputs beyond what is revealed by the protocol's output. This property is essential for applications such as searchable encryption [26, 81] and forward-secure encryption [30, 89].

4. **Output privacy**: In special cases, the client may wish to hide the pairing evaluations it seeks to delegate. Learning these values could potentially allow the server to distinguish whether the verification of the underlying pairing-based protocol run by the client resulted in success or failure by checking equalities between the delegated pairings (e.g.,

to verify a BLS signature the client needs to verify the equality between two pairings).

Achieving these properties simultaneously has proven to be a challenging task. A good illustration of how pairing delegation typically looks like can be given by describing the CDS14 protocol for public inputs proposed by Canard, Devigne and Sanders in [29]. This protocol can be compactly described by the verification equation

$$\begin{gathered} e\left(\left[y^{-1}\right]A + [x]P, \left[x^{-1}\right]B + [y]Q\right) \cdot [e(P,B) \cdot e(A,Q)]^{-1} \\ \stackrel{?}{=} \\ e(P,Q)^{x \cdot y} \cdot e(A,B)^{(x \cdot y)^{-1}}, \end{gathered} \tag{1}$$

where the curve points $A, B, P, Q$ are public and $x, y \xleftarrow{\$} \mathbb{Z}_q^*$ are secret scalars. The client in this case computes *offline* (at any chosen time) the value $e(P,Q)^{x \cdot y}$ and *online* (when the pairing inputs $A$ and $B$ become available to the client) the values

$$\mathsf{pub} = (C, D) = \left(\left[y^{-1}\right]A + [x]P, \left[x^{-1}\right]B + [y]Q\right)$$

which will be sent to the server. The server returns the evaluations

$$\mathsf{out} = \left(e(C,D) \cdot [e(P,B), e(A,Q)]^{-1}, e(A,B)\right)$$

and finally the client checks the verification in Equation 1. The total client cost can be symbolically written as

$$\mathsf{cost}(\text{CDS14}) = 2\mathfrak{m}_1 + 2\mathfrak{m}_2 + 2\mathfrak{m}_T + \in_T + \mathfrak{g}_1 + \mathfrak{g}_2 + \mathfrak{g}_T$$

which translates into 1.7 times the cost of locally computing a pairing on BLS24-509 according to pairing group operations benchmarked with RELIC [4]. Despite two decades of research since the pioneering work of Girault-Lefranc [52] and Chevallier-Mames et al. [36], *no prior work has demonstrated significant efficiency improvements over local pairing evaluation.* This negative result stems from two fundamental limitations in existing approaches:

**The one-shot approach.** All existing protocols treat each delegation as an independent execution requiring a fresh offline preprocessing phase. In many cases, this offline phase itself involves computing one or more pairings, immediately negating any efficiency advantage. Even protocols with lighter preprocessing must compensate this cost over a single delegation but fail, making the overall client workload comparable to—or exceeding—local evaluation. Moreover, real-world applications often require multiple pairing evaluations over time (e.g., a hardware wallet repeatedly verifying zero-knowledge proofs), but precisely because of the one-shot nature of current protocols, this kind of composability is lacking.

**The information-theoretic security level.** To achieve verifiability, existing protocols rely on information-theoretic arguments that provide security against computationally unbounded adversaries (often called *unconditional security*). While theoretically appealing, this strong security notion comes at a steep efficiency cost: the client must perform sufficient computation to information-theoretically bind the server to correct outputs. Chevallier-Mames et al. [37] explicitly identified this trade-off, noting that *"an interesting research direction would be to further optimize the protocols by trading off unconditional security against computational security"*. Yet this direction has remained unexplored until now.

The goal of this thesis is to study the design of pairing delegation protocols that improve client-side performance and identify sufficient conditions for this to be done. In this context, three main research questions can be formulated:

### Research Questions

RQ1 *How can pairing delegation be achieved sequentially, and in a way that the involved computational costs are amortized over several delegations?*

RQ2 *What adversarial computational assumption can be formalized as an alternative to proving security against unbounded adversaries?*

RQ3 *How can the answers from* RQ1 *and* RQ2 *be leveraged to settings that require input privacy?*

## Scientific Contributions

Chapters 1, 2, and 3 include contributions from the paper [66]:

*"That's AmorE: Amortized Efficiency for Pairing Delegation"*
Authors: Adrián P. Keilty, Diego F. Aranha, Elena Pagnin, and Francisco Rodríguez-Henríquez.
Appeared in: Advances in Cryptology - CRYPTO 2025 .

The material in chapters 4, 5 and 6 has not been peer-reviewed at the time of writing this thesis.

### Background (chapter 1)

This part provides notation, preliminary notions, a unified framework for the upcoming systematization (chapter 2), and security experiments for verifiability and input privacy of a delegation protocol. The Tsang et al.'s taxonomy [85] is introduced and extended to classify existing constructions based on input variability ($\mathsf{V} \rightarrow$ variable, $\mathsf{C} \rightarrow$ constant) and secrecy ($\mathsf{P} \rightarrow$ public, $\mathsf{S} \rightarrow$ secret), e.g. $\mathsf{PVPV}$ (both inputs public and variable), and a further distinction is made according to whether protocols support single or batch pairing delegation. The following sections are an adaptation from [66]: 1.2 (Motivation for Pairing Delegation and Application Scenarios), 1.3 (General Delegation Framework) and 1.4 (Single & Batch Pairing Delegation).

**Statement of Contributions.** Elena came up with the idea of using elliptic curve point distribution when formulating a new security experiment for scenarios in which one or both pairing inputs are presumably private. Stemming from this, I developed the notions of *Sampled Input (SI) verifiability* and *One-Way (OW) input privacy* for pairing delegation protocols (subsections 1.6.1 and 1.6.2). Additionally, I provided Lemma 1.6.4 in which it is proven that if a protocol does not conceal the output of the delegated pairing, then it cannot be input-private in the standard sense.

### Systematization of pairing delegation protocols (chapter 2)

This part presents a systematization and comprehensive survey of pairing delegation protocols in the two-party setting (client and single server). Protocols are reviewed in historical order within each classification. The sys-

tematization identifies two recently published protocols that claim both verifiability and high efficiency but are shown to be vulnerable to malleability attacks. An impossibility result that identifies necessary and sufficient conditions for this attack to succeed, and therefore compromises the verifiability of a protocol, is also presented. The following sections are an adaptation from [66]: 2.3 (Efficient but Malleable Pairing Delegation Protocols) and 2.4 (Securely Delegating a Pairing without Precomputation is Impossible).

**Statement of Contributions.** I discovered the malleability attacks presented in section 2.3. The impossibility result was initially formalized and proved by Elena, and I provided a simplified version afterwards (section 2.4).

### Amortized efficiency for pairing delegation with public inputs (chapter 3)

This chapter addresses research questions RQ1 and RQ2. In particular, it focuses on the design and analysis of a novel pairing delegation protocol that achieves, for the first time in the literature, significant concrete efficiency gains over local pairing evaluation. The key innovation is *amortized efficiency* over *sequential delegation*: rather than treating each delegation as an independent one-shot protocol requiring a fresh offline phase, KAPR25 [66, §5] is introduced as a unified framework that amortizes a one-time setup cost over multiple delegation rounds. This approach allows the client to choose between single or batch pairing delegations on-demand while avoiding the repeated offline phases that hindered previous design attempts. Additionally, by leveraging a computational security assumption rather than information-theoretic arguments, substantial efficiency gains are obtained in return. Apart from the sequential verifiability experiment $\mathbf{Exp}^{\mathsf{seq,ver}}_{\mathbf{\Pi},\mathcal{A}}$, the rest of the content of this chapter is included in [66].

**Statement of Contributions.** Elena and I formalized the notion of *amortized efficiency* in the setting of pairing delegation (Definition 3.1.2) and the security model for the sequential setting. I came up with the proposed construction that unifies single and batch delegation satisfying this property (section 3.2), a new computational assumption that allows a tunable tradeoff between security and efficiency, and the corresponding security proofs for the information theoretic and computational variants. (section 3.3).

### Amortized efficiency for pairing delegation with private inputs (chapter 4)

This chapter addresses research question RQ3. More specifically, it presents a novel way of converting a type-PVPV protocol into types SVSV, SVPV, and PVSV with minimal extra computation, and introduces new constructions for single and batch delegation that satisfy SI-verification and IND-privacy. The proving techniques are essentially the same as those introduced in [66] but leverage the security parameter $\lambda$ instead of the efficiency parameter $\varphi$. The proposed protocols are also shown to have better performance than previous proposals.

### Performance results (chapter 5)

This chapter presents a performance comparison between existing protocols and the ones introduced in this thesis. For the comparison with prior work, concrete costs are derived from symbolic expressions using bilinear group operation benchmarks obtained with RELIC. Additionally, a direct comparison between the KAPR25 and K26 protocols is presented using actual benchmark measurements from the implementations, providing empirical validation of the theoretical efficiency gains.

### Future directions (chapter 6)

This final part encourages research towards public verifiability and product pairing delegation.

# Bibliography


[1] M. Abdalla, E. Kiltz, and G. Neven. Generalized key delegation for hierarchical identity-based encryption. In J. Biskup and J. López, editors, *Computer Security – ESORICS 2007*, pages 139–154, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg.

[2] S. S. Al-Riyami and K. G. Paterson. Certificateless public key cryptography. In C.-S. Laih, editor, *Advances in Cryptology – ASIACRYPT 2003*, volume 2894 of *Lecture Notes in Computer Science*, pages 452–473, Berlin, Heidelberg, 2003. Springer.

[3] M. Ambrona, M. Beunardeau, A.-L. Schmitt, and R. R. Toledo. aPlonK: Aggregated PlonK from multi-polynomial commitment schemes. In J. Shikata and H. Kuzuno, editors, *IWSEC 23: 18th International Workshop on Security, Advances in Information and Computer Security*, volume 14128 of *Lecture Notes in Computer Science*, pages 195–213, Yokohama, Japan, August 29–31, 2023. Springer, Cham, Switzerland.

[4] D. F. Aranha, C. P. L. Gouvêa, T. Markmann, R. S. Wahby, and K. Liao. RELIC is an Efficient LIbrary for Cryptography. `https://github.com/relic-toolkit/relic`.

[5] D. F. Aranha and E. Pagnin. The simplest multi-key linearly homomorphic signature scheme. In *LATINCRYPT*, volume 11774 of *LNCS*, pages 280–300. Springer, 2019.

[6] D. F. Aranha, E. Pagnin, and F. Rodríguez-Henríquez. LOVE a pairing. In *LATINCRYPT*, volume 12912 of *Lecture Notes in Computer Science*, pages 320–340. Springer, 2021.

[7] J. Baek and Y. Zheng. Identity-based threshold decryption. In F. Bao, R. Deng, and J. Zhou, editors, *Public Key Cryptography – PKC 2004*, pages 262–276, Berlin, Heidelberg, 2004. Springer Berlin Heidelberg.

[8] R. Barbulescu and S. Duquesne. Updating key size estimations for pairings. *J. Cryptol.*, 32(4):1298–1336, 2019.

[9] P. S. L. M. Barreto, B. Lynn, and M. Scott. Constructing elliptic curves with prescribed embedding degrees. In *SCN*, volume 2576 of *LNCS*, pages 257–267. Springer, 2002.

[10] P. S. L. M. Barreto, B. Lynn, and M. Scott. Efficient implementation of pairing-based cryptosystems. *J. Cryptol.*, 17(4):321–334, 2004.

[11] P. S. L. M. Barreto and M. Naehrig. Pairing-friendly elliptic curves of prime order. In *SAC*, volume 3897 of *LNCS*, pages 319–331. Springer, 2005.

[12] R. Barua, R. Dutta, and P. Sarkar. Extending joux's protocol to multi party key agreement (extended abstract). In T. Johansson and S. Maitra, editors, *Progress in Cryptology - INDOCRYPT 2003, 4th International Conference on Cryptology in India, New Delhi, India, December 8-10, 2003, Proceedings*, volume 2904 of *Lecture Notes in Computer Science*, pages 205–217. Springer, 2003.

[13] E. Ben-Sasson, A. Chiesa, D. Genkin, E. Tromer, and M. Virza. SNARKs for C: Verifying program executions succinctly and in zero knowledge. In R. Canetti and J. A. Garay, editors, *Advances in Cryptology – CRYPTO 2013, Part II*, volume 8043 of *Lecture Notes in Computer Science*, pages 90–108, Santa Barbara, CA, USA, August 18–22, 2013. Springer Berlin Heidelberg, Germany.

[14] D. J. Bernstein and T. Lange. Faster addition and doubling on elliptic curves. In K. Kurosawa, editor, *Advances in Cryptology – ASIACRYPT 2007*, volume 4833 of *Lecture Notes in Computer Science*, pages 29–50, Kuching, Malaysia, December 2–6, 2007. Springer Berlin Heidelberg, Germany.

[15] J. Bethencourt, A. Sahai, and B. Waters. Ciphertext-policy attribute-based encryption. In *2007 IEEE Symposium on Security and Privacy*, pages 321–334. IEEE, 2007.

[16] A. Boldyreva. Threshold signatures, multisignatures and blind signatures based on the gap-diffie-hellman-group signature scheme. In Y. Desmedt, editor, *Public Key Cryptography - PKC 2003, 6th International Workshop on Theory and Practice in Public Key Cryptography*, volume 2567 of *Lecture Notes in Computer Science*, pages 31–46. Springer, 2003.

[17] D. Boneh and X. Boyen. Efficient selective-id secure identity-based encryption without random oracles. In C. Cachin and J. L. Camenisch, editors, *Advances in Cryptology - EUROCRYPT 2004*, pages 223–238, Berlin, Heidelberg, 2004. Springer Berlin Heidelberg.

[18] D. Boneh, X. Boyen, and E.-J. Goh. Hierarchical identity based encryption with constant size ciphertext. In R. Cramer, editor, *Advances in Cryptology – EUROCRYPT 2005*, pages 440–456, Berlin, Heidelberg, 2005. Springer.

[19] D. Boneh, G. D. Crescenzo, R. Ostrovsky, and G. Persiano. Public key encryption with keyword search. In C. Cachin and J. Camenisch, editors, *Advances in Cryptology - EUROCRYPT 2004, International Conference on the Theory and Applications of Cryptographic Techniques*, volume 3027 of *Lecture Notes in Computer Science*, pages 506–522. Springer, 2004.

[20] D. Boneh, M. Drijvers, and G. Neven. Compact multi-signatures for smaller blockchains. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 435–464. Springer, 2018.

[21] D. Boneh and M. K. Franklin. Identity-Based Encryption from the Weil Pairing. In *CRYPTO*, volume 2139 of *LNCS*, pages 213–229. Springer, 2001.

[22] D. Boneh, C. Gentry, B. Lynn, and H. Shacham. Aggregate and verifiably encrypted signatures from bilinear maps. In E. Biham, editor, *Advances in Cryptology — EUROCRYPT 2003*, pages 416–432, Berlin, Heidelberg, 2003. Springer Berlin Heidelberg.

[23] D. Boneh, E.-J. Goh, and K. Nissim. Evaluating 2-dnf formulas on ci-

phertexts. In J. Kilian, editor, *Theory of Cryptography*, pages 325–341, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg.

[24] D. Boneh, B. Lynn, and H. Shacham. Short Signatures from the Weil Pairing. *J. Cryptol.*, 17(4):297–319, 2004.

[25] D. Boneh and B. Waters. A fully collusion resistant broadcast, trace, and revoke system. In *Proceedings of the 13th ACM Conference on Computer and Communications Security*, CCS '06, page 211–220, New York, NY, USA, 2006. Association for Computing Machinery.

[26] D. Boneh and B. Waters. Conjunctive, subset, and range queries on encrypted data. In S. P. Vadhan, editor, *Theory of Cryptography*, pages 535–554, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg.

[27] J. W. Bos, C. Costello, and M. Naehrig. Exponentiating in pairing groups. In *SAC*, volume 8282 of *LNCS*, pages 438–455. Springer, 2013.

[28] X. Boyen and B. Waters. Full-domain subgroup hiding and constant-size group signatures. In T. Okamoto and X. Wang, editors, *Public Key Cryptography – PKC 2007*, pages 1–15, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg.

[29] S. Canard, J. Devigne, and O. Sanders. Delegating a pairing can be both secure and efficient. In *ACNS*, volume 8479 of *LNCS*, pages 549–565. Springer, 2014.

[30] R. Canetti, S. Halevi, and J. Katz. A forward-secure public-key encryption scheme. In E. Biham, editor, *Advances in Cryptology — EUROCRYPT 2003*, pages 255–271, Berlin, Heidelberg, 2003. Springer Berlin Heidelberg.

[31] D. Catalano, D. Fiore, and E. Giunta. Efficient and universally composable single secret leader election from pairings. In A. Boldyreva and V. Kolesnikov, editors, *PKC 2023: 26th International Conference on Theory and Practice of Public Key Cryptography, Part I*, volume 13940 of *Lecture Notes in Computer Science*, pages 471–499, Atlanta, GA, USA, May 7–10, 2023. Springer, Cham, Switzerland.

[32] M. Chase and A. Lysyanskaya. Simulatable vrfs with applications to multi-theorem nizk. In A. Menezes, editor, *Advances in Cryptology -*

*CRYPTO 2007*, pages 303–322, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg.

[33] L. Chen, Z. Cheng, and N. P. Smart. Identity-based key agreement protocols from pairings. *International Journal of Information Security*, 6(4):213–241, 2007.

[34] L. Chen and C. Kudla. Identity based authenticated key agreement protocols from pairings. In *16th IEEE Computer Security Foundations Workshop, 2003. Proceedings.*, pages 219–233, 2003.

[35] X. Chen, W. Susilo, J. Li, D. S. Wong, J. Ma, S. Tang, and Q. Tang. Efficient algorithms for secure outsourcing of bilinear pairings. *Theoretical Computer Science*, 562:112–121, 2015.

[36] B. Chevallier-Mames, J. Coron, N. McCullagh, D. Naccache, and M. Scott. Secure delegation of elliptic-curve pairing. *IACR Cryptol. ePrint Arch.*, page 150, 2005.

[37] B. Chevallier-Mames, J. Coron, N. McCullagh, D. Naccache, and M. Scott. Secure delegation of elliptic-curve pairing. In *CARDIS*, volume 6035 of *LNCS*, pages 24–35. Springer, 2010.

[38] S. S. M. Chow. Verifiable pairing and its applications. In C. H. Lim and M. Yung, editors, *Information Security Applications*, pages 170–187, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg.

[39] S. S. M. Chow, C. Boyd, and J. M. G. Nieto. Security-mediated certificateless cryptography. In M. Yung, Y. Dodis, A. Kiayias, and T. Malkin, editors, *Public Key Cryptography - PKC 2006*, pages 508–524, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.

[40] S. S. M. Chow and K.-K. R. Choo. Strongly-secure identity-based key agreement and anonymous extension. In J. A. Garay, A. K. Lenstra, M. Mambo, and R. Peralta, editors, *Information Security*, pages 203–220, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg.

[41] G. D. Crescenzo, M. Khodjaeva, and D. D. M. Caro. Single-server batch delegation of variable-input pairings with unbounded client lifetime. In *European Symposium on Research in Computer Security*, pages 233–255. Springer, 2023.

[42] G. D. Crescenzo, M. Khodjaeva, D. Kahrobaei, and V. Shpilrain. Secure and efficient delegation of pairings with online inputs. In *CARDIS*, volume 12609 of *LNCS*, pages 84–99. Springer, 2020.

[43] A. W. Dent. A survey of certificateless encryption schemes and security models. *International Journal of Information Security*, 7(5):349–377, 2008.

[44] G. Di Crescenzo, M. Khodjaeva, D. Kahrobaei, and V. Shpilrain. Secure and efficient delegation of elliptic-curve pairing. In M. Conti, J. Zhou, E. Casalicchio, and A. Spognardi, editors, *Applied Cryptography and Network Security*, pages 45–66, Cham, 2020. Springer International Publishing.

[45] W. Diffie and M. E. Hellman. Exhaustive cryptanalysis of the NBS data encryption standard. *COMPUTER*, pages 74–84, 1977.

[46] Y. Dodis and A. Yampolskiy. A verifiable random function with short proofs and keys. In S. Vaudenay, editor, *Public Key Cryptography - PKC 2005*, pages 416–431, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg.

[47] G. Fuchsbauer, E. Kiltz, and J. Loss. The algebraic group model and its applications. In H. Shacham and A. Boldyreva, editors, *Advances in Cryptology – CRYPTO 2018, Part II*, volume 10992 of *Lecture Notes in Computer Science*, pages 33–62, Santa Barbara, CA, USA, August 19–23, 2018. Springer, Cham, Switzerland.

[48] N. Gailly, M. Maller, and A. Nitulescu. SnarkPack: Practical SNARK aggregation. In I. Eyal and J. A. Garay, editors, *FC 2022: 26th International Conference on Financial Cryptography and Data Security*, volume 13411 of *Lecture Notes in Computer Science*, pages 203–229, Grenada, May 2–6, 2022. Springer, Cham, Switzerland.

[49] S. Garg, A. Jain, P. Mukherjee, R. Sinha, M. Wang, and Y. Zhang. hints: Threshold signatures with silent setup. In *2024 IEEE Symposium on Security and Privacy (SP)*, pages 3034–3052. IEEE, 2024.

[50] R. Gennaro, C. Gentry, and B. Parno. Non-interactive verifiable computing: Outsourcing computation to untrusted workers. In *CRYPTO*,

volume 6223 of *Lecture Notes in Computer Science*, pages 465–482. Springer, 2010.

[51] C. Gentry. Practical identity-based encryption without random oracles. In S. Vaudenay, editor, *Advances in Cryptology - EUROCRYPT 2006*, pages 445–464, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.

[52] M. Girault and D. Lefranc. Server-aided verification: Theory and practice. In *ASIACRYPT*, volume 3788 of *LNCS*, pages 605–623. Springer, 2005.

[53] V. Goyal, O. Pandey, A. Sahai, and B. Waters. Attribute-based encryption for fine-grained access control of encrypted data. In A. Menezes, editor, *Advances in Cryptology – CRYPTO 2006*, pages 89–98, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.

[54] J. Groth. On the size of pairing-based non-interactive arguments. In *EUROCRYPT (2)*, volume 9666 of *LNCS*, pages 305–326. Springer, 2016.

[55] J. Groth, R. Ostrovsky, and A. Sahai. Perfect non-interactive zero knowledge for np. In S. Vaudenay, editor, *Advances in Cryptology - EUROCRYPT 2006*, pages 339–358, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.

[56] A. Guillevic and D. Vergnaud. Algorithms for outsourcing pairing computation. In *CARDIS*, volume 8968 of *LNCS*, pages 193–211. Springer, 2014.

[57] D. Harnik and M. Naor. On everlasting security in the hybrid bounded storage model. In M. Bugliesi, B. Preneel, V. Sassone, and I. Wegener, editors, *ICALP 2006: 33rd International Colloquium on Automata, Languages and Programming, Part II*, volume 4052 of *Lecture Notes in Computer Science*, pages 192–203, Venice, Italy, July 10–14, 2006. Springer Berlin Heidelberg, Germany.

[58] F. Hess. Efficient identity based signature schemes based on pairings. In K. Nyberg and H. Heys, editors, *Selected Areas in Cryptography*, pages 310–324, Berlin, Heidelberg, 2003. Springer Berlin Heidelberg.

[59] A. Joux. A one round protocol for tripartite diffie-hellman. *J. Cryptology*, 17:263–276, 2004.

[60] E. J. Kachisa, E. F. Schaefer, and M. Scott. Constructing brezing-weng pairing-friendly elliptic curves using elements in the cyclotomic field. In S. D. Galbraith and K. G. Paterson, editors, *Pairing-Based Cryptography – Pairing 2008*, pages 126–135, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg.

[61] Ö. Kalkar, M. S. Kiraz, İ. Sertkaya, and O. Uzunkol. 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*, pages 155–164. Springer, 2018.

[62] O. Kalkar, I. Sertkaya, and S. Tutdere. On the batch outsourcing of pairing computations. *The Computer Journal*, 66(10):2437–2446, 2023.

[63] B. G. Kang, M. S. Lee, and J. H. Park. Efficient delegation of pairing computation. *IACR Cryptol. ePrint Arch.*, 2005:259, 2005.

[64] A. Kate, G. M. Zaverucha, and I. Goldberg. Pairing-based onion routing. In N. Borisov and P. Golle, editors, *PET 2007: 7th International Symposium on Privacy Enhancing Technologies*, volume 4776 of *Lecture Notes in Computer Science*, pages 95–112, Ottawa, Canada, June 20–22, 2007. Springer Berlin Heidelberg, Germany.

[65] A. Kate, G. M. Zaverucha, and I. Goldberg. Constant-size commitments to polynomials and their applications. In M. Abe, editor, *Advances in Cryptology - ASIACRYPT 2010 - 16th International Conference on the Theory and Application of Cryptology and Information Security, Singapore, December 5-9, 2010. Proceedings*, volume 6477 of *Lecture Notes in Computer Science*, pages 177–194. Springer, 2010.

[66] A. P. Keilty, D. F. Aranha, E. Pagnin, and F. Rodríguez-Henríquez. That's AmorE: Amortized efficiency for pairing delegation. In Y. Tauman Kalai and S. F. Kamara, editors, *Advances in Cryptology – CRYPTO 2025*, pages 211–246, Cham, 2025. Springer Nature Switzerland.

[67] M. Khodjaeva. and G. Di Crescenzo. On single-server delegation without precomputation. In *Proceedings of the 20th International Conference on Security and Cryptography - SECRYPT*, pages 540–547. INSTICC, SciTePress, 2023.

[68] T. Kim and R. Barbulescu. Extended tower number field sieve: A new complexity for the medium prime case. In *CRYPTO (1)*, volume 9814 of *LNCS*, pages 543–571. Springer, 2016.

[69] M. S. Kiraz, I. Sertkaya, O. Uzunkol, and O. Arabaci. More efficient secure outsourcing methods for bilinear maps. 10 2015.

[70] K. Kurosawa and S.-H. Heng. The power of identification schemes. In M. Yung, Y. Dodis, A. Kiayias, and T. Malkin, editors, *Public Key Cryptography - PKC 2006*, pages 364–377, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.

[71] J. K. Liu, M. H. Au, and W. Susilo. Self-generated-certificate public key cryptography and certificateless signature/encryption scheme in the standard model. In *Proceedings of the 2nd ACM Symposium on Information, Computer and Communications Security*, ASIACCS '07, pages 273–283. ACM Press, 2007.

[72] N. M. Mbang, D. F. Aranha, and E. Fouotsa. Computing the Optimal Ate pairing over elliptic curves with embedding degrees 54 and 48 at the 256-bit security level. *Int. J. Appl. Cryptogr.*, 4(1):45–59, 2020.

[73] T. Mefenza and D. Vergnaud. Verifiable outsourcing of pairing computations. `https://tinyurl.com/Batch-Mefenza-Pairings`, 2018.

[74] V. S. Miller. The Weil Pairing, and Its Efficient Calculation. *J. Cryptol.*, 17(4):235–261, 2004.

[75] E. Pagnin, A. Mitrokotsa, and K. Tanaka. Anonymous single-round server-aided verification. In *LATINCRYPT*, volume 11368 of *LNCS*, pages 23–43. Springer, 2017.

[76] B. Parno, J. Howell, C. Gentry, and M. Raykova. Pinocchio: nearly practical verifiable computation. *Commun. ACM*, 59(2):103–112, 2016.

[77] A. Sahai and B. Waters. Fuzzy identity-based encryption. In D. Boneh, editor, *Advances in Cryptology - EUROCRYPT 2005*, volume 3494 of *Lecture Notes in Computer Science*, pages 457–473. Springer, 2005.

[78] R. Sakai, K. Ohgishi, and M. Kasahara. Cryptosystems based on pairing. In *2000 Symposium on Cryptography and Information Security (SCIS2000), Okinawa, Japan*, pages 26–28, Jan. 2000.

[79] M. Scott. Unbalancing pairing-based key exchange protocols. Cryptology ePrint Archive, Paper 2013/688, 2013.

[80] J. Shao and G. Wei. Secure outsourced computation in connected vehicular cloud computing. *IEEE Network*, 32(3):36–41, 2018.

[81] E. Shi, J. Bethencourt, T.-H. H. Chan, D. Song, and A. Perrig. Multidimensional range query over encrypted data. In *2007 IEEE Symposium on Security and Privacy (SP '07)*, pages 350–364, 2007.

[82] V. Shoup. Lower bounds for discrete logarithms and related problems. In W. Fumy, editor, *Advances in Cryptology — EUROCRYPT '97*, pages 256–266, Berlin, Heidelberg, 1997. Springer Berlin Heidelberg.

[83] H. Tian, F. Zhang, and K. Ren. Secure bilinear pairing outsourcing made more efficient and flexible. In F. Bao, S. Miller, J. Zhou, and G.-J. Ahn, editors, *ASIACCS 15: 10th ACM Symposium on Information, Computer and Communications Security*, pages 417–426, Singapore, April 14–17, 2015. ACM Press.

[84] L. Tong, J. Yu, and H. Zhang. Secure outsourcing algorithm for bilinear pairings without pre-computation. In *2019 IEEE Conference on Dependable and Secure Computing (DSC)*, pages 1–7, 2019.

[85] P. P. Tsang, S. S. M. Chow, and S. W. Smith. Batch pairing delegation. In *IWSEC*, volume 4752 of *LNCS*, pages 74–90. Springer, 2007.

[86] F. Vercauteren. Optimal pairings. *IEEE Trans. Inf. Theory*, 56(1):455–461, 2010.

[87] Z. Wang. A new construction of the server-aided verification signature scheme. *Mathematical and computer modelling*, 55(1-2):97–101, 2012.

[88] W. Wu, Y. Mu, W. Susilo, and X. Huang. Provably secure server-aided verification signatures. *Computers & Mathematics with Applications*, 61(7):1705–1723, 2011.

[89] D. Yao, N. Fazio, Y. Dodis, and A. Lysyanskaya. Id-based encryption for complex hierarchies with applications to forward security and broadcast encryption. In *Proceedings of the 11th ACM Conference on Computer and Communications Security*, CCS '04, page 354–363, New York, NY, USA, 2004. Association for Computing Machinery.

[90] E. Zavattoni, L. J. D. Perez, S. Mitsunari, A. H. Sánchez-Ramírez, T. Teruya, and F. Rodríguez-Henríquez. Software implementation of an attribute-based encryption scheme. *IEEE Trans. Computers*, 64(5):1429–1441, 2015.

[91] F. Zhang, R. Safavi-Naini, and W. Susilo. An efficient signature scheme from bilinear pairings and its applications. In F. Bao, R. Deng, and J. Zhou, editors, *PKC 2004: 7th International Workshop on Theory and Practice in Public Key Cryptography*, volume 2947 of *Lecture Notes in Computer Science*, pages 277–290, Singapore, March 1–4, 2004. Springer Berlin Heidelberg, Germany.

# 1

# Background

This chapter establishes the foundational framework for pairing delegation protocols. It begins with mathematical preliminaries and computational motivation, introduces the Tsang et al. taxonomy for classifying delegation protocols based on input properties, and develops formal security models for verifiability and input-privacy. These security notions are later revisited and generalized to the sequential delegation setting in chapter 3.

## 1.1 Preliminaries

### 1.1.1 $\lambda$-Min-Entropy Distributions

Let $\mathcal{X}$ be a finite set and $\mathcal{D}\colon \mathcal{X} \to [0,1]$ a probability distribution over $\mathcal{X}$. The *min-entropy* of $\mathcal{D}$ is defined as

$$H_\infty(\mathcal{D}) \;:=\; -\log_2\Big(\max_{x\in\mathcal{X}} \mathcal{D}(x)\Big).$$

A distribution $\mathcal{D}$ has $\lambda$ *bits of min-entropy* if

$$H_\infty(\mathcal{D}) \geq \lambda \quad \Leftrightarrow \quad \max_{x\in\mathcal{X}} \mathcal{D}(x) \leq 2^{-\lambda}.$$

In other words, a freshly sampled value $x \xleftarrow{\mathcal{D}} \mathcal{X}$ is correctly guessed with probability less or equal than $2^{-\lambda}$. The set of all distributions over $\mathcal{X}$ with at least $\lambda$ bits of min-entropy is denoted by

$$\mathsf{Dist}^{\lambda}_{\infty}(\mathcal{X}) \;:=\; \Big\{\mathcal{D} \in \mathsf{Dist}(\mathcal{X}) \,:\, H_\infty(\mathcal{D}) \geq \lambda\Big\}.$$

Let $\mathsf{Dist}^{\lambda}_{\infty}(\mathcal{X}\times\mathcal{Y})\big|_{\mathsf{marg}}$ denote the set of joint distributions over $\mathcal{X}\times\mathcal{Y}$ with $\lambda$ min-entropy whose marginal distributions also have $\lambda$ min-entropy, i.e.,

$$\mathsf{Dist}^{\lambda}_{\infty}(\mathcal{X}\times\mathcal{Y})\big|_{\mathsf{marg}} \;=\; \Big\{\mathcal{D} \in \mathsf{Dist}^{\lambda}_{\infty}(\mathcal{X}\times\mathcal{Y}) \;\Big|\; \mathcal{D}_X \in \mathsf{Dist}^{\lambda}_{\infty}(\mathcal{X}),\, \mathcal{D}_Y \in \mathsf{Dist}^{\lambda}_{\infty}(\mathcal{Y})\Big\}$$

### 1.1.2 One Way Functions

Let $f\colon \{0,1\}^* \to \{0,1\}^*$ be a function. For any adversary, $\mathcal{A}$ and security parameter $\lambda$, the one-way experiment $\mathbf{Exp}^{\mathsf{OW}}_{f,\mathcal{A}}(\lambda)$ is defined as follows:

1. $x \xleftarrow{\$} \{0,1\}^\lambda$, and $y := f(x)$.
2. $\mathcal{A}$ is given $y$ as input, and outputs $x^*$.
3. If $f(x^*) = y$ output 1, else output 0.

$f$ is one-way if it is computable in polynomial time and for every probabilistic polynomial-time adversary $\mathcal{A}$,

$$Pr[\mathbf{Exp}^{\mathsf{OW}}_{f,\mathcal{A}}(\lambda) = 1] \leq \mathsf{negl}(\lambda).$$

### 1.1.3 The Algebraic Group Model

**The Generic Group Model:** The Generic Group Model (GGM) [82] is an idealized computational model where adversaries can only perform group operations without exploiting any specific properties of the group representation. In the GGM, group elements are represented by random labels, and the adversary can only manipulate these elements through a group operation oracle. This abstraction ensures that any algorithm breaking a cryptographic scheme in the GGM must do so using only the algebraic structure of the group, without relying on implementation details. While the GGM provides strong security guarantees, it is often considered overly restrictive since real-world adversaries have access to concrete group representations.

**The Algebraic Group Model:** The Algebraic Group Model (AGM), introduced by Fuchsbauer, Kiltz and Loss [47], is a computational model that lies between the GGM and the standard model. The AGM relaxes the restrictions of the GGM by allowing adversaries to exploit the concrete representation of group elements, while maintaining the requirement that any group element output by the adversary must be efficiently computable from previously received elements. This makes the AGM less restrictive than the GGM—an adversary can use the bit representation of group elements—yet more structured than the standard model, capturing the intuition that efficient adversaries must exploit the algebraic structure of the group.

**Definition 1.1.1** (Algebraic Adversary [47])**.** An adversary $\mathcal{A}$ is *algebraic* with respect to a group $(\mathbb{G}, \circ)$ if, whenever $\mathcal{A}$ outputs a group element $Z \in \mathbb{G}$, it also outputs a representation of $Z$ as a linear combination of all group elements it has received so far. Formally, if $\mathcal{A}$ has been given group elements

$X_1, \ldots, X_n \in \mathbb{G}$, then whenever $\mathcal{A}$ outputs $Z \in \mathbb{G}$, it must also output coefficients $z_1, \ldots, z_n \in \mathbb{Z}_q$ such that

$$Z = \prod_{i=1}^{n} X_i^{z_i} = X_1^{z_i} \circ \cdots \circ X_n^{z_i},$$

where the group operation $\circ$ is typically addition or multiplication.

The key insight of the AGM is that any efficient adversary that outputs group elements must compute them using the group operation, and thus can efficiently express them in terms of previously received elements. This model excludes adversaries that might exploit specific encodings of group elements or structural weaknesses in particular implementations, while still being more realistic than the GGM by allowing adversaries to exploit algebraic relationships.

In the context of pairing-based cryptography, an algebraic adversary operating on the groups $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T)$ must provide representations for new element outputs in any of these groups, i.e., $a = \log_P(A)$ serves as a representation for $A = [a]P \in \mathbb{G}_1$ as well as for $e(A, Q) = \gamma_T^a \in \mathbb{G}_T$. Naturally, if the adversary has already received the group elements $(A, B) \in \mathbb{G}_1 \times \mathbb{G}_2$, then it is allowed to output the evaluated pairing $e(A, B)$ and use $(A, B)$ as a representation for this value. The AGM will be particularly useful for analyzing the security of the proposed construction in section 3.2 as will become apparent in the proofs of section 3.3.

### 1.1.4 Bilinear Pairings

Let $(\mathbb{G}_1, +)$, $(\mathbb{G}_2, +)$ and $(\mathbb{G}_T, \cdot)$ be cyclic groups of prime order $q$. A cryptographic pairing is a bilinear map $e : \mathbb{G}_1 \times \mathbb{G}_2 \mapsto \mathbb{G}_T$, satisfying the following properties:

- **Bilinearity:** For all $A \in \mathbb{G}_1$, $B \in \mathbb{G}_2$, and $r, t \in \mathbb{Z}_q$, it holds that

$$e([r]A, [t]B) = e(A, B)^{r \cdot t}.$$

- **Non-degeneracy:** If $P$ and $Q$ generate $\mathbb{G}_1$ and $\mathbb{G}_2$, then $e(P, Q)$ generates $\mathbb{G}_T$. Equivalently, $e(P, Q) \neq 1_{\mathbb{G}_T}$ for all $P \neq 1_{\mathbb{G}_1}$ and $Q \neq 1_{\mathbb{G}_2}$.

- **Efficient computability:** There exists a polynomial-time algorithm to compute $e(A, B)$ for any $A \in \mathbb{G}_1$ and $B \in \mathbb{G}_2$.

In practical implementations, cryptographic pairings are instantiated using pairing-friendly elliptic curves $E$ defined over finite fields $\mathbb{F}_p$, where $p$ is a large prime. Let $E(\mathbb{F}_p)$ denote the set of points satisfying the elliptic curve equation together with the point at infinity. The groups $\mathbb{G}_1$ and $\mathbb{G}_2$ are order-$q$ subgroups of $E(\mathbb{F}_p)$ and $E(\mathbb{F}_{p^k})$, respectively, where $k$ is the embedding degree—the smallest integer such that $q$ divides $p^k - 1$. The target group $\mathbb{G}_T$ is an order-$q$ subgroup of the multiplicative group $\mathbb{F}^*_{p^k}$.

Pairings are classified based on the relationship between $\mathbb{G}_1$ and $\mathbb{G}_2$. A pairing is of *Type 1* (symmetric) if $\mathbb{G}_1 = \mathbb{G}_2$, requiring a distortion map for non-degeneracy. A pairing is of *Type 2* if $\mathbb{G}_1 \neq \mathbb{G}_2$ but there exists an efficiently computable homomorphism between $\mathbb{G}_1$ and $\mathbb{G}_2$. A pairing is of *Type 3* (asymmetric) if $\mathbb{G}_1 \neq \mathbb{G}_2$ and there are no efficiently computable homomorphisms between the groups. Type 3 pairings typically employ a twist to compress elements in $\mathbb{G}_2$, offering better efficiency and are the most widely used in modern pairing-based cryptography.

The computation of pairings is based on Miller's algorithm [74], which evaluates a rational function through a double-and-add procedure analogous to scalar multiplication. The reduced Tate pairing extends Miller's algorithm with a final exponentiation step that raises the result to the power $(p^k-1)/q$, mapping it into the desired subgroup of $q$-th roots of unity.

**Pairing-friendly curves and optimal pairings** The state of the art in pairing-based cryptography employs pairing-friendly curves—elliptic curves with small embedding degrees that enable efficient pairing computation [10]. Such curves are typically specified by parameterized polynomial formulae for the prime modulus $p$ and the prime order subgroup $q$, instantiated using seeds with low Hamming weight for efficiency.

After the Tower Number Field Sieve (TNFS) algorithm was proposed for solving the discrete logarithm in parameterized extension fields [68], Barreto-Naehrig (BN) curves [11] lost their performance advantage at the 128-bit security level. Currently, Barreto-Lynn-Scott (BLS) curves [9] offer the best performance: BLS12 curves for 128-bit security, BLS24 curves for 192-bit security [8], and BLS48 curves for 256-bit security [72].

For pairing computation, the Optimal Ate pairing [86] represents the state-of-the-art algorithm, minimizing the loop length in Miller's algorithm and thereby achieving optimal efficiency for pairing evaluation. Despite these advances, the computational cost of pairings remains significantly higher

than scalar multiplications in $\mathbb{G}_1$ and $\mathbb{G}_2$, and prohibitive for certain devices such as smartcards or hardware wallets. This motivates the study of secure pairing delegation protocols which are discussed in the next section.

## 1.2 Motivation for Pairing Delegation and Application Scenarios

Constructive cryptographic applications of bilinear pairings emerged around the year 2000, independently proposed by Sakai-Ohgishi-Kasahara (pairing-based cryptosystems [78]) and Joux (tripartite Diffie-Hellman [59]). Since then, many efficient and/or otherwise unrealized cryptographic constructions have been discovered, including: the first usable identity-based encryption scheme, by Boneh and Franklin [21], aggregate signatures, e.g., Boneh et al. [22], multiparty key agreement, e.g., Barua Dutta and Sarkar [12], short signatures, e.g., BLS [24], attribute-based encryption, e.g., Sahai and Waters [77], constant-size polynomial commitments, e.g., KZG [65], succinct non-interactive zero-knowledge proofs, e.g., Groth16 [54], compact multi-signatures [20]; and most recently SnarkPack [48] and hinTS [49].

**Computational Cost of Pairings:** Pairing-based protocols commonly rely on various ancillary building blocks, including membership tests, exponentiation of group elements in $\mathbb{G}_T$, scalar multiplication in $\mathbb{G}_1$ and $\mathbb{G}_2$ elliptic curves, and the hash-to-point primitive, among others. From Figure 1.1, one can see that the computation of a single bilinear pairing can be six to nine times more computationally costly than scalar multiplications in $\mathbb{G}_1$ and $\mathbb{G}_2$ respectively. Given this substantial computational cost discrepancy, it is not surprising that the concept of outsourcing pairing computations emerged early on in the development of pairing-based protocols, more than two decades ago, especially for scenarios involving computationally constrained devices.

**Pairing Delegation:** In essence, the primary objective of pairing delegation is to empower a computationally constrained device, referred to as the client, to securely offload pairing computations to a more potent yet potentially untrustworthy computational entity, known as the server. Given that the standard security model for practical applications assumes the possibility

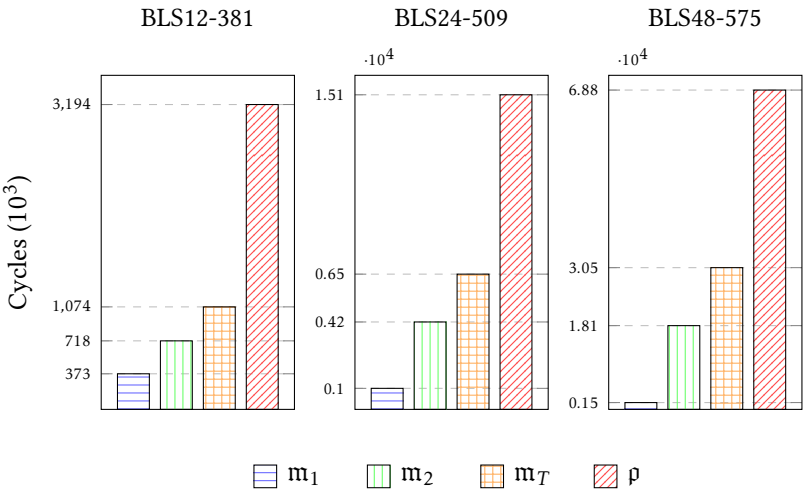


**Figure 1.1:** Comparison of computational costs for group operations across pairing-friendly curves (see [66, Table 2] for exact numbers). Each subplot shows the costs for a different curve with its own vertical scale. The histograms demonstrate that pairing computation ($\mathfrak{p}$) is significantly more expensive than scalar multiplications in groups $\mathbb{G}_1$, $\mathbb{G}_2$, and exponentiations in $\mathbb{G}_T$, with the cost gap widening at higher security levels.

of server dishonesty, any delegating protocol must incorporate mechanisms enabling the client to efficiently verify the pairing computation provided by the server.

**Candidate Clients:** Secure and efficient pairing delegation is well-suited for non-time-critical applications where client devices have reliable communication channels with the server. Examples include IoT devices, constrained Electronic Control Units (ECUs) connected via CAN bus [80], and USB-C hardware wallets.

IoT devices are typically constrained by battery and processing power. For devices lacking sufficient memory to store pairing libraries or processing power to handle pairing computations, delegation protocols offer significant advantages.

Hardware wallets represent another suitable client class. These devices

securely store cryptocurrency private keys offline and connect via USB to sign transactions. Many support anonymity-preserving cryptocurrencies like Zcash, which require computationally intensive zk-SNARK verifications. The popular Ledger Nano S Plus (2025) uses an ST33K1M5 chip with a Cortex-M35P processor that cannot handle pairing computations locally.

## 1.3 General Delegation Framework

Having established the formal models and security requirements, the general framework underlying all verifiable pairing delegation protocols is described below. All pairing delegation protocols proposed to date that include client verifiability rely on variations of the following mechanism:

The server is provided with two elliptic curve points, denoted as $A$ and $B$, belonging to groups $\mathbb{G}_1$ and $\mathbb{G}_2$, respectively, along with several group elements computed by the client, which will be required for verifying the server's computation. Subsequently, the server generates two outputs: the pairing $\gamma = e(A, B) \in \mathbb{G}_T$ and a proof $\pi \in \mathbb{G}_T$, aiding the client in verifying the honest and accurate computation of $\gamma$. Crucially, the verification process conducted by the client must be sufficiently lightweight to ensure that the delegation of pairing computations indeed yields computational savings. This is illustrated in Figure 1.2.

Naturally, the framework can be easily adapted to the batch case by replacing the single argument pair $(A, B)$ with the vector pair

$$\left(\vec{A}, \vec{B}\right) = ((A_1, \ldots, A_n), (B_1, \ldots, B_n)),$$

and the pairing output $\gamma$ with the vector $\vec{\gamma} = (\gamma_1, \ldots, \gamma_n)$.

Moreover, pairing delegation protocols may address additional requirements concerning the confidentiality of the evaluated points. If one or both of the delegated points are confidential, additional masking techniques must be implemented to prevent the server from learning something about their values.

## 1.4 Single & Batch Pairing Delegation

Models for single- and batch pairing delegation appear in a more or less formalized fashion in many works. Here, a unifying syntax that additionally

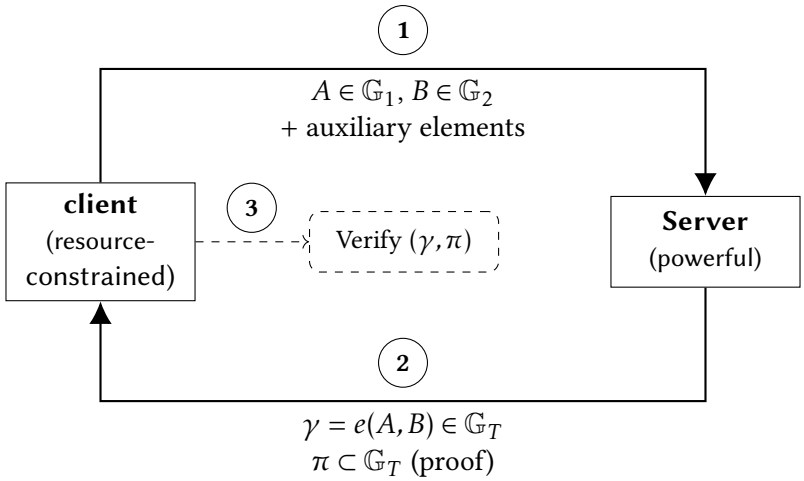


**Figure 1.2:** General framework for pairing delegation protocols. The client sends the points to be paired along with auxiliary elements (step 1), the Server returns the pairing result with a proof (step 2), and the client verifies the computation (step 3).

covers the sequential delegation setting introduced in [66] will be provided.

**Generic Remarks** Following established practice (e.g., [6, 62]), it is assumed that the communication channel between the client and the server is not vulnerable to integrity or replay attacks. Additionally, it is assumed that the client is always honest, while the server is untrusted.

### 1.4.1 Syntax for Pairing Delegation

With straightforward adaptations, the following definition allows describing all public input pairing delegation protocols in [85, 37, 29, 73, 42, 67, 41, 62]. The following definition presents a unified syntax that handles both single and batch pairing delegation, where single pairing delegation corresponds to the special case of batch size $\mathsf{M} = 1$.

**Definition 1.4.1** (Pairing Delegation Protocol)**.** A protocol $\mathbf{\Pi}$ for delegating pairing computations consists of five probabilistic polynomial time algorithms ($\mathsf{GlobalSetup}, \mathsf{OneTimeSetup}, \mathsf{Setup}, \mathsf{Compute}, \mathsf{Verify}$) with the following syntax:

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

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

$\mathsf{Setup}\left(\mathsf{sk}, \vec{A}, \vec{B}\right) \rightarrow (\mathsf{pub}, \mathtt{sec})$ 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 "$\mathsf{sk}$" and vectors of pairing arguments $\vec{A}, \vec{B} \in \mathbb{G}_1^{\mathsf{M}} \times \mathbb{G}_2^{\mathsf{M}}$ for some batch size $\mathsf{M} \geq 1$. It returns a public tuple "$\mathsf{pub}$" for the server, and the client's secret randomness "$\mathtt{sec}$" (possibly a vector) for the client.

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

$\mathsf{Verify}(\mathsf{sk}, \mathtt{sec}, \mathtt{out}) \rightarrow \mathtt{result}$ is a deterministic algorithm run by the client. It takes as input the client's long term secret $\mathsf{sk}$, the secret input "$\mathtt{sec}$", and the server's output "$\mathtt{out}$". It returns a value $\mathtt{result} \in \{\mathbb{G}_T^{\mathsf{M}} \cup \perp\}$, where $\mathsf{M}$ is the batch size from $\mathsf{Setup}$.

**Single vs. Batch Delegation:** The case $\mathsf{M} = 1$ corresponds to *single pairing delegation*, where the client delegates the computation of a single pairing $e(A, B)$ with $A \in \mathbb{G}_1$ and $B \in \mathbb{G}_2$. $\mathsf{M} > 1$ yields *batch pairing delegation*, where the client delegates the computation of $\mathsf{M}$ pairings $\{e(A_i, B_i)\}_{i=1}^{\mathsf{M}}$ simultaneously. This setting typically seeks more efficiency then the single one, by amortizing costs as the size of the batch grows.

**Intended usage:** In all pairing delegation protocols, the $\mathsf{GlobalSetup}$ 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., [37, 29, 67]. The procedures Setup, Compute, and Verify are executed in this sequence to perform one pairing delegation (single or batch). In [6, 42, 67] this triplet of algorithms 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).

## 1.5 Taxonomy for Pairing Delegation

Tsang et al. [85] proposed a taxonomy to classify pairing delegation protocols based on the properties of each pairing input. The classification considers whether each of the two input points is secret (S) or public (P), and whether it is variable (V) or constant (C). This two-dimensional categorization yields 16 possible combinations, e.g., SVPC denotes $A$ as *secret and variable*, $B$ as *public and constant*. Cases where both inputs are constant are naturally excluded, and symmetric pairs such as PVSC and SCPV are considered equivalent, reducing the taxonomy to seven distinct types. This work further distinguishes SVPV from PVSV, since scalar multiplication in $\mathbb{G}_2$ is significantly more expensive than in $\mathbb{G}_1$ (see Figure 1.1), making it generally cheaper to keep the first input secret rather than the second. This yields a total of eight delegation types.

Table 1.1 presents these eight types along with their applications in pairing-based cryptography. The table separates single pairing applications (e.g., ID-based encryption, key agreement, signature verification) from batch pairing applications that require computing multiple pairings simultaneously (e.g., BLS aggregate signatures with 2 pairings, Groth16 zk-SNARKs with 3 pairings, polynomial commitments, and general SNARKs with up to 21 pairings [13]).

## 1.6 Security Models

In the public-variable input setting (Type-PVPV delegation), protocols are required to satisfy two properties: correctness (also called completeness [29]) and security (also called verifiability [29], or $\epsilon_s$-result security [41]).

| Type | Single | Batch |
|---|---|---|
| PVPV | [38, 46, 7, 39, 23, 59] | [24, 16, 22, 54, 13, 48, 65, 3, 55, 31] |
| PVSV **or** SVPV | [26, 81, 51, 40] | [25] |
| SVSV | Basis for building delegation protocol variants in [37, 63]. | (no documented applications) |
| **SVSC** | [17, 34, 40] | Potential but undocumented application: verifying multiple signatures from the same signer. |
| **SVPC** | [70, 58] | [15] |
| **PVSC** | [21, 51, 78, 39, 43, 30, 89, 33, 34] | [15, 53, 1] |
| **PVPC** | [21, 89, 19, 32, 46, 58, 64, 85, 3] | [16, 22, 28, 38] |

**Table 1.1:** Tsang et al. taxonomy of pairing delegation types and their applications in pairing-based cryptography.
Convention: S = Secret, P = Public, V = Variable, C = Constant.
Applications: Verifiable pairing [38]; VRF [46, 32]; Ciphertext validity checking [7, 39, 51]; Doubly homomorphic encryption [23]; Tripartite key exchange [59]; BLS short signatures [24]; Multisignature/blind signature [16]; Aggregate/ring/verifiably encrypted signature [22]; Groth16 [54]; SNARKs [13]; SnarkPack [48]; Polynomial commitments [65, 3]; Homomorphic proof commitments [55]; Secret leader election [31]; Searchable encryption [26, 81]; ID-based key agreement [40, 34]; Trace-and-revoke broadcast [25]; Used as a basis for constructing different delegation protocol types [37, 63]; Inversion IBE [17]; ID-based identification [70]; ID-based signature [58]; Ciphertext-policy ABE [15]; IBE [21, 51, 78]; Certificateless encryption [39, 43]; Forward-secure encryption [30, 89]; Key agreement [33]; ABE [53]; ID-based broadcast encryption [1]; Public key encryption with keyword search [19]; Identity-based onion routing [64]; Trusted computing [85]; aPlonk inner product argument [3]; Group signature [28].

Correctness is defined in the natural way: if the server behaves honestly the output of the verification phase is the intended (vector of) pairing(s).

Security or verifiability, on the other hand, states that a malicious server should not be able to output an incorrect pairing value that will be accepted by the client with non-negligible probability over the security parameters. An output as such, that is accepted by the client is referred to as a *forgery*.

Additionally, in the secret-variable input setting (type SVSV, SVPV, and PVSV delegation), protocols are also required to satisfy input-privacy (also called secrecy [85, §3]). This property ensures that the server cannot learn information about the pairing inputs from the public value sent by the client. In the following, the security experiments capturing verifiability and input-privacy for one-shot pairing delegation protocols are formalized.

### 1.6.1 Verifiability

The verifiability experiment is parameterized by a privacy type

$$\mathsf{type} \in \{\mathsf{PVPV}, \mathsf{SVPV}, \mathsf{PVSV}, \mathsf{SVSV}\},$$

which determines the adversary's control over the pairing inputs. This unified formulation, denoted $\mathbf{Exp}^{\mathsf{ver}}_{\mathbf{\Pi},\mathcal{A}}$ and depicted in Figure 1.3, encompasses both chosen-input and sampled-input security notions:

**Chosen-Input (CI) Verifiability:** This notion corresponds to the case type = PVPV in Figure 1.3, where both inputs are public and the adversary has full control over them. This is a simplification of the security experiment for verifiable computation (VC) presented in [50], adapted to the pairing delegation framework and previously introduced by Aranha et al. in [6]. In this model, the adversary directly chooses both pairing inputs $(A, B)$.

**Sampled-Input (SI) Verifiability:** Introduced in this work, this notion generalizes CI-verifiability for type-PVPV delegation to the input-privacy types SVSV, SVPV and PVSV in a weaker but realistic way: For each input designated as secret by the protocol type, the adversary cannot choose its value directly but instead selects a $\lambda$-min-entropy distribution from which it is sampled. This notion captures scenarios where the inputs have inherent distributional structure that the adversary may know but cannot directly

control, while the actual input values remain hidden. For example, in type-SVSV protocols, the adversary chooses a jointly $\lambda$-min-entropy distribution $\mathcal{D} \in \mathsf{Dist}^{\lambda}_{\infty}(\mathbb{G}_1 \times \mathbb{G}_2)\Big|_{\mathsf{marg}}$, and the actual inputs $A$ and $B$ are sampled from its marginals. While weaker than CI-verifiability, SI-verifiability is a reasonable and practically relevant security goal for input-private delegation protocols, ensuring that distributional knowledge alone is insufficient to produce forgeries.

**Remark 1.6.1.** The $\lambda$ bits of min-entropy requirement that applies to the input-privacy types aligns with the setting of the one-way function experiment $\mathbf{Exp}^{\mathsf{OW}}_{f,\mathcal{A}}$ defined in subsection 1.1.2: given $f(x)$, the adversary needs to find a preimage $x^* \in f^{-1}(x)$ without learning the initial input $x \xleftarrow{\$} \{0,1\}^{\lambda}$ (which possesses $\lambda$ bits of min-entropy). This approach however, is not suitable for pairing-based protocols in which inputs may encode a predictable value, especially if plaintext is involved, e.g., *"Good morning"*. In these cases, it would be better to use salting in the hash-to-point algorithm and delegate under the PVPV type rather than delegate under a privacy type.

**Remark 1.6.2.** It is worth noticing that a polynomial-time adversary can only specify $\lambda$-min entropy distributions in a compact way, i.e., it cannot individually assign a probability to each element of a set of size $\geq 2^{\lambda}$. Instead, the adversary outputs a succinct parametrization consisting of polynomially many base elements with their associated probabilities. As an example, the adversary can specify a $\lambda$-min entropy distribution over $\mathbb{G}_{\iota}$ as a collection of points $\{Q_i\}_{i=1}^{n}$ to describe the larger set

$$\bigcup_{i \in [n]} \left\{ [t]\, Q_i \colon t \in \left[\!\!\left[ \frac{2^{\lambda}}{n} \right]\!\!\right] \right\},$$

for $\iota \in \{1,2\}$ and where each resulting point is assigned the probability $2^{-\lambda}$. This compact representation mirrors real world scenarios, e.g., in BLS signature verification (which can be delegated as type PVSV), the public key is $\mathsf{pk} = [x]Q$ for a secret value $x \xleftarrow{\$} \mathbb{Z}_q^*$ and a public generator $Q \in \mathbb{G}_2$. The induced distribution over $\mathbb{G}_2$ can be compactly specified as $\{Q\}$ which results in $\{[x]Q : x \in \mathbb{Z}_q^*\}$ and maintains $\log q \approx 2\lambda$ bits of min-entropy despite having exponential support.

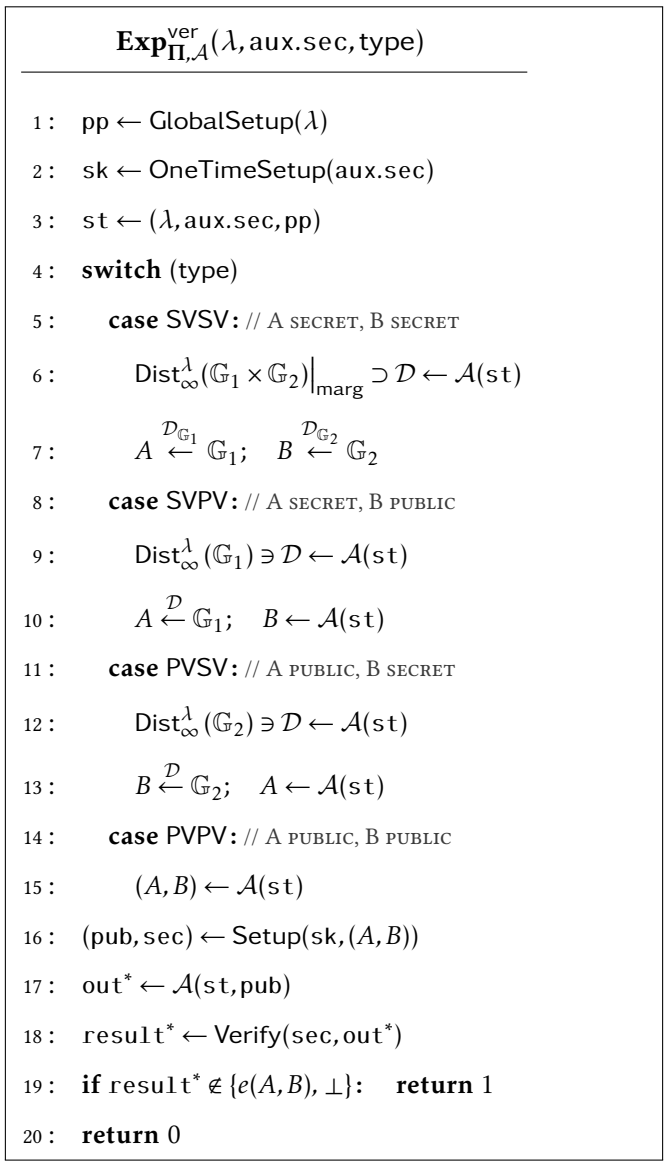


**Figure 1.3:** Verifiability experiment for pairing delegation protocols. If one of the inputs is secret, the adversary is allowed to choose a $\lambda$-min-entropy distribution from which it will be sampled. When type is set to PVPV, the regular chosen-input (CI) verifiability experiment is obtained.

In order to reach the winning condition, the adversary needs to produce an output $\mathtt{out}^*$ that is not rejected by the verification (i.e., $\mathtt{result}^* \neq \perp$) and that yields an incorrect value $\mathtt{result}^* \neq e(A, B)$. A protocol of type type is verifiable if any adversary has only negligible probability in the security parameters of winning the corresponding security experiment.

**Definition 1.6.3** (Verifiability for pairing delegation)**.** A protocol for pairing delegation of type $\mathsf{type} \in \{\mathsf{PVPV}, \mathsf{SVPV}, \mathsf{PVSV}, \mathsf{SVSV}\}$ is said to be *verifiable* against a class of adversaries, if for every adversary $\mathcal{A}$ in the class it holds that

$$\mathcal{P}\Big[\mathbf{Exp}^{\mathsf{ver}}_{\mathbf{\Pi},\mathcal{A}}(\lambda, \mathsf{aux.sec}, \mathsf{type}) = 1\Big] \leq \mathsf{negl}(\lambda) + \sum_{\delta \in \mathsf{aux.sec}} \mathsf{negl}(\delta)$$

where $\mathbf{Exp}^{\mathsf{ver}}_{\mathbf{\Pi},\mathcal{A}}(\lambda, \mathsf{aux.sec}, \mathsf{type})$ denotes the verifiability experiment shown in Figure 1.3. When $\mathsf{type} = \mathsf{PVPV}$, this corresponds to chosen-input (CI) verifiability; for $\mathsf{type} \in \{\mathsf{SVPV}, \mathsf{PVSV}, \mathsf{SVSV}\}$, it corresponds to sampled-input (SI) verifiability.

### 1.6.2 Input-Privacy

Tsang et al. [85] formalized the notion of input privacy (referred to as *input secrecy*) for type-SVPC batch delegation, while Canard et al. [29] formalized the type-SVSV case for a single pairing delegation scheme. Both approaches used a chosen-input indistinguishability argument, where an adversary attempts to distinguish between two chosen input pairs. However, another notion of input privacy is also introduced as an alternative, with the aim of capturing a more real-world adversarial scenario.

Two fundamentally different notions of input privacy are formalized below, following standard cryptographic terminology:

- *IND-privacy* (indistinguishability), where the adversary attempts to distinguish between two chosen input pairs.
- *OW-privacy* (one-wayness), where the adversary attempts to recover an input sampled from a chosen $\lambda$-min-entropy distribution. This corresponds to the same adversarial setting as SI-verifiability from the previous subsection.

These notions capture orthogonal security guarantees and may be independently satisfied by a protocol.

**IND-Privacy:** Following the formalization of Canard et al. [29] for type-SVSV delegation, the adversary selects two distinct input pairs $(A_0, B_0)$ and $(A_1, B_1)$ of its choice. The adversary is then given access to a delegation oracle that, upon uniformly sampling a bit $b$, returns a transcript of the delegation of $(A_b, B_b)$. The adversary's goal is to *distinguish* which inputs were used: it wins if it can correctly identify the bit $b$ with probability significantly better than random guessing.

A fundamental requirement for achieving IND-privacy is that the protocol must also hide the pairing output value, as formalized in the following lemma.

**Lemma 1.6.4.** *A protocol* $\mathbf{\Pi}$ *that is IND-input private must also be output-private.*

*Proof.* Suppose $\mathbf{\Pi}$ is not output-private. Then in the IND-input-privacy experiment $\mathbf{Exp}^{\mathsf{IND-priv}}_{\mathbf{\Pi},\mathcal{A}}$ (line 3), the adversary $\mathcal{A}$ can choose pairs $(A_0, B_0)$ and $(A_1, B_1)$ such that $e(A_0, B_0) \neq e(A_1, B_1)$. Upon learning $\rho = e(A_b, B_b)$, the adversary outputs 0 if $\rho = e(A_0, B_0)$ and 1 otherwise, thereby winning the experiment with probability 1. □

This lemma shows that masking the inputs while leaving the pairing value exposed is insufficient for IND-privacy. For example, delegating the pairing evaluation on

$$(C, D) \leftarrow ([r^{-1}]A, [r]B) \text{ for } r \xleftarrow{\$} \mathbb{Z}_q^*$$

information-theoretically hides the inputs, but since $e(C, D) = e(A, B)$, an adversary can trivially win the IND-privacy experiment.

**OW-Privacy:** The OW-privacy experiment is parameterized by the protocol's privacy type $\mathsf{type} \in \{\mathsf{SVPV}, \mathsf{PVSV}, \mathsf{SVSV}\}$, which determines which inputs the adversary must recover. As with SI-verifiability, for each secret input, the adversary cannot choose its value directly but instead selects a $\lambda$-min-entropy distribution from which it is sampled. The adversary wins based on the privacy type:

- $\mathsf{type} = \mathsf{SVPV}$ (A secret, B public): The adversary must recover $A \in \mathbb{G}_1$.
- $\mathsf{type} = \mathsf{PVSV}$ (A public, B secret): The adversary must recover $B \in \mathbb{G}_2$.

- type = SVSV (both secret): The adversary must recover $A$ or $B$.

**Remark 1.6.5.** The OW-privacy notion mirrors the SI-verifiability one in the sense that it models scenarios where the adversary has distributional knowledge about secret inputs rather than control over specific input values. OW-privacy ensures that such distributional knowledge alone is insufficient to recover the actual inputs from the protocol's execution with non-negligible probability.

Definition 1.6.6 formalizes the notions of input privacy for pairing delegation protocols.

**Definition 1.6.6** (IND and OW Input-Privacy for pairing delegation)**.** For a given computational security parameter $\lambda$, and a tuple of auxiliary security parameters $\mathsf{aux.sec}$, a delegation protocol $\mathbf{\Pi}$ of type $\mathsf{type} \in \{\mathsf{SVPV}, \mathsf{PVSV}, \mathsf{SVSV}\}$ is said to be *IND-private* (resp. *OW-private*) against a given class of adversaries if, for any $\mathcal{A}$ in the class, it holds that:

$$\mathcal{P}\Big[\mathbf{Exp}^{\mathsf{IND-priv}}_{\mathbf{\Pi},\mathcal{A}}(\lambda, \mathsf{aux.sec}) = 1\Big] \leq \frac{1}{2} + \mathsf{negl}(\lambda) + \sum_{\delta \in \mathsf{aux.sec}} \mathsf{negl}(\delta)$$

$$\left(\text{resp. } \mathcal{P}\Big[\mathbf{Exp}^{\mathsf{OW-priv}}_{\mathbf{\Pi},\mathcal{A}}(\lambda, \mathsf{aux.sec}, \mathsf{type}) = 1\Big] \leq \mathsf{negl}(\lambda) + \sum_{\delta \in \mathsf{aux.sec}} \mathsf{negl}(\delta)\right).$$

where $\mathbf{Exp}^{\mathsf{IND-priv}}_{\mathbf{\Pi},\mathcal{A}}$ and $\mathbf{Exp}^{\mathsf{OW-priv}}_{\mathbf{\Pi},\mathcal{A}}(\lambda, \mathsf{aux.sec}, \mathsf{type})$ denote the IND and OW input-privacy experiments shown in Figure 1.4.

**Interdependence of security properties:** As pointed out by Canard et al. in [29, §3.2], when building a private input delegation scheme, the probability of success of an adversary against the verifiability property may depend on the one against the privacy. Indeed, they show that Kang et al. scheme [63, §3] allows a server that has learned the private inputs $A, B$ to also produce an incorrect output that will be accepted by the client. This overlap is uncommon in cryptography, where property definitions are usually independent from one another. Designing a protocol that satisfies these two properties independently and also achieves efficiency remains a challenging task.

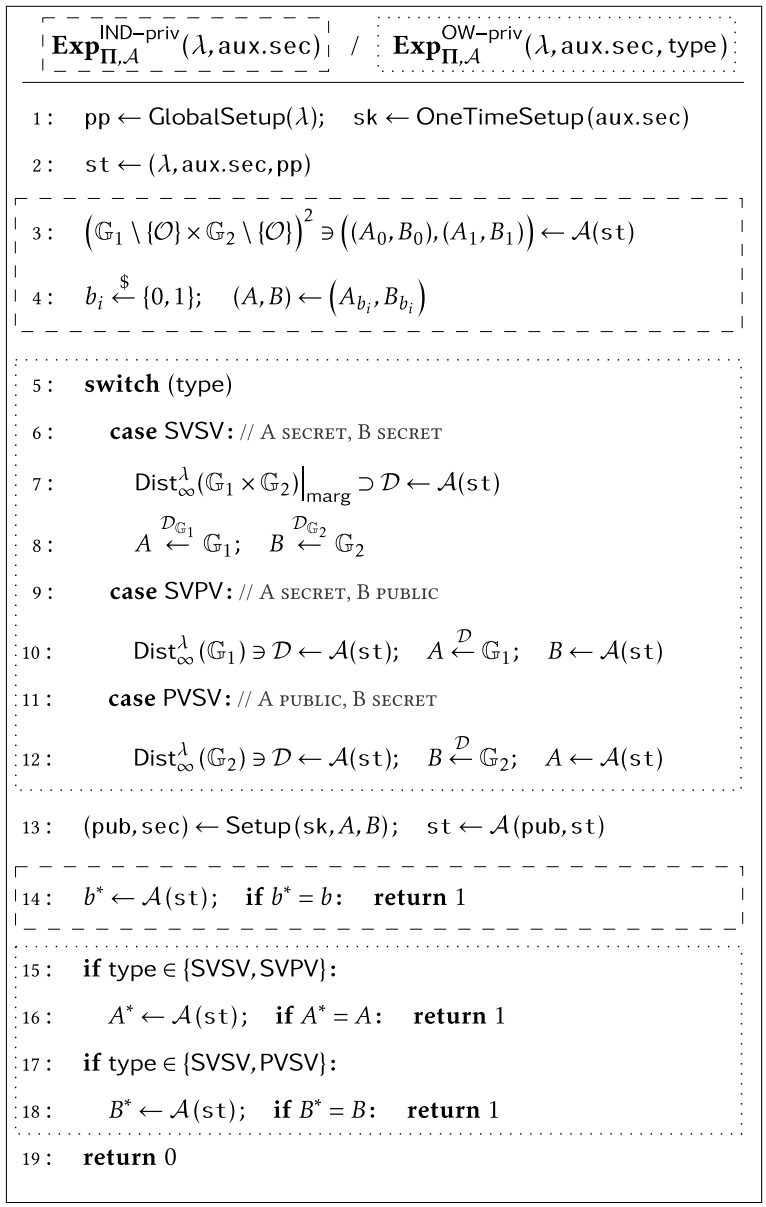


**Figure 1.4:** Input-privacy experiments for one-shot pairing delegation. The IND variant is obtained by excluding the dot-boxed code, while the OW variant is obtained by excluding the dash-boxed code.

# 2

# Delegating Bilinear Pairings: A Systematization

This section presents a comprehensive survey of existing pairing delegation protocols in the two-party setting (client and single server). The protocols are presented in historical order to illustrate the progressive improvements in efficiency and security. Notably, two recently published protocols that allegedly achieve both strong security and high efficiency are identified here as malleable and therefore insecure. The goal of this survey is to serve as a valuable resource for researchers and practitioners interested in pairing delegation.

## 2.1 Survey of Pairing Delegation Protocols

Having established the formal framework and security models, existing pairing delegation protocols are surveyed below. The field has evolved significantly since the earliest works in 2005 [52, 63, 36, 37], with various authors introducing constructions claiming improved efficiency and enhanced security guarantees [63, 88, 87, 29, 56, 75]. However, due to continuously evolving computational costs of pairings and their ancillary operations, some protocols that were once considered practical are now outdated, occasionally requiring the client to perform more computation than local pairing evaluation.

This survey is organized by delegation type: single pairing delegation, batch delegation, and alternative models. Table 2.2 summarizes the state of the art for all described protocols classified between their taxonomy type (PVPV, SVSV,etc) and delegation type (single or batch) in the single untrusted server model.

### 2.1.1 Single Pairing Delegation

Single pairing delegation is relevant for signature verifications and other applications discussed in section 1.2. However, no existing work achieves concrete efficiency for the client when considering the full computational cost.

**Early Approaches:**

- Girault-Lefranc [52] introduced the first secure pairing delegation protocol through the *Server-Aided Verification* notion which consists in speeding up the verification step of an authentication or signature

scheme. In particular, the protocol serves for delegating the pairing-based verification check

$$e(\sigma, f(\mathsf{pp}, m, r)) \stackrel{?}{=} e(P, Q)$$

where $f$ is a public function specific to the scheme, $\mathsf{pp}$ the public parameters including the public key, $r$ stands as an (optional) random coin and $\sigma$ as the signature of a message $m$. An application to the ZSNS [91] signature scheme is given as an example.

- Chevallier-Coron-McCullagh-Naccache-Scott [37] (first published in 2005) propose a type-SVSV protocol and applies variants of it to obtain protocol of types SVPC, PVSC and PVPV. This last variant requires 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.
- Kang-Lee-Park [63] improved the type-SVSV construction of Chevallier et al. requiring the client to perform seven full-range $\mathbb{G}_T$ exponentiations instead of ten, and proposed a variant to type-SVSC (the only one in the literature), PVSC and SVPC. However, Canard et al. [29] later showed that their type-SVSV construction had interdependent security properties: if a server learned the pairing inputs with non-negligible probability $\alpha$, then it could forge a pairing with the same probability, i.e., the scheme is not CI-verifiable.

**Canard-Devigne-Sanders (CDS14):** To improve client efficiency, Canard, Devigne, and Sanders [29] divided the client's workload into an *offline* and an *online phase*. The offline phase performs computations independent of the pairing arguments, while the online phase processes the actual pairing inputs. The authors report efficiency gains solely for the online phase when delegating on the KSS-18 [60] curve but shortly after, Guillevic and Vergnaud [56] showed that this didn't hold for the more widely used Barreto-Naehrig (BN) curves. Implementation results for their type-PVPV protocol in [66, Table 1] demonstrate that the overall cost remains slightly inefficient for the client on all BLS curves.

**Guillevic-Vergnaud (GV14):** Guillevic and Vergnaud [56] proposed two efficient protocols for secret pairing delegation targeting specific use cases

such as Pay-TV smartcards delegating to set-top boxes and GSM sim-cards delegating to smartphone processors. Notably, their protocols are *not verifiable*, which the authors argue is acceptable for encryption primitives where verifiability can be achieved through other means. Their first approach uses a generalized knapsack-based method with endomorphisms to mask the secret point, while their second approach delegates only non-critical steps of Miller's algorithm. The latter is reported to achieve a 65% improvement when applied to the Ate pairing on a Barreto-Naehrig curve for the restricted device compared to local pairing computation.

**Crescenzo et al. (CKKS20):** Crescenzo et al. [42] follow the online/offline setting of Canard et al. and focus mainly on online efficiency by leveraging for the first time *short* $\mathbb{G}_T$ *exponentiations*. They propose a type-PVPV construction and four type-SVSV constructions. However, their offline phases already require computing a pairing. The authors provide efficiency estimates that are extrapolated from a hypothetical text-book implementation using the well-known, but by now outdated, performance figures from [27].

**Aranha-Pagnin-Rodríguez (APR21)** Aranha et al. [6] introduced LOVE (Lowering the cost of Outsourcing and Verifying Efficiently), which optimizes the Compute and Verify phases of Crescenzo et al.'s type-PVPV construction [42, §3]. Leveraging state-of-the-art implementations of pairings and auxiliary operations, the authors reported computational savings in the online phase compared to local pairing evaluation across five curves: BN-254 (15.7%), BN-382 (33.6%), BLS12-381 (45.7%), BLS12-383 (56.1%), and BLS24-509 (56.2%).

### 2.1.2 Batch Pairing Delegation

**Tsang-Chow-Smith (TCS07):** In 2007, besides providing a useful pairing delegation taxonomy, Tsang, Chow, and Smith [85] proposed batch pairing delegation protocols for types PVPC, PVSC and SVPC, i.e., every pair of inputs contains a constant value. Additionally, their main protocol requires the client to perform a pairing computation during the offline phase.

**Mefenza-Vergnaud (MV19):** Later Mefenza and Vergnaud [73] introduced more efficient batch pairing delegation protocols by leveraging the endo-

morphism trick outlined in Guillevic and Vergnaud [56], along with reduced exponent sizes. Four new constructions were presented: one for the type PVPC (Algorithm 2), two for the type PVPV (Algorithms 3 and 4) and one for PVSC (Algorithm 5). However, in all of the proposals the client needs to perform a pairing computation during either the *offline* or the *online* phase for verifying the outsourced pairings. The efficiency estimations were symbolically obtained based on RELIC [4] benchmarks on a BN Curve at the 128-bit security level.

**Crescenzo-Khodjaeva-Caro (CKC23):** Crescenzo et al. [41] proposed batch pairing delegation protocols for hybrid input scenarios where inputs may be available online or offline (see Table 2.1 for details). An implementation of their PVPV protocol shows that its total client cost is consistently higher than in MV19 for all four BLS curves for batches of size $\mathsf{M} < 10$ (see [66, Tables 4 and 5]). Moreover, the authors claim "unlimited client lifetime" without a formal model to support this concept, and "after the offline phase, the number of delegation protocols executable by the resource-constrained client is an arbitrary polynomial" whereas the security proof considers only one-shot executions.

**Kalkar-Sertkaya-Tutdere (KST23):** Kalkar et al. [62] proposed protocols for types PVPV (Algorithm 1), PVSC (Algorithm 2) and SVPC (Algorithm 3) which were claimed verifiable through the Schwartz-Zippel lemma but were later proven broken in [66] due to a simple malleability attack. However, they also introduced the first Type-SVSV batch pairing delegation protocol for which the authors reported client costs of $1.25\rho$ on BN-256, $2.13\rho$ on BLS-512, and $0.97\rho$ on KSS-384 using the Miracl library[1].

### 2.1.3 Alternative Models and Approaches

**Delegation with both online and offline inputs [44, 41]:** An alternative approach to pairing delegation considers scenarios where one of the pairing inputs is known to the client in advance (offline input), while the other input is provided later (online input). This distinction proves useful for constructing case-specific protocols that boost the efficiency of the online phase only, rather than the overall computation. Crescenzo et al. [44]

[1] https://github.com/miracl/MIRACL/tree/master

proposed protocols for five different input scenarios in this hybrid setting for single pairings. Later, the same authors [41] extended this approach to batch pairing delegation, introducing protocols for four input scenarios: (1) $\vec{A}$ secret online, $\vec{B}$ public online; (2) $\vec{A}$ and $\vec{B}$ public online; (3) $\vec{A}$ private online, $\vec{B}$ public offline; and (4) $\vec{A}$ secret online, $\vec{B} = \vec{h} \cdot \vec{H}$ secret online, where the second input follows a special structured form. Table 2.1 summarizes both the single and batch input scenarios along with their corresponding pairing-based cryptographic applications.

| Type | Input Scenario | Applications |
|---|---|---|
| **Single** | A public online, B public offline | [2, 21, 24, 58, 71] |
| | A private online, B public offline | [2, 58] |
| | A private online, B private offline | [2, 21, 19, 71] |
| | A public online, B public online | [59, 71] |
| | A private online, B private online | (No documented applications) |
| **Batch** | $\vec{A}$ secret online, $\vec{B}$ public online | [18, 53] |
| | $\vec{A}$ and $\vec{B}$ public online | [59, 24] |
| | $\vec{A}$ private online, $\vec{B}$ public offline | [2, 58] |
| | $\vec{A}$ secret online, $\vec{B} = \vec{h} \cdot \vec{H}$ secret online | [19] |

**Table 2.1:** Input scenarios for hybrid pairing delegation and their applications. Single pairing cases adapted from [44, Table 1]. Batch cases from [41]. Applications for single pairings: Certificateless public key cryptography [2]; Identity-based encryption [21]; Short signatures (BLS) [24]; Identity-based signature [58]; Self-generated-certificate public key cryptography [71]; Public key encryption with keyword search [19]; Tripartite Diffie–Hellman [59]. Batch applications: Hierarchical identity-based encryption [18]; Attribute-based encryption [53].

**Multi-Server Protocols:** An alternative research line considers more servers via the OMTUP (One-Malicious version of a Two-Untrusted-Program) model [35, 69, 61, 83]. However, 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. As a notable contribution, Tong-Yu-Zhang present a pairing delegation construction ([84, Algorithm 2]) that requires no pre-computation (and hence efficient), achieving provable security through interaction with an honest server.

## 2.2 Symbolic Cost Comparison

Table 2.3 lists the symbolic client costs of the protocols for pairing delegation that are compared in this thesis, including the ones from the literature and the ones proposed in this work. The costs are expressed in terms of the bilinear group operations used in the description of each protocol. The table is organized by the types PVPV, PVSV, SVPV and SVSV, and by whether the protocol is designed for single or batch pairing delegation. The protocols proposed in this thesis are highlighted in grey.

## 2.3 Efficient but Malleable Pairing Delegation Protocols

In this section, we present two recently proposed pairing delegation protocols, KC23 [67] and KST23 [62] that do not require any client pre-processing on $\mathbb{G}_T$ elements, thereby skipping the offline phase. Both protocols aim to provide computational savings for the client while ensuring efficiency compared to locally evaluating the pairing. However, both protocols are vulnerable to malleability attacks, i.e., the server can efficiently produce an output $\mathtt{out}^* \neq \mathtt{out}$ that will be accepted by the client with probability 1. Furthermore, a computationally unbounded server can choose any tuple $\vec{\rho}$ to include

| Type | Single | Batch |
|---|---|---|
| PVPV | CMCNS10 [37, §5.2],<br>CDS14 [29, §4],<br>CKKS20 [42, §3],<br>APR21 [6, §4],<br>KAPR25 [66, §5, $M = 1$] | MV19 [73, §4.2, Alg. 3],<br>MV19 [73, §4.2, Alg. 4],<br>CKC23 [41, §3.2],<br>KAPR25 [66, §5, $M > 1$] |
| SVPV | CMCNS10 [37, §5.1],<br>KLP05 [63, §4.1],<br>K26-s-SVPV (Fig. 4.2) | CKC23 [41, §3.1],<br>K26-svpv-b (Fig. 4.2) |
| PVSV | K26-s-PVSV (Fig. 4.2) | K26-pvsv-b (Fig. 4.2) |
| SVSV | KLP05 [63, §3],<br>CMCNS10 [37, §4.1],<br>CDS14 [29, §5.1],<br>CKKS20 [42, §4],<br>K26 (Fig. 4.2) | KST23 [62, §3.6],<br>KST23-Opti (Fig. 4.4),<br>K26-svsv-b (Fig. 4.4) |
| **SVSC** | KLP05 [63, §4.2] | TCS07 [85, §4.1, SVSC],<br>MV19 [73, §4.3, Alg. 5] |
| **SVPC** | KLP05 [63, §4.3],<br>CDS14 [29, §5.2] | TCS07 [85, §4.1, SVPC] |
| **PVSC** | CMCNS10 [37, §6.2] | TCS07 [85, §4.1, PVSC, $n > 1$] |
| **PVPC** | TCS07 [85, §4.1, PVPC],<br>CMCNS10 [37, §6.1] | MV19 [73, §4.1, Alg. 2] |

**Table 2.2:** Classification of pairing delegation protocols. Boxed protocols are contributions of this thesis and presented in chapters 3 and 4.

| Type | Protocol | Symbolic client cost |
|---|---|---|
| PVPV (Single) | CDS14[29, §4] | $\mathsf{N}(2\mathfrak{m}_1 + 2\mathfrak{m}_2 + 2\mathfrak{m}_T + \in_T + \mathfrak{g}_1 + \mathfrak{g}_2 + \mathfrak{g}_T)$ |
| | CKKS20[42, §3] | $\mathsf{N}\left(\mathfrak{p} + \mathfrak{m}_1^{\$} + \mathfrak{m}_2^{\$} + \mathfrak{m}_1 + \mathfrak{m}_2 + \mathfrak{m}_2^{\mathsf{sh}} + \mathfrak{m}_T^{\mathsf{sh}} + 2 \in_T + \mathfrak{g}_1 + \mathfrak{g}_2 + 2\mathfrak{g}_T\right)$ |
| | APR21[6, §4] | $\mathsf{N}\left(\mathfrak{p} + \mathfrak{m}_1^{\$} + \mathfrak{m}_2^{\$} + \mathfrak{m}_1 + \mathfrak{m}_2 + \mathfrak{m}_2^{\mathsf{sh}} + \mathfrak{m}_T^{\mathsf{sh}} + \in_T + \mathfrak{g}_1 + \mathfrak{g}_2 + \mathfrak{g}_T\right)$ |
| | KAPR25[66, §5, M = 1] | $\mathfrak{m}_T + \mathsf{N}\left(2\mathfrak{m}_1 + \mathfrak{m}_2 + \mathfrak{m}_2^{\mathsf{sh}} + \mathfrak{m}_T^{\mathsf{sh}} + \in_T + \mathfrak{g}_1 + \mathfrak{g}_2 + \mathfrak{g}_T\right)$ |
| PVPV (Batch) | MV19[73, §4.2, Alg. 4] | $\mathsf{N}\left(\mathfrak{p} + \mathfrak{m}_2^{\$} + \mathsf{M}\left(\mathfrak{m}_1 + \mathfrak{m}_2 + \mathfrak{m}_2^{\mathsf{sh}} + \mathfrak{m}_T^{\mathsf{sh}} + 2 \in_T + \mathfrak{g}_1 + \mathfrak{g}_2 + 2\mathfrak{g}_T\right)\right)$ |
| | CKC23[41, §3.2] | $\mathsf{N}\left(\mathfrak{p} + \mathfrak{m}_1^{\$} + \mathfrak{m}_2^{\$} + \mathsf{M}\left(\mathfrak{m}_1 + \mathfrak{m}_1^{\mathsf{sh}} + \mathfrak{m}_2 + \mathfrak{m}_T^{\mathsf{sh}} + \in_T + \mathfrak{g}_1 + \mathfrak{g}_T\right)\right)$ |
| | KAPR25[66, §5, M > 1] | $\mathfrak{m}_T + \mathsf{N}\left(\mathfrak{m}_1 + \mathfrak{m}_1^{\$} + 2\mathfrak{m}_2 + \mathsf{M}\left(\mathfrak{m}_1^{\mathsf{sh}} + \mathfrak{m}_T^{\mathsf{sh}} + \in_T + \mathfrak{g}_1 + \mathfrak{g}_2 + \mathfrak{g}_T\right)\right)$ |
| SVSV (Single) | CDS14[29, §5.1] | $\mathsf{N}(3\mathfrak{m}_1 + 3\mathfrak{m}_2 + 3\mathfrak{m}_T + \in_T + \mathfrak{g}_1 + \mathfrak{g}_2 + \mathfrak{g}_T)$ |
| | CKKS20[42, §4] | $\mathsf{N}\left(\mathfrak{p} + 2\mathfrak{m}_1^{\$} + 2\mathfrak{m}_2^{\$} + 3\mathfrak{m}_1 + 5\mathfrak{m}_2 + \mathfrak{m}_2^{\mathsf{sh}} + \mathfrak{m}_T^{\mathsf{sh}} + 3 \in_T + 2\mathfrak{g}_1 + 2\mathfrak{g}_2 + 4\mathfrak{g}_T\right)$ |
| | APR21[6, §5.2] | $\mathsf{N}\left(\mathfrak{p} + 2\mathfrak{m}_1^{\$} + 2\mathfrak{m}_2^{\$} + 3\mathfrak{m}_1 + 5\mathfrak{m}_2 + \mathfrak{m}_2^{\mathsf{sh}} + \mathfrak{m}_T^{\mathsf{sh}} + \in_T + 2\mathfrak{g}_1 + 2\mathfrak{g}_2 + 3\mathfrak{g}_T\right)$ |
| | K26-svsv-s (Fig. 4.2) | $\mathfrak{m}_T + \mathsf{N}\left(\frac{5}{2}\mathfrak{m}_1 + \mathfrak{m}_2 + \frac{1}{2}\mathfrak{m}_T + \in_T + \mathfrak{g}_1 + \mathfrak{g}_2 + \mathfrak{g}_T\right)$ |
| SVSV (Batch) | KST23[62, §3.6] | $\mathsf{N}\left(\mathsf{M}\left(\mathfrak{m}_T + \mathfrak{m}_2^{\mathsf{sh}} + \mathfrak{m}_T^{\mathsf{sh}} + \mathfrak{g}_2 + 2\mathfrak{g}_T\right) + (1 + 2\mathsf{M}) \cdot (\mathfrak{m}_1 + \mathfrak{m}_2 + \in_T)\right)$ |
| | KST23-Opti (Fig. 4.4) | $\mathsf{N}\left(\mathsf{M}\left(\mathfrak{m}_T + \mathfrak{m}_2^{\mathsf{sh}} + \mathfrak{m}_T^{\mathsf{sh}} + \in_T + \mathfrak{g}_2 + \mathfrak{g}_T\right) + (1 + 2\mathsf{M}) \cdot (\mathfrak{m}_1 + \mathfrak{m}_2)\right)$ |
| | K26-svsv-b (Fig. 4.3) | $\mathfrak{m}_T + \mathsf{N}\left(\mathfrak{m}_1 + \mathfrak{m}_1^{\$} + \frac{3}{2}\mathfrak{m}_2 + \mathsf{M}\left(2\mathfrak{m}_1 + \frac{1}{2}\mathfrak{m}_2 + \frac{1}{2}\mathfrak{m}_T + \in_T + \mathfrak{g}_1 + \mathfrak{g}_2 + \mathfrak{g}_T\right)\right)$ |
| SVPV (Single) | CMCNS10[37, §5.1] | $\mathsf{N}(3\mathfrak{m}_1 + 2\mathfrak{m}_2 + 5\mathfrak{m}_T + 3 \in_T + \mathfrak{g}_1 + \mathfrak{g}_2 + 4\mathfrak{g}_T)$ |
| | KLP05[63, §4.1] | $\mathsf{N}(2\mathfrak{m}_1 + 2\mathfrak{m}_2 + 5\mathfrak{m}_T + 3 \in_T + \mathfrak{g}_1 + \mathfrak{g}_2 + 3\mathfrak{g}_T)$ |
| | K26-svpv-s (Fig. 4.2) | $\mathfrak{m}_T + \mathsf{N}\left(2\mathfrak{m}_1 + \mathfrak{m}_2 + \frac{1}{2}\mathfrak{m}_T + \in_T + \mathfrak{g}_1 + \mathfrak{g}_2 + \mathfrak{g}_T\right)$ |
| SVPV (Batch) | CKC23[41, §3.1] | $\mathsf{N}\left(\mathfrak{p} + \mathfrak{m}_1^{\$} + \mathfrak{m}_2^{\$} + \mathsf{M}\left(2\mathfrak{m}_1 + \mathfrak{m}_1^{\mathsf{sh}} + \mathfrak{m}_2 + \mathfrak{m}_T + \mathfrak{m}_T^{\mathsf{sh}} + \in_T + \mathfrak{g}_1 + \mathfrak{g}_T\right)\right)$ |
| | K26-svpv-b (Fig. 4.3) | $\mathfrak{m}_T + \mathsf{N}\left(\mathfrak{m}_1 + \mathfrak{m}_1^{\$} + \frac{3}{2}\mathfrak{m}_2 + \mathsf{M}\left(\mathfrak{m}_1 + \frac{1}{2}\mathfrak{m}_T + \in_T + \mathfrak{g}_1 + \mathfrak{g}_2 + \mathfrak{g}_T\right)\right)$ |
| PVSV | K26-pvsv-s (Fig. 4.2) | $\mathfrak{m}_T + \mathsf{N}\left(\frac{3}{2}\mathfrak{m}_1 + \frac{3}{2}\mathfrak{m}_2 + \frac{1}{2}\mathfrak{m}_T + \in_T + \mathfrak{g}_1 + \mathfrak{g}_2 + \mathfrak{g}_T\right)$ |
| | K26-pvsv-b (Fig. 4.3) | $\mathfrak{m}_T + \mathsf{N}\left(\mathfrak{m}_1 + \mathfrak{m}_1^{\$} + \frac{3}{2}\mathfrak{m}_2 + \mathsf{M}\left(\mathfrak{m}_1 + \frac{1}{2}\mathfrak{m}_2 + \frac{1}{2}\mathfrak{m}_T + \in_T + \mathfrak{g}_1 + \mathfrak{g}_2 + \mathfrak{g}_T\right)\right)$ |

**Table 2.3:** Symbolic client cost for types PVPV, SVSV, SVPV and PVSV pairing delegation.

in $\mathtt{out}^*$ that will be wrongly accepted by the client as the correct pairing values $e(A_i, B_i)$. This insight is particularly worrying since

- Both protocols jointly cover the single (KC23) and batch (KST23) delegation setting and achieve high efficiency as a result of omitting expensive $\mathbb{G}_T$ pre-processing operations by the client. By claiming unconditional security as well, they present themselves as attractive solutions for practical deployment in real use cases.

- The property that a malicious server can bypass the client's verification phase with any arbitrarily chosen output (full malleability), can cause serious security issues apart from correctness in higher-level pairing-based protocols, as briefly shown in subsection 2.3.3.

### 2.3.1 Khodjaeva-Crescenzo Protocol (2023)

| $\mathsf{Setup}(A, B) \to (\mathsf{pub}, \mathtt{sec})$ | $\mathsf{Compute}(\mathsf{pub}) \to \mathtt{out}$ | $\mathsf{Verify}(\mathtt{sec}, \mathtt{out}) \to \mathtt{result}$ |
|---|---|---|
| 1: $r \xleftarrow{\$} [\![2^\sigma]\!]$ | 1: **parse** $\mathsf{pub} = (A, B, W, Y)$ | 1: **parse** $\mathtt{sec} = r$ |
| 2: $u \xleftarrow{\$} \mathbb{Z}_q^*, U \xleftarrow{\$} \mathbb{G}_1$ | 2: $\rho \leftarrow e(A, B)$ | 2: **if** $(\rho \notin \mathbb{G}_T$ OR $\gamma_1 \notin \mathbb{G}_T)$ |
| 3: $W \leftarrow [u^{-1}]U$ | 3: $\gamma_1 \leftarrow e(W, B)$ | 3: **return** $\perp$ |
| 4: $Y \leftarrow [r]A + U$ | 4: $\gamma_2 \leftarrow e(Y, B)$ | 4: **if** $(\gamma_2 \neq \rho^r \cdot \gamma_1^u)$ |
| 5: $\mathsf{pub} \leftarrow (A, B, W, Y)$ | 5: $\mathtt{out} \leftarrow (\rho, \gamma_1, \gamma_2)$ | 5: **return** $\perp$ |
| 6: $\mathtt{sec} \leftarrow r$ | | 6: $\mathtt{result} \leftarrow \rho$ |

**Figure 2.1:** Khodjaeva and Crescenzo's Protocol (KC23) in [67, §4].

In KC23 (see figure Figure 2.1), the server outputs three elements $(\rho, \gamma_1, \gamma_2)$, and the client verifies the delegation by checking that $\rho$ and $\gamma_1$ 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).

**The malleability 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 $\rho^x$ and $\gamma_1^x$ belong to $\mathbb{G}_T$ and

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

Furthermore, if the server can solve the discrete logarithm problem in $\mathbb{G}_T$, then for any desired target value $\rho^* \in \mathbb{G}_T$, the server can compute $x = \log_\rho(\rho^*)$ (where $\rho = e(A,B)$ is the honest pairing output), and return $\mathtt{out}^* = (\rho^*, \gamma_1^x, \gamma_2^x)$. This forged output satisfies the verification equation and will be accepted by the client as the correct pairing value $e(A,B) = \rho^*$, even though $\rho^*$ was arbitrarily chosen by the server. Thus, a computationally unbounded adversary has full control over the pairing value that the client accepts.

### 2.3.2 Kalkar, Sertkaya, and Tutdere Batch Pairing Delegation Protocol (2023)

$\mathsf{Setup}(\vec{A}, \vec{B}) \to (\mathsf{pub}, \mathsf{sec})$

1: **for** $i \in [\![ \mathtt{len}(\vec{A}) ]\!]$ :
2: $\quad r_i \xleftarrow{\$} [\![ 2^{2\lambda-1} ]\!]$
3: $\quad C \leftarrow [r_i]A_i$
4: $\mathsf{pub} \leftarrow (\vec{A}, \vec{B}, \vec{C})$
5: $\mathsf{sec} \leftarrow (\vec{r})$

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

1: **parse** $\mathsf{pub} = (\vec{A}, \vec{B}, \vec{C})$
2: **for** $i \in [\![ \mathtt{len}(\vec{A}) ]\!]$ :
3: $\quad \rho_i \leftarrow e(A_i, B_i)$
4: $\quad \gamma_i \leftarrow e(C_i, B_i)$
5: $\mathtt{out} \leftarrow (\vec{\rho}, \vec{\gamma})$

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

1: **for** $i \in [\![ \mathtt{len}(\vec{A}) ]\!]$ :
2: $\quad$ **if** $(\rho_i, \gamma_i \notin \mathbb{G}_T)$:
3: $\quad\quad$ **return** $\perp$
4: $\quad$ **if** $(\gamma_i \neq \rho_i^{r_i})$:
5: $\quad\quad$ **return** $\perp$
6: $\mathtt{result} \leftarrow \vec{\rho}$

**Figure 2.2:** Kalkar, Sertkaya, and Tutdere batch pairing delegation protocol (KST23) with public variable inputs [62, Algorithm 1].

In KST23 (see Figure 2.2), the server outputs two vectors $(\vec{\rho}, \vec{\gamma})$, and the client verifies the delegation by checking that $(\vec{\rho} \in \mathbb{G}_T^n) \wedge (\vec{\gamma} \in \mathbb{G}_T^n)$ and that $\gamma_i = \rho_i^r$, where each $r_i$ is a client random secret coin.

**The malleability attack.** Analogous to the one against KC23. Simply consider multiple (malleable) verification equations. Instead of the honest tuple $(\vec{\gamma}, \vec{\rho})$, the malicious server returns

$$\mathtt{out}^* = (\vec{\gamma}^*, \vec{\rho}^*) = (\vec{\gamma}^x, \vec{\rho}^x),$$

where $x \in \mathbb{Z}_q \setminus \{0,1\}$. As before, $\vec{\gamma}, \vec{\rho} \in \mathbb{G}_T^n$ implies $\vec{\gamma}^*, \vec{\rho}^* \in \mathbb{G}_T^n$ since $\mathbb{G}_T$ is closed under multiplication. Moreover, for every $i \in [\![n]\!]$,

$$\gamma_i = \rho_i^{r_i} \Leftrightarrow \gamma_i^* = (\rho_i^*)^{r_i} \text{ since } \gamma_i^* = \gamma_i^x = (\rho_i^{r_i})^x = (\rho_i^x)^{r_i} = (\rho_i^*)^{r_i}.$$

Hence, $\mathtt{out}^*$ is not rejected by the verification and the client accepts the pairing values $\vec{\rho}^* \neq \vec{\rho}$. Similarly to the previous malleability attack against KC23 [67], if the server can solve the discrete logarithm problem in $\mathbb{G}_T$, then for any desired target vector $\vec{\rho}^* \in \mathbb{G}_T^n$, the server can compute $x_i = \log_{\rho_i}(\rho_i^*)$ for each $i \in [\![n]\!]$ (where $\rho_i = e(A_i, B_i)$ are the honest pairing outputs), and return $\mathtt{out}^* = (\vec{\gamma}^*, \vec{\rho}^*)$ with $\gamma_i^* = \gamma_i^{x_i}$. This forged output satisfies all verification equations and will be accepted by the client as the correct pairing values, even though $\vec{\rho}^*$ was arbitrarily chosen by the server. Thus, a computationally unbounded adversary has full control over the pairing values that the client accepts.

**Remark 2.3.1.** The constructions for types PVSC and SVPC in KST23 [62, Algorithms 2 and 3] are built upon their flawed type-PVPV construction [62, Algorithms 1] (Figure 2.2) and thus inherit the same malleability vulnerability.

### 2.3.3 Impact on Higher-Level Pairing-Based Protocols

Canard et al. [29] already highlighted the dangers of using non verifiable pairing delegation protocols for a later use of pairing-based cryptographic schemes, like digital signatures. Since a malleable pairing delegation protocol is by its own definition non verifiable, the same concerns apply here. A few examples of the (possibly devastating) impacts are given below.

**Signature Forgeries** Consider the BLS signature scheme [24], where the verification of a signature $\sigma = [s]H(m) \in \mathbb{G}_1$ on a message $m \in \{0,1\}^*$ under a public key $pk = [s]Q \in \mathbb{G}_2$ involves checking the equality

$$e(\sigma, Q) \stackrel{?}{=} e(H(m), pk),$$

with $H : \{0,1\}^* \to \mathbb{G}_1$ being a hash function modeled as a random oracle. If the pairing computations $e(\sigma, Q)$ and $e(H(m), pk)$ are delegated using a fully malleable pairing delegation protocol, a malicious server could simply return any two equal values in $\mathbb{G}_T$, causing the client to accept any incoming

BLS signature, even invalid ones. Hence, the security of the entire signature scheme is compromised.

**zk-SNARK soundness** Groth16 [54] is one of the most widely used pairing-based Zero Knowledge Succint Non interactive ARgument of Knowledge (zk-SNARK) protocols, especially in on-chain verification like in Zcash and Ethereum. Given the proof $\pi = (A, B, C)$ of a prover-claimed statement, the verifier checks an equality of the form

$$e(A, B) \stackrel{?}{=} e(\alpha, \beta) \cdot e(\gamma, \delta) \cdot e(C, \theta),$$

where the remaining pairing arguments are obtained from the public parameters and a trusted setup. Similarly to the BLS case, a malicious server could cause a false statement from a malicious prover to be accepted by the verifier. This completely breaks the soundness property of the zk-SNARK system since the pairing equation is the only soundness check. The consequences of this could be disastrous in practice, as it would allow forgeries of zero-knowledge proofs for false statements, potentially leading to unauthorized transactions.

**Consensus Failures in Blockchain Networks** For blockchains that rely on Groth16 for validity checks, if some nodes outsource pairing computations using an efficient but (unknowingly) fully malleable pairing delegation protocol, while other nodes compute pairings locally, then the former may accept invalid zk-SNARK proofs that the latter correctly reject. This results in inconsistent block validity rules across the network, which can cause chain bifurcations (forks) and, in the case of widespread delegation, a violation of consensus safety through the acceptance of blocks containing invalid state transitions.

## 2.4 Securely Delegating a Pairing without Precomputation is Impossible

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 challenging task, even when considering the pairing arguments as public inputs. This has led to insecure constructions for both single and batch pairing delegation, as

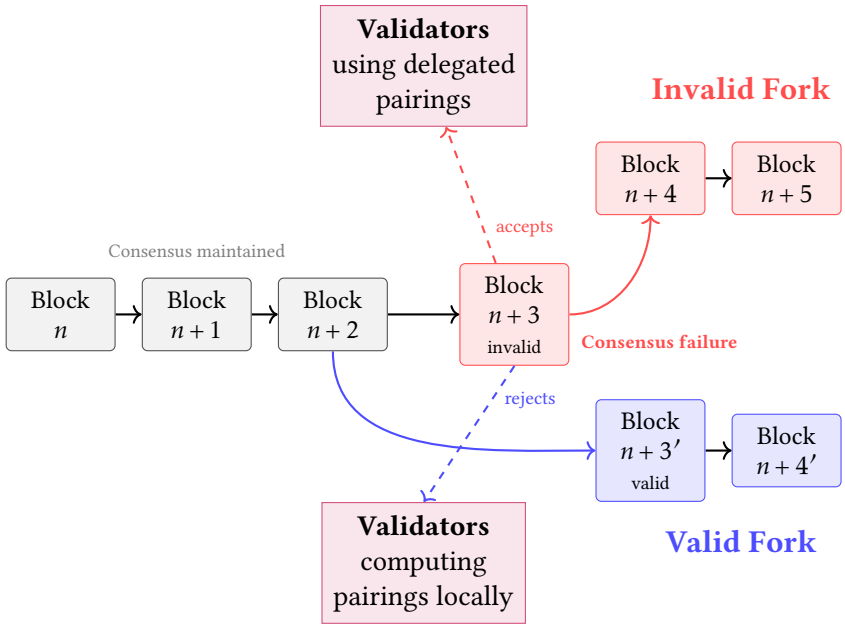


**Figure 2.3:** Impact of fully malleable pairing delegation on blockchain consensus that uses Groth16 as a zk-SNARK. Validators relying on a fully malleable pairing delegation protocol can be tricked into accepting an invalid zk-SNARK proof, leading to the finalization of invalid blocks in the blockchain consensus.

we have seen in Section 2.3 with the malleability attacks against KC23 [67, Section 4] and KST23 [62, Alg. 1]. Theorem 2.4.1 establishes sufficient and necessary conditions for the existence of malleability attacks against single and batch pairing delegation protocols. As a warmup, we first define the notion of group homomorphism that will be used in the theorem statement and proof.

**Group homomorphism:** A homomorphism $\phi$ in a group $(\mathbb{G}, \circ)$ satisfies the following property: $\phi(\gamma_1,\ldots,\gamma_n)\circ\phi(\gamma'_1,\ldots,\gamma'_n) = \phi(\gamma_1\circ\gamma'_1,\ldots,\gamma_n\circ\gamma'_n)$. In particular, for any $x \in \mathbb{Z}_q$, $\phi(\gamma_1^x,\ldots,\gamma_n^x) = \phi(\gamma_1,\ldots,\gamma_n)^x$.

**Theorem 2.4.1.** *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,* $\vec{\gamma} = (\gamma_1,\ldots,\gamma_n)$ *the values returned by the server, and*

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

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

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

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

□

# 3

# Amortized Efficiency for Pairing Delegation with Public Inputs

This chapter focuses on the core contributions of [66] for type-PVPV delegation and on the introduction of new state-of-the-art constructions for type-SVSV delegation. All previous works on pairing delegation are one-shot in the sense that the execution is terminated after a single interaction between the client and the server, which obliges the client to re-compute the offline phase each time a delegation is performed. Additionally, these protocols are all designed for achieving verifiability via information theoretic arguments. This comes at the cost of limiting the overall efficiency of the scheme. Chevallier-Mames et al. [37] already pointed out that *"an interesting research direction would be to further optimize the protocols by trading-off unconditional security against computational security"*.
The proposed construction, KAPR25 [66, §5], unifies single and batch delegation by allowing sequential delegation from a one-time setup, and is proven secure/verifiable in the more realistic setting of computationally bounded adversaries instead of unbounded ones, which ensures *everlasting security* [57] provided the server is unable to produce a pairing forgery within the protocol's execution time. Finally, as a consequence of this "security degradation", significantly higher levels of efficiency are reached with respect to the previous state-of-the-art.

## 3.1 Modeling Sequential Pairing Delegation and Amortized Efficiency

Sequential pairing delegation can be expressed using the syntax introduced in Definition 1.4.1, 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 $\mathsf{N} \geq 1$ of times, using the output of OneTimeSetup. Sequential delegations enable the consideration of 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 $\mathsf{M}_i = 1$, and *batch* whenever $\mathsf{M} > 1$.

In the remainder of this section, definitions of correctness, concrete amortized efficiency, and security (verifiability) are presented for sequential pairing delegation protocols, covering both the single and batch settings.

### 3.1.1 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) $i \in [\![\mathsf{N}]\!]$ 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, $\mathsf{M}_i$, may vary at every delegation round).

**Definition 3.1.1** (Correctness)**.** A protocol $\mathbf{\Pi}$ for sequential pairing delegation is said to be $L$-correct if, for any number of rounds (delegations) $\mathsf{N}$, and batch sizes $\vec{\mathsf{M}} = \{M_i\}_{i=1}^{\mathsf{N}}$ satisfying $\mathbf{1}^T \cdot \vec{\mathsf{M}} \leq L$, for any choice of parameters $\lambda$ (computational security), $\texttt{aux.sec}$ (e.g., efficiency, statistical security,

timeout); for any pp output by GlobalSetup($\lambda$), and for any sk produced by OneTimeSetup(aux.sec), it holds that: for all $i \in [\![\mathsf{N}]\!]$,

$$\Pr\left[\begin{array}{l|c} \mathtt{result} = \{e(A_{ij}, B_{ij})\}_{j \in [\![\mathsf{M}_i]\!]} & \begin{array}{c}(\mathsf{pub}, \mathtt{sec}) \leftarrow \mathsf{Setup}(\mathsf{sk}, \vec{A}_i, \vec{B}_i) \\ \mathtt{out} \leftarrow \mathsf{Compute}(\mathsf{pub}) \\ \mathtt{result} \leftarrow \mathsf{Verify}(\mathsf{sk}, \mathtt{sec}, \mathtt{out})\end{array} \end{array}\right] = 1\,.$$

### 3.1.2 Defining Amortized Efficiency

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

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

**Definition 3.1.2** (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 $(\mathsf{N}, \vec{\mathsf{M}})$ for delegating $L$ pairings and a real value $\epsilon \in ]0, 1[$ such that:

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

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

From Definition 3.1.2 the notion of concrete amortized efficiency can be drawn for a fixed batch size $\mathsf{M} \geq 1$, and a target efficiency ratio $\epsilon \in ]0, 1[$ by finding (if it exists) the smallest integer $\mathsf{N} \geq 1$ such that $\frac{\mathsf{cost(client)}}{\mathsf{M} \cdot \mathsf{N} \cdot \mathfrak{p}} \leq \epsilon$.

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

### 3.1.3 Verifiability models in the Sequential Setting

The verifiability experiment for one-shot delegation presented in section 1.6 can be generalized to the sequential setting in a natural way. As in the one-shot case, the experiment is parameterized by a privacy type $\mathsf{type} \in \{\mathsf{PVPV}, \mathsf{SVPV}, \mathsf{PVSV}, \mathsf{SVSV}\}$, which determines the adversary's control over the pairing inputs. This experiment is denoted $\mathbf{Exp}^{\mathsf{seq,ver}}_{\mathbf{\Pi},\mathcal{A}}$ and is depicted in Figure 3.1.

Intuitively, the adversary $\mathcal{A}$ plays the role of a malicious server that at each delegation round, based on the protocol's privacy type:

- When $\mathsf{type} = \mathsf{PVPV}$ (CI verifiability): The adversary directly controls both pairing inputs at each delegation round.
- When $\mathsf{type} \in \{\mathsf{SVPV}, \mathsf{PVSV}, \mathsf{SVSV}\}$ (SI verifiability): For each input designated as secret, the adversary selects a $\lambda$-min-entropy distribution from which it is sampled, rather than choosing the input directly.

In detail, the experiment sets up an instance of a sequential pairing delegation protocol $\mathbf{\Pi}$ for a fixed number $L$ of pairings; and for every delegation round, $\mathcal{A}$ adaptively selects the pairing arguments or sampling distributions according to the protocol type. The adversary wins the experiment if

1. At a given delegation round $i$, a forgery is produced, i.e., an $\mathsf{out}^*$ for which $\mathsf{Verify}$ returns a tuple in $\mathbb{G}_T$ not matching $\{e(A_{ij}, B_{ij})\}_{j=1}^{\mathsf{M}_i}$.
2. All outputs returned by the adversary prior to the forgery are honest.
3. The number of delegated pairings up that point does not 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 one-shot delegation security experiment $\mathbf{Exp}^{\mathsf{ver}}_{\mathbf{\Pi},\mathcal{A}}$.

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

$\mathsf{pp} \leftarrow \mathsf{GlobalSetup}(\lambda)$; $\quad \mathsf{sk} \leftarrow \mathsf{OneTimeSetup}(\mathsf{aux.sec})$

$\mathtt{st} \leftarrow (\lambda, \mathsf{aux.sec}, \mathsf{pp})$; $\quad \mathsf{ctr} \leftarrow 0$

**while** true: // DELEGATION ROUNDS

  **switch** (type)

    **case** SVSV: // A SECRET, B SECRET

      $\mathsf{Dist}^{\lambda}_{\infty}(\mathbb{G}_1 \times \mathbb{G}_2)\big|_{\mathsf{marg}} \ni \vec{\mathcal{D}} \leftarrow \mathcal{A}(\mathtt{st})$;

      $\vec{A} \xleftarrow{\vec{\mathcal{D}}_{\mathbb{G}_1}} \mathbb{G}_1$; $\quad \vec{B} \xleftarrow{\vec{\mathcal{D}}_{\mathbb{G}_2}} \mathbb{G}_2$

    **case** SVPV: // A SECRET, B PUBLIC

      $\mathsf{Dist}^{\lambda}_{\infty}(\mathbb{G}_1) \ni \vec{\mathcal{D}} \leftarrow \mathcal{A}(\mathtt{st})$; $\quad \vec{A} \xleftarrow{\vec{\mathcal{D}}} \mathbb{G}_1$; $\quad \vec{B} \leftarrow \mathcal{A}(\mathtt{st})$

    **case** PVSV: // A PUBLIC, B SECRET

      $\mathsf{Dist}^{\lambda}_{\infty}(\mathbb{G}_2) \ni \vec{\mathcal{D}} \leftarrow \mathcal{A}(\mathtt{st})$; $\quad \vec{B} \xleftarrow{\vec{\mathcal{D}}} \mathbb{G}_2$; $\quad \vec{A} \leftarrow \mathcal{A}(\mathtt{st})$

    **case** PVPV: // A PUBLIC, B PUBLIC

      $\left(\vec{A}, \vec{B}\right) \leftarrow \mathcal{A}(\mathtt{st})$

  $\mathsf{ctr} \leftarrow \mathsf{ctr} + \mathsf{len}(\vec{A})$; $\quad$ **if** $\mathsf{ctr} \geq L$: $\quad$ **break**

  $(\mathsf{pub}, \mathtt{sec}) \leftarrow \mathsf{Setup}(\mathsf{sk}, \vec{A}, \vec{B})$;

  $(\mathtt{out}^*, \mathtt{st}) \leftarrow \mathcal{A}(\mathsf{pub}, \mathtt{st})$

  $\mathtt{result}^* \leftarrow \mathsf{Verify}(\mathsf{sk}, \mathtt{sec}, \mathtt{out}^*)$

  **if** $\mathtt{result}^* \notin \{\perp, e(\vec{A}, \vec{B})\}$: $\quad$ **return** 1

  **if** $\mathtt{result}^* = \perp$: $\quad$ **return** 0

**return** 0

**Figure 3.1:** Verifiability experiment for sequential pairing delegation.

**Remark 3.1.3.** A more typical and stronger security notion could be defined by allowing the adversary to return non successful forgeries $\mathsf{out}^*$ without automatically losing the game, simply by removing line 19 in Figure 3.1. However, this extra requirement is not realistic in the current framework: if the client rejects the server's output in the verification phase, it can decide to interact with another server for future delegations, and/or refresh its secret randomness by re-running OneTimeSetup.

**Definition 3.1.4** (Sequential verifiability)**.** For a given positive integer $L \geq 1$, a computational security parameter $\lambda$, and a tuple of auxiliary security parameters $\mathsf{aux.sec}$, a sequential delegation protocol $\mathbf{\Pi}$ of type $\mathsf{type} \in \{\mathsf{PVPV}, \mathsf{SVPV}, \mathsf{PVSV}, \mathsf{SVSV}\}$ is said to be *sequentially verifiable* against a given class of adversaries if, for any $\mathcal{A}$ in the class, it holds that:

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

where $\mathbf{Exp}^{\mathsf{seq,ver}}_{\mathbf{\Pi},\mathcal{A}}(\lambda, \mathsf{aux.sec}, L, \mathsf{type})$ denotes the sequential verifiability experiment shown in Figure 3.1. When $\mathsf{type} = \mathsf{PVPV}$, this corresponds to sequential chosen-input (CI) verifiability; for $\mathsf{type} \in \{\mathsf{SVPV}, \mathsf{PVSV}, \mathsf{SVSV}\}$, it corresponds to sequential sampled-input (SI) verifiability.

## 3.2 Amortized Efficiency for Type-PVPV Pairing Delegation

This section presents the core construction of this thesis, KAPR25 [66, §5] (named "AmorE" by the authors), a type-PVPV sequential pairing delegation protocol that is CI-verifiable and achieves concrete amortized efficiency.

KAPR25 is instantiated with the following auxiliary security parameters: (1) a latency parameter $\tau$, which upperbounds the duration of the protocol in seconds, and (2) an efficient parameter $\varphi$, which balances the trade-off between (statistical) security and efficiency.

Introducing $\tau$ 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 $\mathsf{pub}$ 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, it

is assumed that the number of delegated pairings is linearly dependent on $\tau$, i.e., $L \in \Theta(\tau)$. On the other hand, $\varphi$ allows the client to adjust the security and efficiency of the construction, ranging from unconditionally secure but not necessarily efficient ($\varphi = 2 \cdot \lambda$), to everlasting secure against a bounded algebraic adversary, and significantly efficient ($\varphi \ll 2 \cdot \lambda$). KAPR25 is presented as pseudocode in Figure 3.2. In subsection 3.2.1, the code flow is described while in subsection 3.2.2 the correctness property of Definition 3.1.1 is proven.

### 3.2.1 Protocol Description

KAPR25 can be described 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 algorithm OneTimeSetup is run once by the client, and takes as input the auxiliary security parameters $\mathsf{aux.sec} = (\tau, \varphi)$, which are set by the client. The time parameter $\tau$ will be used as a termination flag, while the efficiency one, $\varphi$, 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 $\tau$ 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 $(\vec{A}, \vec{B})$. 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 5-6), linked by the relation $e(U, V) = \xi$. Noticeably, these computations are independent of the pairing inputs, and hence can be precomputed. Next, whether the batch $(\vec{A}, \vec{B})$ consists of a single pair or

**GlobalSetup($\lambda$) $\to$ pp**

// Bilinear Group parameters

**return** $(q, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, P, Q, \gamma_T, e)$

**Setup(sk, $\vec{A}, \vec{B}$) $\to$ (pub, sec)**

1: **parse** $\mathsf{sk} = (s, \cdot, \cdot, \varphi, \cdot)$
2: **if** $\mathtt{len}(\vec{A}) \neq \mathtt{len}(\vec{B})$:
3: **return** $\perp$
4: **if** $\mathcal{O} \in \vec{A} \cup \vec{B}$: **return** $\perp$
5: $u \xleftarrow{\$} \mathbb{Z}_q^*$
6: $(U, V) \leftarrow \left([u]P, [s \cdot u^{-1}]Q\right)$
7: $\mathsf{M} \leftarrow \mathtt{len}(\vec{A})$
8: **if** $\mathsf{M} = 1$:
9: $r \xleftarrow{\$} [\![\min\{2^\varphi, q-1\}]\!]$
10: $C \leftarrow \left[-s \cdot u^{-1}\right](U + A)$
11: $D \leftarrow V - [r]B$
12: $(X, Y) \leftarrow (\perp, \perp)$
13: **else :**
14: $W \xleftarrow{\$} \mathbb{G}_1 \setminus \{\mathcal{O}\}$
15: $D \leftarrow \sum_{j=1}^{\mathsf{M}} B_j$
16: $(X, Y) \leftarrow ([u](D - V), W - U)$
17: **for** $j \in [\![\mathsf{M}]\!]$ :
18: $r_j \xleftarrow{\$} [\![\min\{2^\varphi, q-1\}]\!]$
19: $C_j \leftarrow \left[r_j\right] A_j + W$
20: **if** $\mathcal{O} \in \{\vec{C}, D, X, Y\}$: **return** $\perp$
21: $\mathsf{pub} \leftarrow \left(\vec{A}, \vec{B}, \vec{C}, D, X, Y\right)$
22: $\mathtt{sec} \leftarrow \vec{r}$
23: **return** (pub, sec)

**OneTimeSetup(aux.sec) $\to$ sk**

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

**Compute(pub) $\to$ out**

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

**Verify(sk, sec, out) $\to$ result**

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

**Figure 3.2:** Pseudocode for KAPR25 [66, §5]. The lines in grey are not input-dependent and can be pre-computed to prioritize online efficiency.

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 $(\vec{A}, \vec{B})$ and additional group elements are computed and checked against the infinity point in order to later verify the correctness of the server's output at the end of the interaction round. The public tuple $\left(\vec{A}, \vec{B}, \vec{C}, D, X, Y\right)$ is then sent to the server, while the secrets $\vec{r}$ 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 $j \in [\![\mathsf{M}]\!]$ along with the check value $\gamma$.

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 $\vec{\rho}$ are $\mathbb{G}_T$ elements (in order to avoid small subgroup attacks, as in [79, Section 8.3]). The last check verifies the identity $\xi \stackrel{?}{=} \left(\prod_{j=1}^{\mathsf{M}} \rho_j^{r_j}\right) \cdot \gamma$ (line 6). If all checks pass, the algorithm returns $\vec{\rho}$, else, it returns $\perp$.

**Remark 3.2.1** ($\mathsf{M} = 1$ vs $\mathsf{M} > 1$ 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. Indeed, for $\mathsf{M} = 1$, 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 $\mathsf{M} > 1$, the verification formula allows the client to perform $\mathsf{M}$ short multiplications on $\mathbb{G}_1$ and a single full one on a $\mathbb{G}_2$ point (see [66][Table 2]).

**Remark 3.2.2** (*Completeness*)**.** KAPR25 can be made *complete* (see [14, §1]) for the client, meaning the protocol handles all pairs of input points on the curve, including the point at infinity. For instance, simply skip the processing for those pairs $(A, B)$ containing $\mathcal{O}$ and set their pairing value to 1 later in Verify. Additionally, if $\sum_{j=1}^{\mathsf{M}} B_j = \mathcal{O}$ (Setup line 15), then $(A_k, B_k) \leftarrow \left([2]A_k, [2^{-1}]B_k\right)$ can be set for some $k \in [\![\mathsf{M}]\!]$, which simultaneously ensures $D \neq \mathcal{O}$ and $e(A_k, B_k)$ to remain unchanged. Finally, if any of the points in $\{\vec{C}, X, Y\}$ (Setup line 20) become $\mathcal{O}$ (event occurring with negligible probability in $\lambda$), then AmorE can simply re-sample the necessary random coins ($U, V$ or $W$) and re-compute the affected points. This ensures that the protocol remains unconditionally secure when $\varphi$ is full range; otherwise the system of equations underlying the public view would cease to be underdetermined, and this can be easily detected by an adversary (see section 3.3).

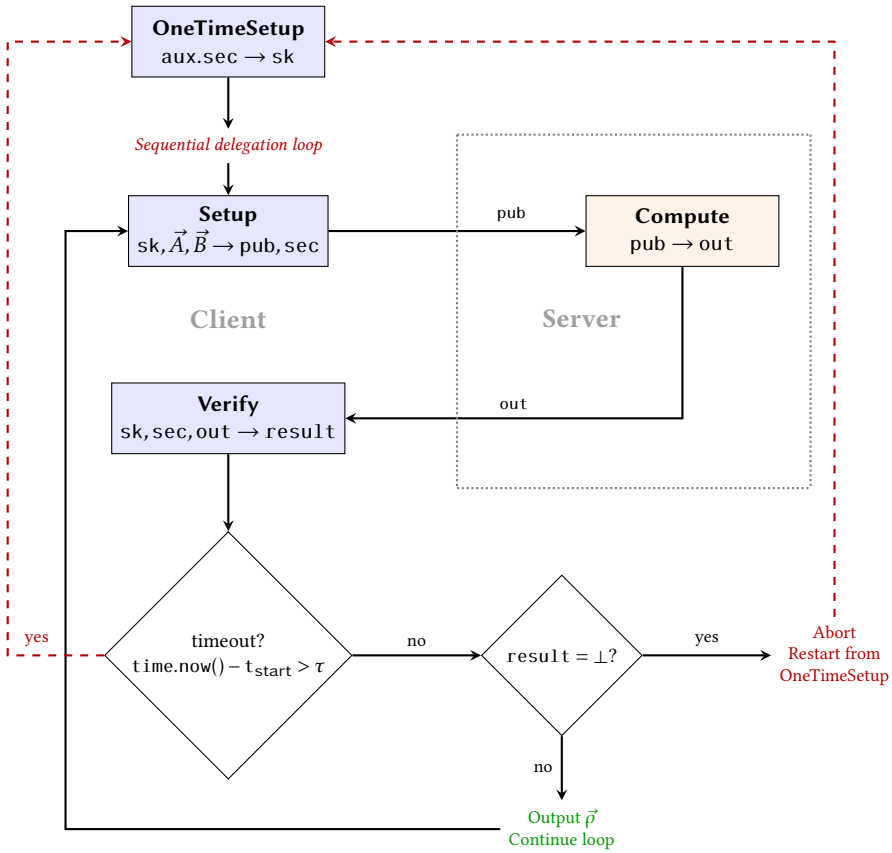


**Figure 3.3:** Flow diagram of the KAPR25 protocol. The `OneTimeSetup` phase is executed once to generate long-term secrets valid for time $\tau$. The sequential delegation loop consists of `Setup` (Client prepares public/secret values), `Compute` (Server evaluates pairings), and `Verify` (Client checks correctness and timeout). If timeout is exceeded or verification fails, the protocol must restart from `OneTimeSetup`.

### 3.2.2 Protocol Correctness

This follows by inspection of the pseudo code in Figure 3.2. In line 3 of Compute, the server computes $\vec{\rho} = \{e(A_j, B_j)\}_{j=1}^{M}$ which is then returned in line 6 of Verify. It remains to show that the verification equation $\xi \stackrel{?}{=} \left(\prod_{j=1}^{\mathsf{M}} \rho_j^{r_j}\right) \cdot \gamma$ in line 6 of Verify, is satisfied when all algorithms are run honestly. There are two cases:

1. $\mathsf{M} = 1$:

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

2. $\mathsf{M} > 1$:

$$\left(\prod_{j=1}^{\mathsf{M}} \rho_j^{r_j}\right) \cdot \gamma = \left(\prod_{j=1}^{\mathsf{M}} e(A_j, B_j)^{r_j}\right) \cdot \left(\prod_{j=1}^{\mathsf{M}} e(C_j, -B_j)\right) \cdot e(Y,D) \cdot e(P,X)$$

$$= \left(\prod_{j=1}^{\mathsf{M}} e(A_j, B_j)^{r_j}\right) \cdot \left(\prod_{j=1}^{\mathsf{M}} e\Big(\big[r_j\big]A_j + W, -B_j\Big)\right) \cdot e(Y,D) \cdot e(P,X)$$

$$= \left(\prod_{j=1}^{\mathsf{M}} e(A_j, B_j)^{r_j}\right) \cdot \left(\prod_{j=1}^{\mathsf{M}} e(A_j, B_j)^{-r_j}\right) \cdot \prod_{j=1}^{\mathsf{M}} e(W, B_j)^{-1} \cdot e(Y,D) \cdot e(P,X)$$

$$= e\left(W, \sum_{j=1}^{\mathsf{M}} B_j\right)^{-1} \cdot e\left(W - U, \sum_{j=1}^{\mathsf{M}} B_j\right) \cdot e\left(P, [u]\left(\sum_{j=1}^{\mathsf{M}} B_j - V\right)\right)$$

$$= e\left(-U, \sum_{j=1}^{\mathsf{M}} B_j\right) \cdot e\left(U, \sum_{j=1}^{\mathsf{M}} B_j - V\right)$$

$$= e(U, -V) = e([u]P, [-su^{-1}]Q) = \xi$$

Hence, for a given number $L \in \Theta(\tau)$ of pairing delegations, and for any choice of delegation rounds $\mathsf{N}$ and corresponding batch sizes $\vec{\mathsf{M}}$ satisfying $L = \mathbf{1}^T \cdot \vec{\mathsf{M}}$, if all algorithms are run honestly, KAPR25 is $L$-correct as per Definition 3.1.1.

## 3.3 Security

**Proof technique overview:**

- In Lemma 3.3.1 (subsection 3.3.1), it is proved that an algebraic adversary $\mathcal{A}$ attempting to produce a forgery against KAPR25's verification equation must produce, in the simplest case, a pair $(\eta, \eta^{r_j})$ for some $\eta \in \mathbb{G}_T \setminus \{1\}$ and $j \in [\![\mathsf{M}]\!]$.
- Next, in subsection 3.3.2 it is shown that in an efficient setting $\varphi < 2 \cdot \lambda$, extracting a secret scalar $r_j$ from the public view $\mathsf{pub}$ reduces to finding the intersection of at least two sets of size $2^{\varphi}$ in one of the bilinear groups $\mathbb{G}_\iota$.
- Moreover, in subsection 3.3.3 it is proved that when $\mathcal{A}$ is limited to computing $2^{\kappa}$ many $\varphi$-bit group operations for a time-dependent parameter $\kappa \in \Theta(\log(\tau))$, setting $\varphi = \left\lceil \frac{\sigma-1}{2} + \kappa \right\rceil$ allows upperbounding the success probability of finding this intersection by $2^{-\sigma}$, where $\sigma$ is the statistical security parameter. In practice, $\kappa = 70 + \log(\tau)$ ($\tau$ 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 3.3.4 and subsection 3.3.5, an unconditional security result stated in Theorem 3.3.4 and a computational one stated in Theorem 3.3.5 are provided.

### 3.3.1 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 3.3.1** (Identifying forgeries). *Let $\lambda$ 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:*

$$\left(\xi = \prod_{i=1}^{\mathsf{M}} \rho_i^{r_i} \cdot \gamma\right) \wedge (\rho_i \in \mathbb{G}_T)_{i \in [\![\mathsf{M}]\!]} \tag{3.1}$$

*for some $\mathsf{M} \geq 1$, with $\xi \in \mathbb{G}_T$ and $\vec{r}$ being the client's secret values, and $\mathtt{out} = (\vec{\rho}, \gamma) \in \mathbb{G}_T^{\mathsf{M}} \times \mathbb{G}_T$ the output of an honest server during a delegation; then, for any $\mathtt{out}^* \neq \mathtt{out}$ satisfying* (3.1)*, there exists a non-empty subset of indexes $J \subset [\![\mathsf{M}]\!]$, and elements $\eta_j \in \mathbb{G}_T \setminus \{1\}$ for $j \in J$ such that $\mathtt{out}^* = (\vec{\rho}^*, \gamma^*)$ with:*

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

*In particular, an algebraic adversary can only produce forgeries of this form. Hence, if it succeeds in producing a valid forgery $\mathtt{out}^*$, it must output at least one secret exponent $r_j$ for some index $j \in J$.*

*Proof.* By the closure property of the multiplicative group $\mathbb{G}_T$, any (forgery) vector $\mathtt{out}^* = (\vec{\rho}^*, \gamma^*) \neq \mathtt{out}$ can be written as a deviation from the honest output: $\mathtt{out} \cdot \vec{\eta}$, for some $\vec{\eta} \in \mathbb{G}_T^{\mathsf{M}} \times \mathbb{G}_T$. Since $\mathtt{out}^* \neq \mathtt{out}$, at least two entries of $\vec{\eta}$ must be different from 1. Let $J \subset [\![\mathsf{M}]\!]$ 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 $\eta_{\mathsf{M}+1} = \prod_{j \in J} \eta_j^{-r_j}$. This fact is immediate, since any other value of $\eta_{\mathsf{M}+1}$ would lead to $\mathtt{out}^*$ not satisfying the verification check in (3.1), indeed by the equality in (3.1):

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

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

### 3.3.2 Extracting partially-masking secret variables

In this section is illustrated how an otherwise unconditionally secure protocol becomes vulnerable 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 KAPR25, the reduced set is $[\![2^\varphi]\!] \subset \mathbb{Z}_q^*$ with $\varphi < 2 \cdot \lambda$ (see lines 9 and 18 in Setup in Figure 3.2).

**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

$$\mathsf{pub} = \begin{cases} C = [r_1]U + X \\ D = [r_2]U + Y \end{cases}, \tag{3.2}$$

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, is obtained the following system of equations:

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

which yields the parametrization

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

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 unconditionally 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

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

For a computationally bounded algebraic adversary, attempting to learn the secret tuple from (3.2) reduces to picking an element

$$U^* \in \mathcal{I} = \left\{[t_1^{-1}](C - X) \colon t_1 \in S\right\} \cap \left\{[t_2^{-1}](D - Y) \colon t_2 \in S\right\} \tag{3.3}$$

and its corresponding indexes $r_1^*, r_2^*$, which takes up to $2 \cdot |S|$ computations. In the worst case (best case for $\mathcal{A}$), when $|\mathcal{I}| = 1$, finding (3.3) 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

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

which is most efficiently done via Diffie-Hellman's *meet-in-the-middle* attack in [45], taking up to $2^{k+1}$ operations. Figure 3.4 illustrates this parallelism.

**Remark 3.3.2.** 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

$$t \cdot (d - y) \cdot (c - x)^{-1} \mod q \stackrel{?}{\in} S$$

which takes up to $|S|$ tries.

**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 $(\eta, \eta^t)$ 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 $(\eta, \eta^r)$ for a secret scalar $r$ as long as the public view is underdetermined.

### 3.3.3 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

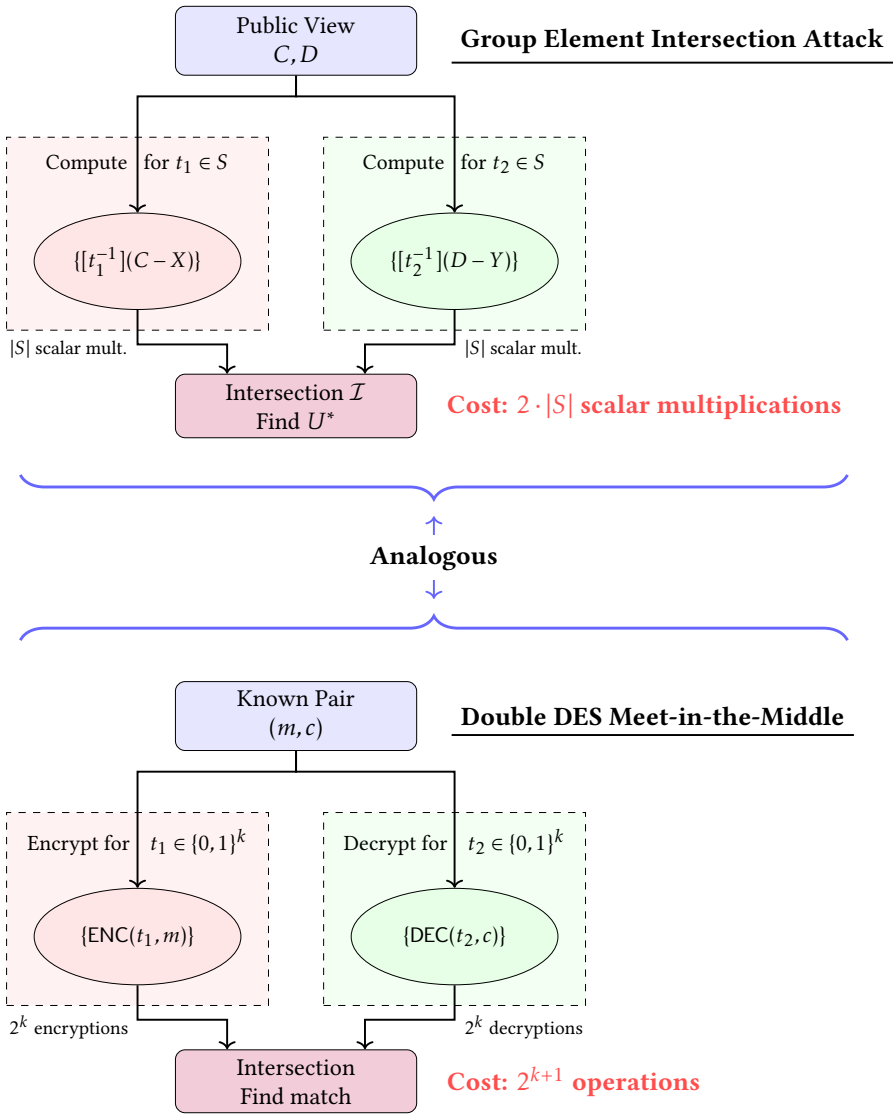


**Figure 3.4:** Analogy between the intersection attack on partially-masking secret variables (above) and Diffie-Hellman's meet-in-the-middle attack (below) on Double DES encryption (adapted from [45]).

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 (3.3) containing one of the client's secret values, without exceeding $2^\kappa$ group operations.

In our *toy example* from subsection 3.3.2, 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

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

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 3.3.3** (Intersection success probability)**.** *Let $\varphi, \kappa, \sigma, \mathsf{M}$ be positive integers such that*

$$\varphi = \left\lceil \frac{\sigma - 1}{2} + \kappa \right\rceil.$$

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

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

*Proof.* Let $c, n \in \mathbb{N}$ and $g : \mathbb{R}^n \to \mathbb{R}$ defined as $g(\vec{x}) = \sum_{j=1}^n x_i - c$. Consider

a tuple $\{x_j\}_{j=1}^n \subset \mathbb{R}^+$ fulfilling the constraint $g(\vec{x}) = 0$. Then, it holds that

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

$$\Leftrightarrow \quad \sum_{1 \le j < k \le n} x_j \cdot x_k = \frac{1}{2} \cdot \left( \left( \sum_{j=1}^{n} x_j \right)^2 - \sum_{j=1}^{n} x_j^2 \right)$$

$$= \frac{1}{2} \cdot \left( (g(\vec{x}) + c)^2 - \sum_{j=1}^{n} x_j^2 \right) = \frac{1}{2} \cdot \left( c^2 - \sum_{j=1}^{n} x_j^2 \right),$$

from which it can established that

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

where the last equality follows from using a Lagrange multiplier to include the constraint $g(\vec{x}) = 0$ in the minimization. Under the new assignment $x_i \leftarrow |S_i|$ and $(c, n) \leftarrow (2^\kappa, N)$, [1] one can write

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

Now,

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

$$= \sum_{1 \le j < k \le \mathsf{M}} \frac{\binom{2^\varphi - 1}{\left|S_j\right| - 1}}{\binom{2^\varphi}{\left|S_j\right|}} \cdot \frac{\binom{2^\varphi - 1}{|S_k| - 1}}{\binom{2^\varphi}{|S_k|}} = 2^{-2\varphi} \cdot \sum_{1 \le j < k \le \mathsf{M}} \left|S_j\right| \cdot |S_k|$$

$$< 2^{-2\varphi + 2\kappa - 1} = 2^{-\sigma}.$$

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

□

### 3.3.4 Unconditional security

KAPR25 achieves unconditional security when the efficient parameter $\varphi$ is set to $2 \cdot \lambda$. This setting simply implies sampling $r$ from $\mathbb{Z}_q^*$ in lines 9 and 18 of Setup in Figure 3.2.

**Theorem 3.3.4.** *For any $\lambda$, 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* $\mathsf{aux.sec} = (\varphi, \tau)$, *with $\varphi = 2 \cdot \lambda$, $\tau > 0$ and $L := \mathsf{poly}(\lambda)$. Then, KAPR25 (Figure 3.2) is unconditionally secure, equipped with $2 \cdot \lambda$ bits of statistical security. In particular, for any algebraic adversary $\mathcal{A}$, it holds that:*

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

*where $\mathbf{Exp}^{\mathsf{seq,CI-ver}}_{\mathbf{\Pi},\mathcal{A}}$ is depicted in Figure 3.1.*

*Proof.* Consider an adversary $\mathcal{A}$ entering the delegation round $i \in [\![\mathsf{N}]\!]$ of the experiment $\mathbf{Exp}^{\mathsf{seq,CI-ver}}_{\mathbf{\Pi},\mathcal{A}}$. KAPR25's verification equation and membership tests (lines 5 and 6 of Verify in Figure 3.2) are of the same type as those specified in Lemma 3.3.1, which ensures that $\mathcal{A}$ wins the experiment if and only if, it succeeds in producing in the simplest case, a triplet $(\eta, \eta^r, j) \in \mathbb{G}_T^2 \times [\![\mathsf{M}_i]\!]$ for some secret exponent $r$ used to verify the element $\rho_{ij}$. 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:

$$\begin{cases} C_{kj} &= \Big[r_{kj} a_{kj} + w_k\Big] P \\ Y_k &= [w_k - u_k] P \\ X_k &= \Big[u_k \cdot \sum_{j=1}^{\mathsf{M}_k} b_{kj} - s\Big] Q, \end{cases} \qquad \begin{cases} C_\ell &= \Big[-s \cdot \Big(1 + u_\ell^{-1} \cdot a_\ell\Big)\Big] P \\ D_\ell &= \Big[s \cdot u_\ell^{-1} - r_\ell \cdot b_\ell\Big] Q \end{cases} \tag{3.4}$$

where $\ell$ ranges over a subset of indexes $\mathcal{L} \subset [\![i]\!]$ identifying the delegation rounds consisting of a single pairing delegation ($\mathsf{M}_\ell = 1$), and $(k, j) \in \{[\![i]\!] \setminus \mathcal{L}\} \times [\![\mathsf{M}_k]\!]$. Now, (3.4) can be transformed into the following parametrization in $\mathbb{Z}_q$ (modulo $q$): [2]

[2] Recall that KAPR25 rejects points at infinity (see lines 4 and 20 in Setup); this implies

$$\begin{cases} u_k &= (x_k + s) \cdot \left(\sum_{j=1}^{\mathsf{M}_k} b_{kj}\right)^{-1} \\ w_k &= y_k + (x_k + s) \cdot \left(\sum_{j=1}^{\mathsf{M}_k} b_{kj}\right)^{-1} \\ r_{kj} &= \left(c_{kj} - y_k - (x_k + s) \cdot \left(\sum_{j=1}^{\mathsf{M}_k} b_{kj}\right)^{-1}\right) \cdot a_{kj}^{-1} \\ u_\ell &= \left(-c_\ell \cdot s^{-1} - 1\right)^{-1} \cdot a_\ell \\ r_\ell &= \left((-c_\ell - s) \cdot a_\ell^{-1} - d_\ell\right) \cdot b_\ell^{-1}. \end{cases} \tag{3.5}$$

In (3.5), each value of $s$ yields a secret tuple $(s, \vec{W}, \vec{\mathbf{r}}, \vec{u})$ that produces the view (3.4), as long as $\mathcal{O} \notin \vec{W}$ and $0 \notin \{\vec{\mathbf{r}}, \vec{u}\}$ [3]. Hence, for any $j \in \mathsf{M}_i$, there are at least $q - 1 - (\sum_{k=1}^{i} \mathsf{M}_k + 2 \cdot i - |\mathcal{L}|)$ possible and equiprobable secret values $r_{ij}$ that could have been sampled according to $\mathcal{A}$'s view (recall that $\left|\vec{W}\right| = i - |\mathcal{L}|$). Since this number decreases as the number of delegated pairings increases, it can be upperbound by $q - 1 - 3 \cdot L$. □

### 3.3.5 Everlasting security

Security for KAPR25 in the efficient setting $\varphi < 2 \cdot \lambda$, can be proven against an adversary operating under the Algebraic Group Model [47], and with limited computational power during the execution of the sequential pairing delegation protocol (recall that $\mathcal{A}$ can only win $\mathbf{Exp}^{\mathsf{seq,Cl-ver}}_{\mathbf{\Pi},\mathcal{A}}$ *between the start and the end of the protocol*). In particular, the protocol timeout parameter $\tau$, determines an additional computational parameter $\kappa \in \Theta(\log(\tau))$, such that $\mathcal{A}$ is limited to computing a total of $2^\kappa$ elements across all groups $\mathbb{G}_1, \mathbb{G}_2$ and $\mathbb{G}_T$.

**Theorem 3.3.5.** *For any $\lambda$, let $q$ denote the prime order of the groups output by any execution of* GlobalSetup($\lambda$). *Let $\tau, \sigma$ be positive integers, and $\kappa \in \Theta(\log(\tau))$ satisfying $\varphi = \left\lceil \frac{\sigma-1}{2} + \kappa \right\rceil < 2 \cdot \lambda$. Then, for any positive integer $L \in \Theta(\tau)$, KAPR25 (Figure 3.2) is secure according to $\mathbf{Exp}^{\mathsf{seq,Cl-ver}}_{\mathbf{\Pi},\mathcal{A}}$ against an*

---

that in order not to lose in the security experiment, $\mathcal{A}$ has to pick inputs $\vec{\mathbf{A}}$ and $\vec{\mathbf{B}}$ that do not contain points at infinity and such that $\sum_{j=1}^{\mathsf{M}_k} B_{kj} \neq \mathcal{O}$ (hence all $a_{kj}$ and $\sum_{j=1}^{\mathsf{M}_k} b_{kj}$ are in $\mathbb{Z}_q^*$ and therefore invertible).

[3] The setting $\varphi = 2 \cdot \lambda$ implies that $r_{kj} \xleftarrow{\$} \mathbb{Z}_q^*$ in lines 9 and 18 of Setup, so no non-zero values can be discarded in (3.5) for the secrets $(\vec{w}, \vec{\mathbf{r}}, \vec{u})$.

*algebraic adversary $\mathcal{A}$ limited to computing a total of $2^{\kappa}$ elements across all groups $\mathbb{G}_{\iota}$ for $\iota \in \{1, 2, T\}$. In particular,*

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

*Proof.* Given a pair $(\mathsf{N}, \vec{\mathsf{M}})$ such that $L = \mathbf{1}^{T} \cdot \vec{\mathsf{M}}$, assume $\mathcal{A}$ enters the delegation round $i \in [\![\mathsf{N}]\!]$ of $\mathbf{Exp}^{\mathsf{seq,CI-ver}}_{\mathbf{\Pi},\mathcal{A}}$. Recall from Lemma 3.3.1 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 (3.4), $\mathcal{A}$ 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 $\vec{z}, \vec{\bar{z}} \subset \mathbb{Z}_q^*$. Additionally, these points can be used as inputs for the bilinear map $e$. Now, since $\mathcal{A}$'s view (3.4) 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 $\vec{z}^{*}, \vec{\bar{z}}^{*} \subset \mathbb{Z}_q^*$. As described in subsection 3.3.2, the constraints $r_{kj} \in [\![2^{\varphi}]\!]$ can be leveraged in this scenario by discarding values via an intersection on sets containing a common secret. 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 (3.4), the aforementioned constraints manifest themselves in several relations, e.g.,

$$\xi = \gamma_T^{s} \in \bigcap_{\ell \in \mathcal{L}} \left\{\rho_{\ell}^{t_{\ell}} \cdot \gamma_{\ell} \colon t_{\ell} \in [\![2^{\varphi}]\!]\right\} \subset \mathbb{G}_T, \tag{3.6}$$

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 $u_{\ell}$ from $C_{\ell}$ and $D_{\ell}$ in (3.4) 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

$$\gamma_T^{z \cdot \hat{z} \cdot s} \in \bigcap_{\ell \in \mathcal{L}} \{\eta^{z \cdot s - \bar{z} \cdot r_\ell \cdot b_\ell} \cdot \eta^{\bar{z} \cdot t \cdot b_\ell} \colon t \in [\![2^\varphi]\!]\} \subset \mathbb{G}_T \,. \tag{3.7}$$

However, due to the sequential design of KAPR25, $\mathcal{A}$ 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 (3.6) or (3.7), and win $\mathbf{Exp}_{\mathbf{\Pi},\mathcal{A}}^{\mathsf{seq,Cl-ver}}$ with non negligible probability whenever $\varphi \ll 2 \cdot \lambda$. If $\mathcal{L} = \emptyset$, the only place where the relevant bounded secret exponents $r_{ij}$ appear in the view (3.4), is in the expression $C_{ij} = \left[r_{ij} a_{ij} + w_i\right] P$, which in turn enables the intersection

$$W_i \in \bigcap_{j=1}^{\mathsf{M}} \left\{C_{ij} - [t_j] A_{ij} \colon t_j \in [\![2^\varphi]\!]\right\} \subset \mathbb{G}_1 \,. \tag{3.8}$$

Now, as expected from the analysis in subsection 3.3.2, 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 $\mathbf{Exp}_{\mathbf{\Pi},\mathcal{A}}^{\mathsf{seq,Cl-ver}}$ equals $|\mathcal{I}|^{-1}$. However, for the worst case $|\mathcal{I}| = 1$, the winning probability of an adversary limited to computing $2^\kappa$ 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 $|S| + \left|\bar{S}\right| \leq 2^\kappa$. 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 (3.8) yields in the worst case (best case for $\mathcal{A}$)

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

after which $\mathcal{A}$ extracts both $r_{ij}$ and $r_{ik}$. For the setting $\varphi = \left\lceil \frac{\sigma-1}{2} + \kappa \right\rceil$, Lemma 3.3.3, upperbounds the probability of this event by $2^{-\sigma}$ for any fixed subset selection by $\mathcal{A}$ . □

# 4

# Amortized Efficiency for Pairing Delegation with Private Inputs

This chapter presents a novel way of converting a type-PVPV into types SVSV, SVPV and PVSV with minimal extra computation, and presents novel and KAPR25-like constructions for single and batch delegation that satisfy SI-verification and IND-privacy. The proving techniques are essentially the same as the ones introduced in [66] but leverage the security parameter $\lambda$ instead of the efficiency parameter $\varphi$.

In what follows, the fundamental challenges in adapting KAPR25 to provide input privacy while maintaining efficiency are discussed, the privacy notions IND-privacy and OW-privacy are generalized to the sequential setting, and new constructions are proposed in the Type-SVSV single and batch delegation settings.

## 4.1 Transitioning from public to private input delegation

### 4.1.1 On the suitability of a timeout parameter

In the type-PVPV construction KAPR25, (section 3.2), the client uses a timeout parameter $\tau$ to limit the server's computation time, rendering the probability of success of the meet-in-the-middle attacks described in subsection 3.3.2 negligible in the statistical parameter $\sigma$. In the public input case, this provides sufficient security: even if the server extracts the secret ephemeral scalars after returning its output, it can no longer affect the client's verification. However, in the private input setting, extracting these scalars post-interaction could allow the server to learn the client's inputs, breaking the input-privacy property. Extending the timeout to the lifetime of the server instead would mean increasing the space $[\![2^{\varphi}]\!]$ from which the "short" ephemeral scalars are sampled, resulting in an increased client cost. Indeed, recall that

$$\varphi = \left\lceil \frac{\sigma - 1}{2} + \log(\mathcal{H}_{Bitcoin} \cdot \tau) \right\rceil$$

where $\mathcal{H}_{Bitcoin}$ represents the current highest hashrate of the Bitcoin network and $\tau$ is in seconds (in [66], $\sigma = 40$, $\tau = 1$ and $\varphi = 90$). Increasing the timeout $\tau$ to, say, 100 years, would add around 31 bits to $\varphi$ which is still acceptable, but then one would also have to predict the evolution of $\mathcal{H}_{Bitcoin}$ over this period, which is highly uncertain, especially if one takes into con-

sideration the potential advent of quantum computing (although elliptic-curve cryptography is not post-quantum secure, so pairing delegation would become useless in this scenario). A natural solution to this problem would be to set $\varphi = \lambda$, the security parameter of the scheme, which would guarantee that meet-in-the-middle attacks are infeasible for any adversary bounded by $\lambda$. However, this would significantly increase the client's cost, as the number of group operations would now scale linearly with $\lambda$ which typically takes the values $128, 192$ and $256$ as opposed to $90$.

### 4.1.2 Additional masking

In KAPR25 (Figure 3.2), only two $\mathbb{G}_2$ full scalar multiplications are required on the client side for delegating a batch of pairings, regardless of the batch size. This is one of the key reasons behind KAPR25's overall efficiency. However, hiding the inputs necessitates additional scalar multiplications.

As shown in Lemma 1.6.4, a protocol must hide not only the inputs but also the pairing output value. For instance, delegating the pairing evaluation on $(C, D) \leftarrow ([r^{-1}]A, [r]B)$ for $r \xleftarrow{\$} \mathbb{Z}_q^*$, even though it provides $2\lambda$ bits of min-entropy to each of the inputs and their cartesian product, since $e(C, D) = e(A, B)$, the pairing value remains exposed, allowing an adversary to trivially win the IND-privacy experiment.

Canard et al. [29] pointed out a simple method for converting a type-PVPV protocol into a type-SVSV that also satisfies IND-input-privacy: given private inputs $(A, B)$ and a type-PVPV protocol $\mathbf{\Pi}$, the client can

1. Sample secret scalars $u, v \xleftarrow{\$} \mathbb{Z}_q^*$,
2. Compute $(C, D) \leftarrow ([u]A, [v]B)$,
3. Obtain $\rho \leftarrow e(C, D)$ via $\mathbf{\Pi}$,
4. Recover the desired output by computing $e(A, B) = \rho^{(u \cdot v)^{-1}}$.

This technique is of course, information theoretically hiding for the inputs and the output, but it comes at the cost of two additional full scalar multiplications on the client side per input, one in $\mathbb{G}_1$ and one in $\mathbb{G}_2$ and a full exponentiation in $\mathbb{G}_T$ for recovering the output. Symbolically, for any type-PVPV protocol $\mathbf{\Pi}$, the cost of its type-SVSV adaptation is given by

$$\mathsf{cost}(\mathtt{to\_SVSV}(\mathbf{\Pi})) = \mathsf{cost}(\mathbf{\Pi}) + \mathfrak{m}_1 + \mathfrak{m}_2 + \mathfrak{m}_T,$$

where $\mathsf{cost}(\boldsymbol{\Pi})$ measures the average computational cost per pairing delegated via $\boldsymbol{\Pi}$. This additional cost can quickly outweigh the efficiency gains achieved by $\boldsymbol{\Pi}$ and ruin any prospect of amortized efficiency in the type-SVSV setting.

### 4.1.3 A fine-grained approach

Lemma 3.3.1 stands as an essential result for proving the verifiability of a protocol that uses a check of the form

$$\xi \stackrel{?}{=} \prod_{i=1}^{\mathsf{M}} \rho_i^{r_i} \cdot \gamma \,.$$

And not surprisingly, virtually every verifiable pairing delegation uses at least one exponentiation in $\mathbb{G}_T$ per delegating pairing (see chapter 2). So, instead of blindly applying Canard et al.'s masking technique to a type-PVPV protocol, one could leverage the existing exponentiation in $\mathbb{G}_T$ used for verifiability to simultaneously mask the pairing value. In particular, the client may construct a public view that hides both the inputs and the pairing evaluation, and later retrieve the pairing value by performing the necessary exponentiation in $\mathbb{G}_T$ during verification.

This idea is of course case-specific and requires careful rearranging of the involved secret scalars for ensuring verifiability, input-privacy and the much-desired overall efficiency.

## 4.2 Input-Privacy models in the Sequential Setting

Recall from subsection 1.6.2 the two fundamentally different notions of input privacy:

- *IND-privacy* (indistinguishability), where the adversary attempts to distinguish between two chosen input pairs.
- *OW-privacy* (one-wayness), where the adversary attempts to recover an input from prior knowledge of its sampling distribution.

These notions capture orthogonal security guarantees and may be independently satisfied by a protocol.

**IND Input-Privacy:** The formalization of Canard et al. [29] can be easily adapted to the sequential setting for type-SVSV delegation, as presented by the experiment $\mathbf{Exp}^{\mathsf{seq,IND-priv}}_{\mathbf{\Pi},\mathcal{A}}$ shown in Figure 4.1. Intuitively, the experiment allows an adversary to choose up to $\mathsf{N}$ delegations and adaptively select pairs of input vectors $(\vec{A_0}, \vec{B_0})$ and $(\vec{A_1}, \vec{B_1})$ of its choice. The adversary's goal is to *distinguish* which inputs were used: it wins if, after $\min(\mathsf{N}, L)$ delegations, it can correctly identify a round $i^*$ and the corresponding bit $b_{i^*}$ used in that round.

**OW Input-Privacy:** The OW-privacy experiment is parameterized by the protocol's privacy type $\mathsf{type} \in \{\mathsf{SVPV}, \mathsf{PVSV}, \mathsf{SVSV}\}$, which determines which inputs the adversary must recover. At the beginning of each round $i$, for each input designated as secret, the adversary selects a $\lambda$-min-entropy distribution from which it is sampled. The adversary wins based on the privacy type if it can recover at least one of the secret inputs used at any round:

- $\mathsf{type} = \mathsf{SVPV}$ (A secret, B public): The adversary must recover some $A \in \vec{A}$ used in any round.
- $\mathsf{type} = \mathsf{PVSV}$ (A public, B secret): The adversary must recover some $B \in \vec{B}$ used in any round.
- $\mathsf{type} = \mathsf{SVSV}$ (both secret): The adversary must recover at least one of $A \in \vec{A}$ or $B \in \vec{B}$ from any round.

The corresponding experiment $\mathbf{Exp}^{\mathsf{seq,OW-priv}}_{\mathbf{\Pi},\mathcal{A}}$ is shown in Figure 4.1.

Both experiments capture the sequential nature of the delegation protocol, as the adversary can adaptively choose inputs (or input-distributions) based on previous delegations. Crucially, the adversary may learn non-negligible information about inputs from earlier rounds at any point of the protocol's execution and use it at a later stage.

Definition 4.2.1 formalizes the notions of input privacy for sequential pairing delegation.

**Definition 4.2.1** (IND and OW Input Privacy for sequential pairing delegation)**.** For a given positive integer $L \geq 1$ , a computational security parameter $\lambda$, and a tuple of auxiliary security parameters $\mathtt{aux.sec}$, a sequential delegation protocol $\mathbf{\Pi}$ of type $\mathsf{type} \in \{\mathsf{SVPV}, \mathsf{PVSV}, \mathsf{SVSV}\}$ is said to be *IND-private*

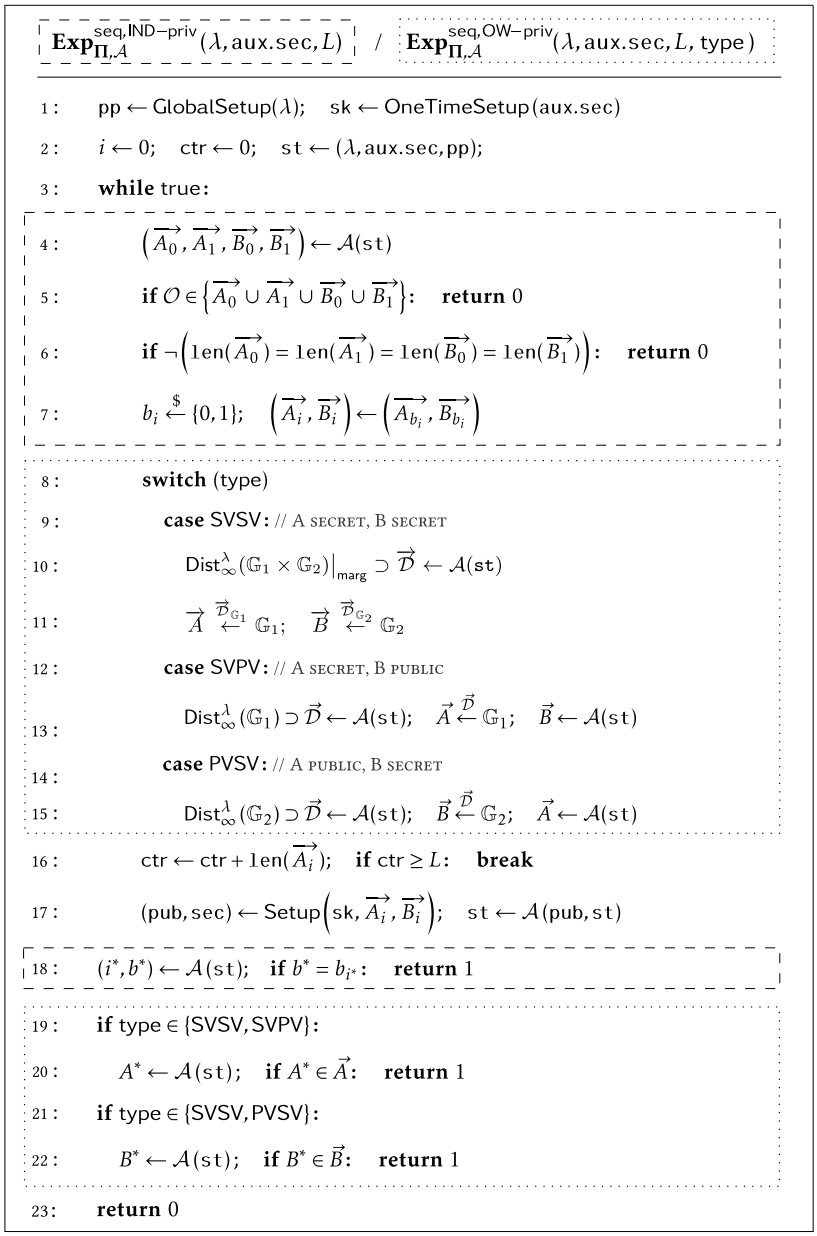


**Figure 4.1:** Input-privacy experiments for sequential pairing delegation. The IND variant (left title) is obtained by excluding the dot-boxed code, while the OW variant (right title) is obtained by excluding the dash-boxed code.

(resp. *OW-private*) against a given class of adversaries if, for any $\mathcal{A}$ in the class, it holds that:

$$\mathcal{P}\Big[\mathbf{Exp}^{\mathsf{seq,IND-priv}}_{\mathbf{\Pi},\mathcal{A}}(\lambda, \mathsf{aux.sec}, L) = 1\Big] \leq \frac{1}{2} + \mathsf{negl}(\lambda) + \sum_{\delta \in \mathsf{aux.sec}} \mathsf{negl}(\delta)$$

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

where $\mathbf{Exp}^{\mathsf{seq,IND-priv}}_{\mathbf{\Pi},\mathcal{A}}$ and $\mathbf{Exp}^{\mathsf{seq,OW-priv}}_{\mathbf{\Pi},\mathcal{A}}(\lambda, \mathsf{aux.sec}, L, \mathsf{type})$ denote the IND and OW input-privacy experiments shown in Figure 4.1.

## 4.3 A New Construction for Single Pairing Delegation accepting the privacy types SVSV, SVPV and PVSV

The proposed construction in the single pairing delegation setting, is designated as "K26-s" following the established protocol naming convention ('-s' stands for *single*) and presented in Figure 4.2. The protocol achieves SI-verifiability and IND-input-privacy while requiring fewer scalar multiplications than prior work, as summarized in Table 2.3.

### 4.3.1 Correctness

See Table 4.1 for the correctness proofs of K26 in all three input privacy settings. Hence, if the server is honest, the verification check in Verify line 6 of Figure 4.2 always passes.

### 4.3.2 Verifiability and Input-Privacy

Theorem 4.3.1 establishes SI-verifiability and IND-privacy for the proposed protocol. The proof technique, which relies on subsection 3.3.2 and Lemma 3.3.1, is the same as for KAPR25.

**Theorem 4.3.1** (Verifiability and Input-Privacy)**.** *K26-s Figure 4.2 is a single pairing delegation protocol that is SI-verifiable and OW-private against algebraic adversaries whose running time is at most* poly *$(\lambda)$.*

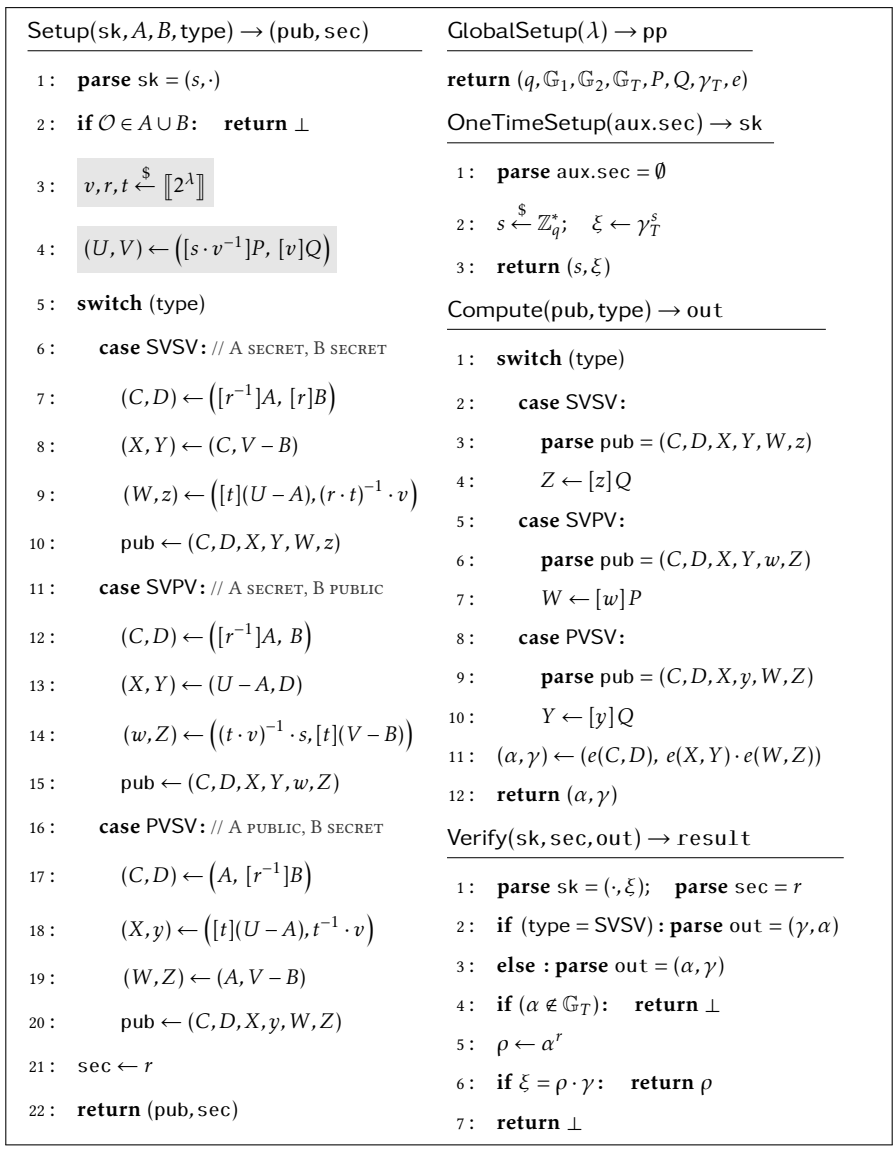


**Figure 4.2:** Pseudocode for the single pairing delegation protocol K26 with support for the privacy types SVSV (both inputs secret), SVPV ($A$ secret, $B$ public), and PVSV ($A$ public, $B$ secret). The lines in grey are not input-dependent and can be pre-computed to prioritize online efficiency. The code flow is identical to the one depicted in Figure 3.3.

| Type | $\rho^r \cdot \gamma = e(C,D)^r \cdot e(X,Y) \cdot e(W,Z)$ |
|---|---|
| SVSV | $= e\Big(\Big[r^{-1}\Big]A, [r]B\Big)^r \cdot e\Big(\Big[r^{-1}\Big]A, V-B\Big)e\Big([t](U-A), [(r\cdot t)^{-1}\cdot v]Q\Big)$<br>$= e(A,B)\cdot e(A,V-B)e(U-A,V) = e(U,V) = \xi$ |
| SVPV | $= e\Big([r^{-1}]A, B\Big)^r \cdot e(U-A,B)e\Big([(t\cdot v)^{-1}\cdot s]P, [t](V-B)\Big)$<br>$= e(A,B)\cdot e(U-A,B)e(U,V-B) = e(U,V) = \xi$ |
| PVSV | $= e\Big(A, [r^{-1}]B\Big)^r \cdot e\Big([t](U-A), \Big[t^{-1}\cdot v\Big]Q\Big)e(A,V-B)$<br>$= e(A,B)\cdot e(U-A,V)e(A,V-B) = e(U,V) = \xi$ |

**Table 4.1:** Correctness proofs for single pairing delegation protocols.

*Proof.* At any round $i$ of the experiments $\mathbf{Exp}^{\mathsf{seq,ver}}_{\mathbf{\Pi},\mathcal{A}}$ and $\mathbf{Exp}^{\mathsf{seq,OW-priv}}_{\mathbf{\Pi},\mathcal{A}}$, the public view of the delegation consists of a tuple of points and scalars from which parametrizations in $\mathbb{Z}_q$ for the secret values can be derived in terms of the long-term secret $s$ and the public values[1]:

SVSV($A$ **and** $B$ **secret**) $\mathsf{pub} = \{C_k, D_k, Y_k, w_k, Z_k\}_{k\in[\![i]\!]}$:

$$\begin{cases} c_k &= r_k^{-1}\cdot a_k \\ d_k &= r_k \cdot b_k \\ y_k &= v_k - b_k \\ w_k &= t_k\cdot(v_k^{-1}\cdot s - a_k) \\ z_k &= (r_k\cdot t_k)^{-1}\cdot v_k \end{cases} \Rightarrow \begin{cases} v_k &= \mathsf{f}_k(s)\cdot(s-c_k\cdot d_k)^{-1} \\ b_k &= d_k\cdot \mathsf{g}_k\cdot(s-c_k\cdot d_k)^{-1} \\ t_k &= \mathsf{g}_k\cdot\mathsf{f}_k(s)\cdot\Big(z_k\cdot(s-c_k\cdot d_k)^2\Big)^{-1} \\ a_k &= c_k\cdot(s-c_k\cdot d_k)\cdot\mathsf{g}_k^{-1} \\ r_k &= (s-c_k\cdot d_k)\cdot\mathsf{g}_k^{-1} \end{cases}$$

where $\mathsf{f}_k(s) = d_k\cdot w_k\cdot z_k + s\cdot y_k$ and $\mathsf{g}_k = c_k\cdot y_k + w_k\cdot z_k$. Each $s' \in \mathbb{Z}_q^*\backslash\{c_k\cdot d_k\}$ yields a complete secret tuple $(t'_k, v'_k, a'_k, b'_k, r'_k)$ that explains the public views across all rounds.

[1] see subsection A.3.1 for a formal verification using the SymPy Python library

$\boxed{\mathsf{SVPV}(A\ \textbf{secret}, B\ \textbf{public})}$ $\mathsf{pub} = \{C_k, X_k, w_k, Z_k\}_{k\in[\![i]\!]}$:

$$\begin{cases} c_k &= r_k^{-1}\cdot a_k \\ x_k &= v_k^{-1}\cdot s - a_k \\ w_k &= (t_k\cdot v_k)^{-1}\cdot s \\ z_k &= t_k\cdot(v_k - b_k) \end{cases} \Rightarrow \begin{cases} v_k &= b_k\cdot s\cdot(s - w_k\cdot z_k)^{-1} \\ a_k &= (s - w_k\cdot z_k - b_k\cdot x_k)\cdot b_k^{-1} \\ t_k &= (s - w_k\cdot z_k)\cdot(b_k\cdot w_k)^{-1} \\ r_k &= (s - w_k\cdot z_k - b_k\cdot x_k)\cdot(b_k\cdot c_k)^{-1} \end{cases}$$

Each $s' \in \mathbb{Z}_q^* \setminus \{w_k\cdot z_k, w_k\cdot z_k + b_k\cdot x_k\}$ yields a complete secret tuple $(v_k', a_k', t_k', r_k')$ that explains the public views across all rounds.

$\boxed{\mathsf{PVSV}(A\ \textbf{public}, B\ \textbf{secret})}$ $\mathsf{pub} = \{D_k, X_k, y_k, Z_k\}_{k\in[\![i]\!]}$:

$$\begin{cases} d_k &= r_k^{-1}\cdot b_k \\ x_k &= t_k\cdot(v_k^{-1}\cdot s - a_k) \\ y_k &= t_k^{-1}\cdot v_k \\ z_k &= v_k - b_k \end{cases} \Rightarrow \begin{cases} v_k &= (s - x_k\cdot y_k)\cdot a_k^{-1} \\ b_k &= (s - x_k\cdot y_k - a_k\cdot z_k)\cdot a_k^{-1} \\ t_k &= (s - x_k\cdot y_k)\cdot(a_k\cdot y_k)^{-1} \\ r_k &= (s - x_k\cdot y_k - a_k\cdot z_k)\cdot(a_k\cdot d_k)^{-1} \end{cases}$$

Each $s' \in \mathbb{Z}_q^* \setminus \{x_k\cdot y_k, x_k\cdot y_k + a_k\cdot y_k\cdot z_k\}$ yields a complete secret tuple $(v_k', b_k', t_k', r_k')$ that explains the public views across all rounds.

If all secret scalars were full-range, the above parametrizations would establish information-theoretic OW-privacy. However, $v_k, r_k, t_k$ are sampled from the reduced set $[\![2^\lambda]\!] \subset \mathbb{Z}_q^*$, and whenever one of the inputs is secret, it possesses $\lambda$ bits of min-entropy as per the SI-verifiability experiment $\mathbf{Exp}^{\mathsf{seq,ver}}_{\mathbf{\Pi},\mathcal{A}}$ (instead of the $2\cdot\lambda$ bits of entropy that a randomly sampled input would produce), hence any secret variable in the system is either full-range or *partially masking*. This is the same setting in which CI-verifiability was proven in Theorem 3.3.5 for KAPR25 and the argument for proving SI-verifiability is the same: bounding secret variables in an underdetermined system of equations adds constraints which are exploited most efficiently by a meet-in-the-middle attack taking on average a number of $\mathbb{G}_T$ exponentiations equal to the cardinality of the bounded sampling set. In detail, by Lemma 3.3.1 (*Identifying forgeries*), an algebraic adversary can only produce a forgery against a verification check of the form $(\alpha \stackrel{?}{\in} \mathbb{G}_T) \wedge (\xi \stackrel{?}{=} \alpha^r\cdot\gamma)$ –which is the verification check in lines line 4 and line 6 of Verify in Figure 4.2– if it can output the scalar $r$. Now, from subsection 3.3.2 (*Extracting partially-masking secret variables*), the most efficient way for an algebraic

adversary to recover a scalar $r_k$ in this setting is to perform a meet-in-the-middle attack that requires, on average $\left|\left[\!\left[2^\lambda\right]\!\right]\right| = 2^\lambda$ exponentiations in $\mathbb{G}_\iota$, i.e., exponential in the security parameter $\lambda$. In all three privacy settings, the added constraint can be expressed as the membership condition

$$\xi \in \left\{\alpha_j^x \cdot \gamma_j \colon x \in \left[\!\left[2^\lambda\right]\!\right]\right\} \bigcap \left\{\alpha_k^x \cdot \gamma_k \colon x \in \left[\!\left[2^\lambda\right]\!\right]\right\} \text{ for } j \neq k \in [\![i]\!] \,.$$

In the worst case (that happens with overwhelming probability), the above intersection consists of the singleton $\{\xi\}$, obtained precisely when the exponents $x$ take the values $r_j$ and $r_k$, found after an average of $2^\lambda$ tries. Hence, the probability that an algebraic $\mathcal{A}$ outputs any of the $r_k$ values within a time bound polynomial in $\lambda$ is negligible in $\lambda$. A similar reasoning applies to OW-privacy for all three privacy types: the best strategy for an algebraic adversary to recover a secret input from the public view is to find elements belonging to the following intersections:

- SVSV:

$$\left\{[x^{-1}]D \colon x \in \left[\!\left[2^\lambda\right]\!\right]\right\} \bigcap \mathsf{supp}\left(\mathcal{D}_{\mathbb{G}_2}\right)$$

- SVPV:

$$\left\{[x]C \colon x \in \left[\!\left[2^\lambda\right]\!\right]\right\} \bigcap \mathsf{supp}\left(\mathcal{D}\right) \text{ and}$$
$$\left\{e(C,D)^x \colon x \in \left[\!\left[2^\lambda\right]\!\right]\right\} \bigcap \left\{e\left(A^*, B\right) \colon A^* \in \mathsf{supp}(\mathcal{D})\right\}$$

- PVSV:

$$\left\{[x^{-1}]D \colon x \in \left[\!\left[2^\lambda\right]\!\right]\right\} \bigcap \mathsf{supp}\left(\mathcal{D}\right) \text{ and}$$
$$\left\{e(C,D)^x \colon x \in \left[\!\left[2^\lambda\right]\!\right]\right\} \bigcap \left\{e\left(A, B^*\right) \colon B^* \in \mathsf{supp}(\mathcal{D})\right\}$$

These are all standard search problems, and the probability that an element is found in either intersection after a $\mathsf{poly}(\lambda)$ number of tries (each requiring at least one scalar multiplication in $\mathbb{G}_1, \mathbb{G}_2$, or $\mathbb{G}_T$), is negligible in $\lambda$. □

| Protocol | Client Cost |
|---|---|
| **K26-svsv-s** | $\mathsf{cost}(\text{KAPR25-s}) + \mathfrak{m}_1 + \mathfrak{m}_2 - \mathfrak{m}_2^{\mathsf{sh}}$ |
| **K26-svpv-s** | $\mathsf{cost}(\text{KAPR25-s}) + \frac{1}{2}\mathfrak{m}_2 - \mathfrak{m}_2^{\mathsf{sh}}$ |
| **K26-pvsv-s** | $\mathsf{cost}(\text{KAPR25-s}) + \mathfrak{m}_2 - \mathfrak{m}_1^{\mathsf{sh}} - \mathfrak{m}_2^{\mathsf{sh}}$ |

**Table 4.2:** Symbolic added cost per delegated pairing to the state-of-the-art KAPR25-s ($\mathsf{M} = 1$).

### 4.3.3 Efficiency

Compared to single type-PVPV pairing delegation via KAPR25 (Figure 3.2, $\mathsf{M} = 1$), K26 achieves input privacy with minimal overhead, as shown in Table 4.2.

This is achieved by conveniently offloading a $\mathbb{G}_2$ masking operation to the server and leveraging a single exponentiation that is used both in the verification phase and to obtain the pairing evaluation (note that in KAPR25, the pairing is directly returned by the server and exponentiating only serves the purpose of verifying).

## 4.4 A New Construction for Batch Pairing Delegation accepting the privacy types SVSV, SVPV and PVSV

The type-PVPV batch delegation in Figure 3.2 (case $\mathsf{M} > 1$) can be 'lifted' to types SVSV, SVPV and PVSV by using the fine-grained approach described in subsection 4.1.3. Specifically, as with K26-s, a single $\mathbb{G}_T$ exponentiation per pairing can be leveraged in the Verify phase to simultaneously *verify and compute* the pairing from the server's output. To do this, a different choice of maskings needs to be made even though the bilinearity of KAPR25's verification formula itself can be left untouched.

Figure 4.3 presents the proposed construction, which, like K26-s, achieves SI-verifiability and IND-input-privacy. As summarized in Table 2.3, this construction requires fewer scalar multiplications than prior work in the batch setting.

### 4.4.1 Correctness

This is first verified for the type-PVSV construction, and then argued for the other two privacy types. For type-PVSV, the correctness of the verification formula is shown as follows:

$$\left(\prod_{j=1}^{\mathsf{M}} \rho_j^{r_j}\right) \cdot \gamma = \left(\prod_{j=1}^{\mathsf{M}} e(C_j, D_j)\right) \cdot \left(\prod_{j=1}^{\mathsf{M}} e(X_j, -D_j)\right) \cdot e(Y, F) \cdot e(P, Z)$$

$$= \left(\prod_{j=1}^{\mathsf{M}} e\left(\left[(r_j \cdot t_j)^{-1}\right] A_j, \left[t_j\right] B_j\right)^{r_j} \cdot e\left([t_j^{-1}] A_j + W, -\left[t_j\right] B_j\right)\right) \cdot e(Y, F) \cdot e(P, Z)$$

$$= \left(\prod_{j=1}^{\mathsf{M}} e\left(A_j, B_j\right) \cdot e\left(-[t_j^{-1}] A_j, \left[t_j\right] B_j\right) \cdot e\left(-W, \left[t_j\right] B_j\right)\right) \cdot e(Y, F) \cdot e(P, Z)$$

$$= \left(\prod_{j=1}^{\mathsf{M}} e\left(-W, \left[t_j\right] B_j\right)\right) \cdot e\left(W - U, \sum_{j=1}^{\mathsf{M}} \left[t_j\right] B_j\right) \cdot e\left(P, [v^{-1} \cdot s]\left(\sum_{j=1}^{\mathsf{M}} [t_j] B_j - V\right)\right)$$

$$= \left(\prod_{j=1}^{\mathsf{M}} e\left(-W, \left[t_j\right] B_j\right)\right) \cdot \left(\prod_{j=1}^{\mathsf{M}} e\left(W - U, [t_j] B_j\right)\right) \cdot \left(\prod_{j=1}^{\mathsf{M}} e\left(U, [t_j] B_j\right)\right) \cdot e(U, -V)$$

$$= \left(\prod_{j=1}^{\mathsf{M}} e\left(-U, [t_j] B_j\right) \cdot e\left(U, [t_j] B_j\right)\right) \cdot e(U, -V) = e(U, -V) = \xi\,.$$

The same verification formula is used for the privacy types SVSV and SVPV with the slight modifications:

- In type-SVPV, $B$ is public, and therefore $t_j$ is set to 1 for all $j \in [\![\mathsf{M}]\!]$.
- In type-SVSV, $t_j$ is constant for each input pair $\left(A_j, B_j\right)$.

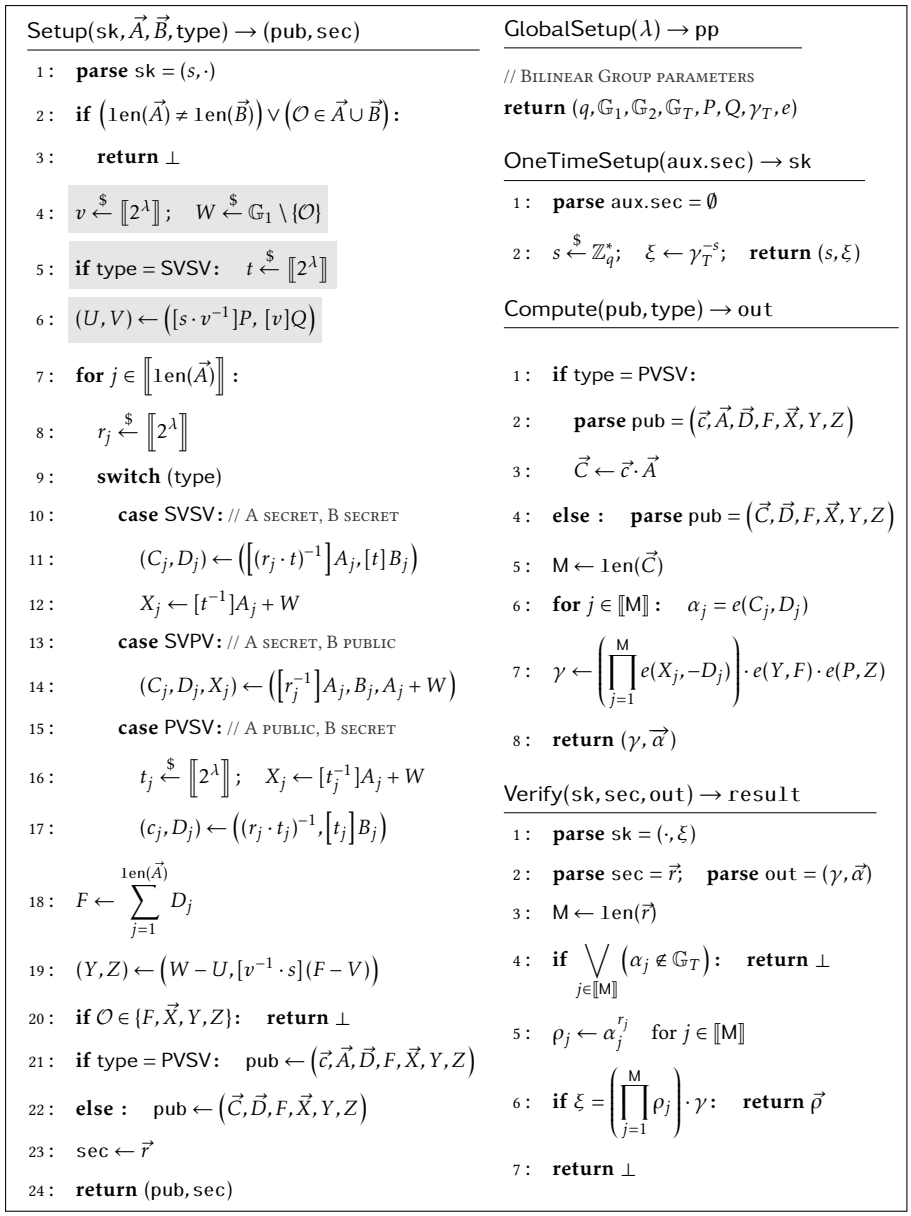


**Figure 4.3:** Pseudocode for the K26-batch protocol supporting three privacy types: SVSV (both $A$ and $B$ secret), SVPV ($A$ secret, $B$ public), and PVSV ($A$ public, $B$ secret).

### 4.4.2 Verifiability and Input-Privacy

**Theorem 4.4.1.** *K26-b Figure 4.3 is a batch pairing delegation protocol that is SI-verifiable and OW-private against algebraic adversaries whose running time is at most* poly $(\lambda)$.

*Proof.* The proof follows the same structure as Theorem 4.3.1. It suffices to show that the public view at any round $i$ admits exponentially many (in $\lambda$) valid secret explanations. This is now shown for each privacy type[2]:

$\boxed{\text{SVSV}(A \textbf{ and } B \textbf{ secret})}$ $\mathsf{pub} = \{\mathsf{pub}_k\}_{k\in[\![i]\!]} = \left\{\left\{C_j, D_j, X_j\right\}_{j\in[\![\mathsf{M}_k]\!]}, Y_k, Z_k\right\}_{k\in[\![i]\!]}$:

$$\begin{cases} c_j &= (t_k \cdot r_j)^{-1} \cdot a_j \\ d_j &= t_k \cdot b_j \\ x_j &= t_k^{-1} a_j + w_k \\ y_k &= w_k - u_k \\ z_k &= u_k \cdot \left(\sum_{j=1}^{\mathsf{M}_k} d_j\right) - s \end{cases} \Rightarrow \begin{cases} u_k &= (s + z_k) \cdot f_k^{-1} \\ a_j &= t_k \cdot \left(f_k \cdot (x_j - y_k) - s - z_k\right) \cdot f_k^{-1} \\ b_j &= d_j \cdot t_k^{-1} \\ r_j &= \left(f_k \cdot (x_j - y_k) - s - z_k\right) \cdot \left(c_j \cdot f_k\right)^{-1} \\ w_k &= (f_k \cdot y_k + s + z_k) \cdot f_k^{-1} \end{cases}$$

where $f_k = \sum_{j=1}^{\mathsf{M}_k} d_j$. Hence, each pair of secrets

$$(s', t_k') \in \left(\mathbb{Z}_q^* \setminus \left(\{-z_k - y_k \cdot f_k\} \cup \bigcup_{j=1}^{\mathsf{M}_k} \{f_k \cdot (x_j - y_k) - z_k\}\right)\right) \times \mathbb{Z}_q^*$$

yields a secret tuple $(s', t_k', u_k', A_j', B_j', r_j', W_k')$ that explains the public views $\mathsf{pub}_k$ for $k \in [\![i]\!]$.

$\boxed{\text{SVPV}(A \textbf{ secret}, B \textbf{ public})}$ $\mathsf{pub} = \left\{\left\{C_j, X_j\right\}_{j\in[\![\mathsf{M}_k]\!]}, Y_k, Z_k\right\}_{k\in[\![i]\!]}$:

$$\begin{cases} c_j &= r_j^{-1} \cdot a_j \\ x_j &= a_j + w_k \\ y_k &= w_k - u_k \\ z_k &= u_k \cdot f_k - s \end{cases} \Rightarrow \begin{cases} u_k &= (s + z_k) \cdot f_k^{-1} \\ a_j &= x_j - y_k - (s + z_k) \cdot f_k^{-1} \\ r_j &= \left(x_j - y_k - (s + z_k) \cdot f_k^{-1}\right) \cdot c_j^{-1} \\ w_k &= y_k + (s + z_k) \cdot f_k^{-1} \end{cases}$$

[2] see subsection A.3.2 for a formal verification using the SymPy Python library

Each secret $s' \in \mathbb{Z}_q^* \setminus \left(\{-z_k - y_k \cdot f_k\} \cup \bigcup_{j=1}^{\mathsf{M}_k} \{f_k \cdot (x_j - y_k) - z_k\}\right)$ yields a secret tuple $(s', u_k', A_j', r_j', W_k')$ that explains the public views $\mathsf{pub}_k$ for $k \in [\![i]\!]$.

$\boxed{\mathsf{PVSV}(A \textbf{ public}, B \textbf{ secret})}$ $\mathsf{pub} = \left\{\left\{C_j, D_j, X_j\right\}_{j \in [\![\mathsf{M}_k]\!]}, Y_k, Z_k\right\}_{k \in [\![i]\!]}$:

$$\begin{cases} c_j &= (t_j \cdot r_j)^{-1} \cdot a_j \\ d_j &= t_j \cdot b_j \\ x_j &= t_j^{-1} \cdot a_j + w_k \\ y_k &= w_k - u_k \\ z_k &= u_k \cdot f_k - s \end{cases} \Rightarrow \begin{cases} u_k &= (s + z_k) \cdot f_k^{-1} \\ b_j &= d_j \cdot \left(f_k \cdot (x_j - y_k) - s - z_k\right) \cdot \left(a_j \cdot f_k\right)^{-1} \\ t_j &= -a_j \cdot f_k \cdot \left(f_k \cdot (x_j - y_k) - s - z_k\right)^{-1} \\ r_j &= \left(f_k \cdot (x_j - y_k) - s - z_k\right) \cdot \left(c_j \cdot f_k\right)^{-1} \\ w_k &= (f_k \cdot y_k + s + z_k) \cdot f_k^{-1} \end{cases}$$

where $f_k = \sum_{j=1}^{\mathsf{M}_k} d_j$. Each secret

$$s' \in \mathbb{Z}_q^* \setminus \left(\{-z_k - y_k \cdot f_k\} \cup \bigcup_{j=1}^{\mathsf{M}_k} \{f_k \cdot (x_j - y_k) - z_k\}\right)$$

yields a secret tuple $(s', u_k', B_j', t_j', r_j', W_k')$ that explains the public views $\mathsf{pub}_k$ for $k \in [\![i]\!]$.

□

### 4.4.3 Efficiency

Table 4.3 compares the symbolic cost of K26-b to the state-of-the-art KAPR25 ($\mathsf{M} > 1$) [66].

## 4.5 Important Remarks

### 4.5.1 Limitations of K26

CI-verifiability cannot be proven using the techniques developed so far: if the adversary is allowed to choose the pairing inputs, the system generated by the public view becomes overdetermined, which may lead to more efficient attacks than meet-in-the-middle. Similarly, for IND-privacy, finding

| Protocol | Client Cost |
|---|---|
| **K26-svsv-b** | $\mathsf{cost}(\text{KAPR25-b}) + 2\mathfrak{m}_1 + \frac{1}{2}(\mathfrak{m}_2 + \mathfrak{m}_T) - \mathfrak{m}_1^{\mathsf{sh}} - \mathfrak{m}_T^{\mathsf{sh}}$ |
| **K26-svpv-b** | $\mathsf{cost}(\text{KAPR25-b}) + \mathfrak{m}_1 + \frac{1}{2}\mathfrak{m}_T - \mathfrak{m}_1^{\mathsf{sh}} - \mathfrak{m}_T^{\mathsf{sh}}$ |
| **K26-pvsv-b** | $\mathsf{cost}(\text{KAPR25-b}) + \mathfrak{m}_1 + \frac{1}{2}(\mathfrak{m}_2 + \mathfrak{m}_T) - \mathfrak{m}_1^{\mathsf{sh}} - \mathfrak{m}_T^{\mathsf{sh}}$ |

**Table 4.3:** Symbolic additional cost per pairing compared to the state-of-the-art KAPR25-b, which represents the batch version of KAPR25 ($\mathsf{M} > 1$).

reduced-sized discrete logarithms using Pollard's rho algorithm is enough for winning $\mathbf{Exp}^{\mathsf{seq,IND-priv}}_{\mathbf{\Pi},\mathcal{A}}$ ($2^{\lambda/2}$ operations). These observations lead to a notable theoretical drawback in the designs of K26-s and K26-b: the protocols either satisfy both SI-verifiability and OW-privacy, or neither of them [3] . While such property interdependence is typically avoided in cryptography, on the practical side, this interdependence is what enhances the much-desired efficiency as will be shown in the next section.

### 4.5.2 Protocol composability

K26-s and K26-b are composable, i.e., can be combined into one protocol that runs the common OneTimeSetup phase once and bifurcates naturally into each delegation type. This can be seen from the general proof argument: as long as the public view consists of an underdetemined system of equations with unknowns having at least $\lambda$ min entropy, SI-verifiability and OW-privacy holds for computationally bounded adversaries. Hence, switching from single to batch delegation types and viceversa always adds at least one extra degree of freedom in the resulting public tuple.

### 4.5.3 Long term efficiency

As opposed to the type-PVPV construction KAPR25 [66, Sec. 5], no timeout parameter $\tau$ is used to indicate that the protocol should be restarted. This is

[3] This could be easily mitigated in practice by integrating a timeout parameter into the verifiability experiment. In this way, winning the input-privacy experiment at any point after the delegation has been (forcefully) completed will not affect the verifiability experiment result.

simply because the secrets in the private constructions are designed to resist any adversary attack developed during the adversary's lifetime rather than the duration of a short client-server interaction as proposed in KAPR25. This means that OneTimeSetup can be *truly run once*, allowing the protocol's overall efficiency to converge exclusively to the online cost (consisting of Setup and Verify).

## 4.6 Optimizing KST23's Type-SVSV Batch Pairing Delegation

In [62, §3.6], Kalkar et al. proposed the only type-SVSV batch pairing delegation scheme found in the literature, KST23. While this scheme is reported as inefficient by the authors (delegating a batch costs roughly twice the cost of computing it locally), it is possible to optimize their protocol by applying the same ideas used by Aranha et al. in APR21[6, §4] to optimize the Crescenzo et al. construction CKKS20 [42, §3].

Figure 4.4 compares the Compute and Verify phases in KST23's original protocol with their optimized versions, which can be described as follows:

### 4.6.1 Compute phase

The server's computational load is reduced by producing $\mathsf{M}+1$ pairings instead of $2\cdot\mathsf{M}+1$. Specifically, the values $\gamma_i$ and $\theta^{-1}$ can be multiplicatively aggregated into a single value $\gamma$, which suffices for the client's subsequent verification. This improves the server's performance since a product of pairings can be computed with a single final exponentiation, as shown in [78, 90]. As a result, the output bandwidth is reduced by $\mathsf{M}$ elements in $\mathbb{G}_T$, which further improves client-side efficiency.

### 4.6.2 Verify phase

The verification equation is simplified as follows:

$$\theta \stackrel{?}{=} \left(\prod_{i=1}^{\mathsf{M}} \rho_i^{r_i}\right)\cdot\left(\prod_{i=1}^{\mathsf{M}} \gamma_i\right)\cdot\xi \quad \Rightarrow \quad \xi^{-1} \stackrel{?}{=} \left(\prod_{i=1}^{\mathsf{M}} \rho_i^{r_i}\right)\cdot\gamma'. \tag{4.1}$$

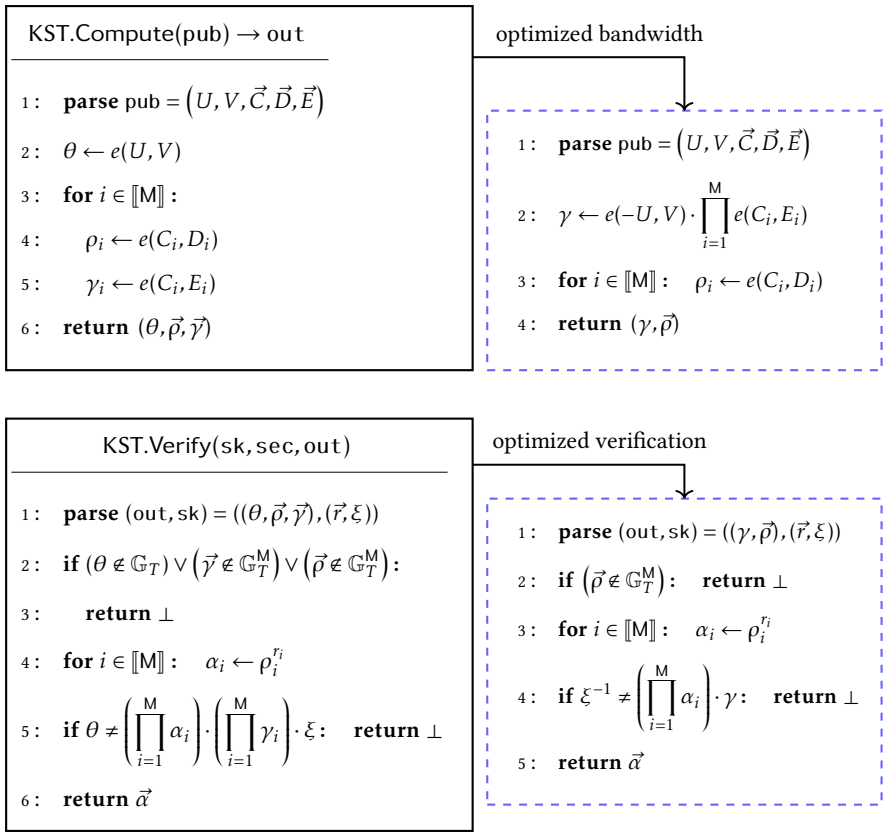


**Figure 4.4:** Comparison of KST23[62, §3.6] and its optimized version à la LOVE [6].

where $\xi \in \mathbb{G}_T$ is precomputed by the client and $\gamma' = \gamma \cdot \theta^{-1}$ is the new value returned by the server. According to Lemma 3.3.1, an algebraic adversary attempting to forge a valid output for this new verification equation (RHS of Equation 4.1) must output one of the ephemeral scalars $r_i$. Thus, provided that $\vec{r}$ is not leaked from the unmodified public tuple pub, verifiability is preserved. Furthermore, input privacy is unaffected since the public view is unchanged. The new verification equation saves the client $\mathsf{M}$ group operations in $\mathbb{G}_T$. Finally, and most importantly, the client's number of membership tests in $\mathbb{G}_T$ is reduced from $2 \cdot \mathsf{M} + 1$ to $\mathsf{M}$ by checking $\vec{\rho} \stackrel{?}{\in} \mathbb{G}_T^{\mathsf{M}}$. The closure property of a group ensures that if $\vec{\rho} \in \mathbb{G}_T^{\mathsf{M}}$, then $\gamma \in \mathbb{G}_T$ must hold for the

RHS of Equation 4.1 to be satisfied.

$$\left(\xi = \prod_{i=1}^{\mathsf{M}} \rho_i^{r_i} \cdot \gamma\right) \wedge (\rho_i \in \mathbb{G}_T)_{i \in [\![\mathsf{M}]\!]} \tag{4.2}$$

For improved referencing, this optimized protocol will be referred to as "KST23-Opti".

# 5

# Performance Results

This chapter presents the performance evaluation of pairing delegation protocols. For comparison with prior work, concrete costs are derived from symbolic expressions in Table 2.3 using bilinear group operation benchmarks obtained with the RELIC toolkit [4], run on the widely used curves BLS12-381, BLS24-509, and BLS48-575. Additionally, direct benchmark comparisons between the KAPR25 and K26 protocol implementations are presented, providing empirical validation of the efficiency improvements achieved by the proposed schemes.

The concrete computational costs presented in the histograms throughout this section are derived from the symbolic expressions in Table 2.3 using the methodology detailed in section A.1.

## 5.1 Type-PVPV Protocols ($A$ and $B$ Public)

Type-PVPV protocols are the ones requiring less client computation. The histograms below compare all PVPV protocols from Table 2.3 across the three selected curves, with the diagonal-patterned baseline bar representing the cost of local pairing computation for reference.

**Single Pairing Delegation** Figure 5.1 compares the average client cost per pairing for single delegation protocols CDS14 [29], CKKS20 [42], APR21 [6], and KAPR25 [66]. The KAPR25 protocol consistently achieves the lowest client cost across all curves, with efficiency gains ranging from 25.4% (BLS24-509) to 44.8% (BLS48-575) compared to local pairing computation.

**Batch Pairing Delegation** Figure 5.2 compares the average client cost per pairing for batch delegation protocols MV19 [73], CKC23 [41], and KAPR25 [66] across batch sizes $\mathsf{M} \in \{2, 3, 10\}$. The KAPR25 protocol achieves the lowest cost for all batch sizes and curves, with efficiency gains ranging from 43.4% ($\mathsf{M} = 2$, BLS24-509) to 76.9% ($\mathsf{M} = 10$, BLS48-575) compared to local computation. The efficiency improvement increases with larger batch sizes, demonstrating the protocol's strong amortization properties.

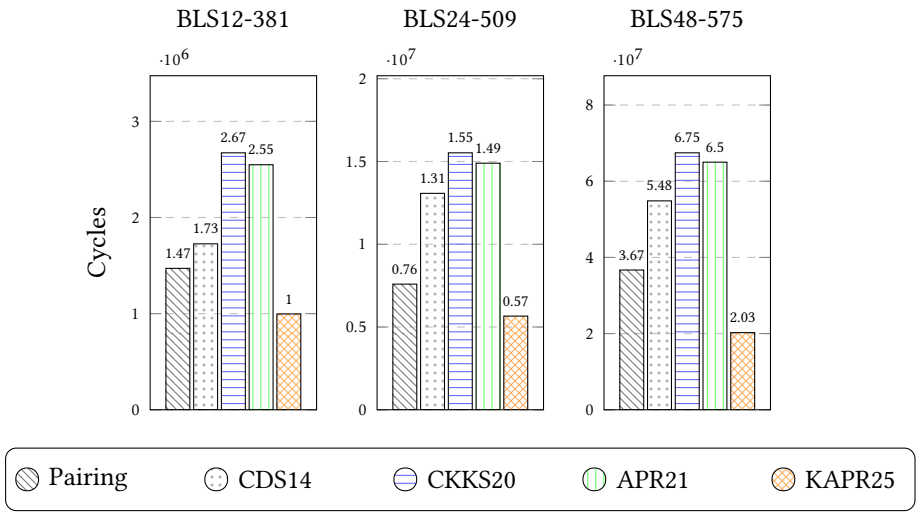


**Figure 5.1:** Single pairing delegation cost comparison for type PVPV.

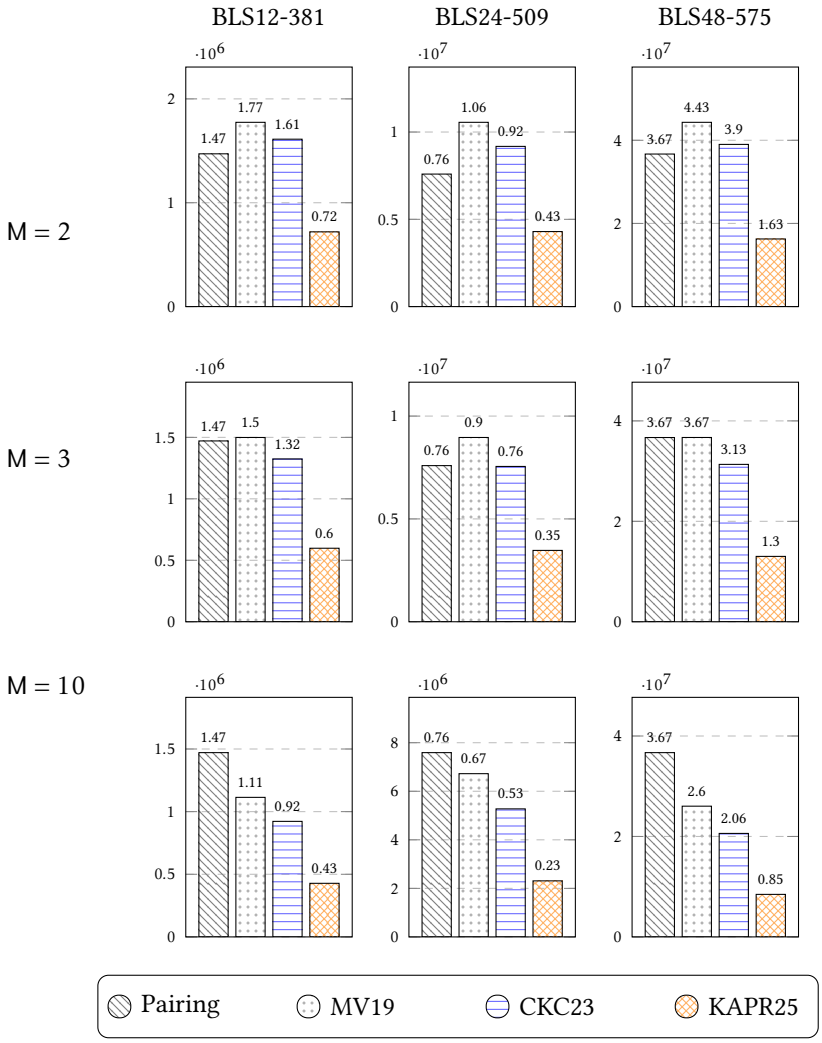


**Figure 5.2:** Batch pairing delegation cost comparison for type PVPV.

## 5.2 Type-SVSV Protocols ($A$ and $B$ Secret)

Type-SVSV protocols provide input privacy by masking both pairing inputs from the server. This additional security requirement typically incurs higher computational costs compared to PVPV protocols.

**Single Pairing Delegation** Figure 5.3 compares the average client cost per pairing for single delegation protocols CDS14 [29], CKKS20 [42], APR21 [6], and the proposed K26-svsv-s protocol. The K26-svsv-s protocol achieves efficiency gains ranging from 24.6% (BLS24-509) to 40.1% (BLS48-575) compared to local pairing computation, while all other protocols in this category exceed the cost of local computation.

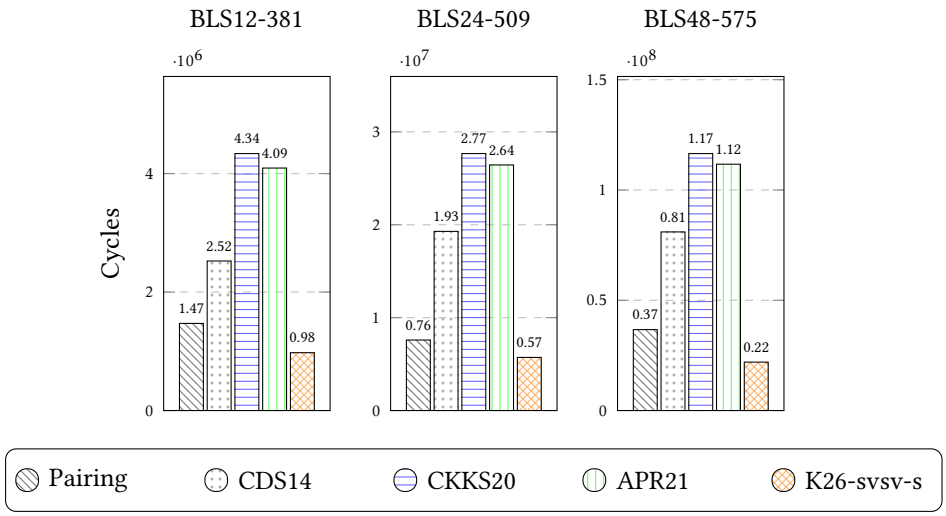


**Figure 5.3:** Single pairing delegation cost comparison for type SVSV.

**Batch Pairing Delegation** Figure 5.4 compares the average client cost per pairing for batch delegation protocols KST23 [62], its optimized variant KST23-Opti, and the proposed K26-svsv-b protocol across batch sizes $M \in \{2, 3, 10\}$. The K26-svsv-b protocol achieves the lowest cost for all batch sizes and curves, with efficiency gains ranging from 17.4% ($M = 2$, BLS24-509) to 50.9% ($M = 10$, BLS48-575) compared to local computation. Notably, the

original KST23 protocol and its optimized variant fail to achieve efficiency gains over local computation for most configurations.

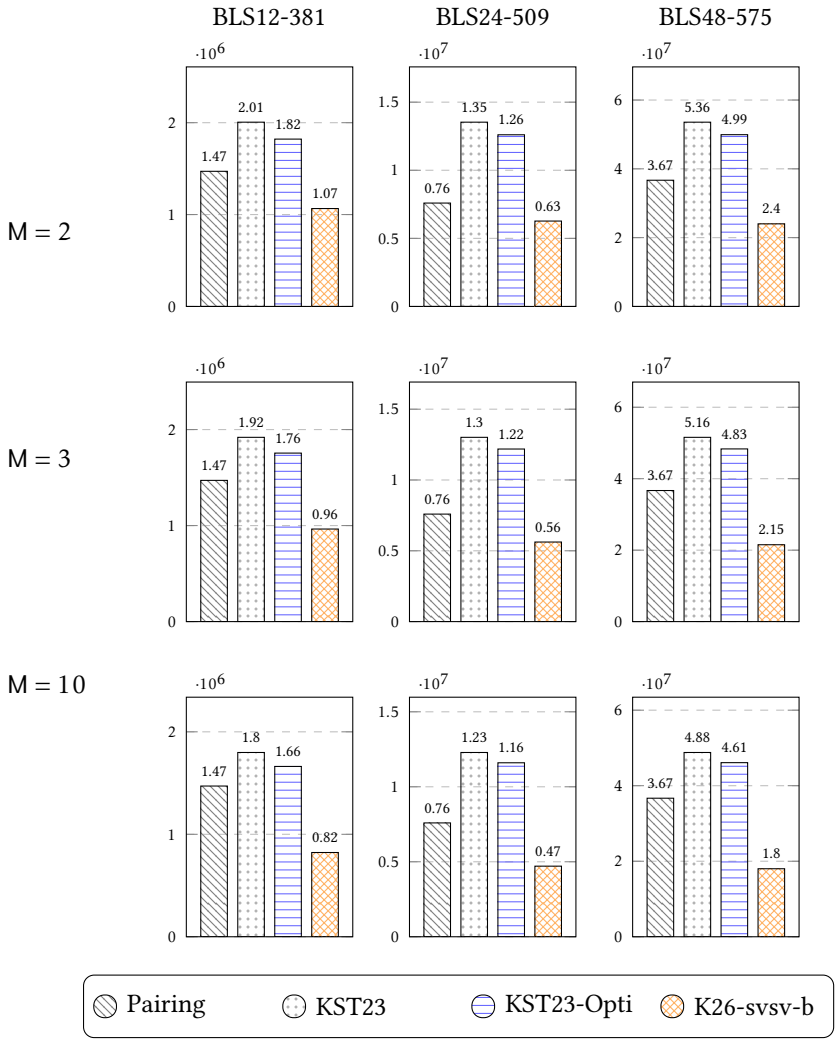


**Figure 5.4:** Batch pairing delegation cost comparison for type SVSV.

## 5.3 Type-SVPV Protocols ($A$ Secret, $B$ Public)

**Single Pairing Delegation** Figure 5.5 compares the average client cost per pairing for single delegation protocols CMCNS10 [37], KLP05 [63], and the proposed K26-svpv-s protocol. The K26-svpv-s protocol achieves efficiency gains ranging from 27.1% (BLS24-509) to 41.0% (BLS48-575) compared to local pairing computation, while the other protocols in this category significantly exceed the cost of local computation.

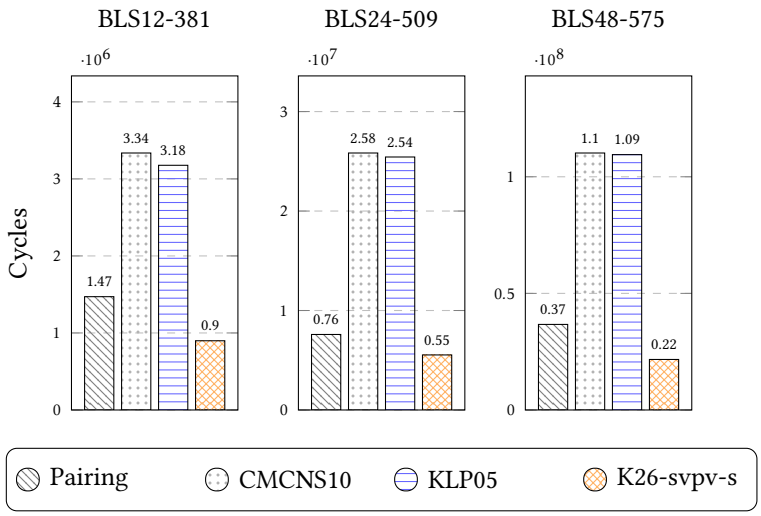


**Figure 5.5:** Single pairing delegation cost comparison for type SVPV.

**Batch Pairing Delegation** Figure 5.6 compares the average client cost per pairing for batch delegation protocols CKC23 [41] and the proposed K26-svpv-b protocol across batch sizes $\mathsf{M} \in \{2, 3, 10\}$. The K26-svpv-b protocol consistently achieves substantial efficiency gains, ranging from 36.4% ($\mathsf{M} = 2$, BLS24-509) to 64.8% ($\mathsf{M} = 10$, BLS48-575) compared to local computation.

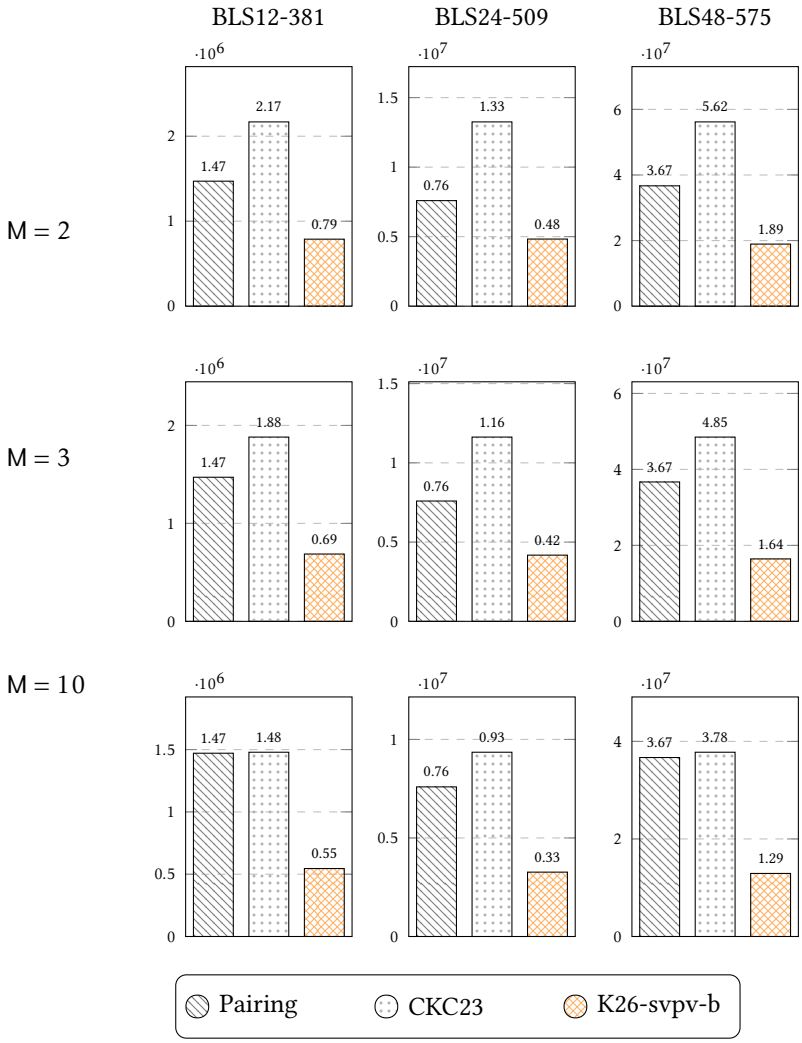


**Figure 5.6:** Batch pairing delegation cost comparison for type SVPV.

## 5.4 Type-PVSV Protocols ($A$ Public, $B$ Secret)

Type-PVSV protocols mask only the second pairing input, representing the complementary configuration to SVPV.

**Single Pairing Delegation** Figure 5.7 presents the average client cost per pairing for the proposed K26-pvsv-s protocol. This protocol achieves efficiency gains ranging from 15.5% (BLS24-509) to 36.0% (BLS12-381) compared to local pairing computation.

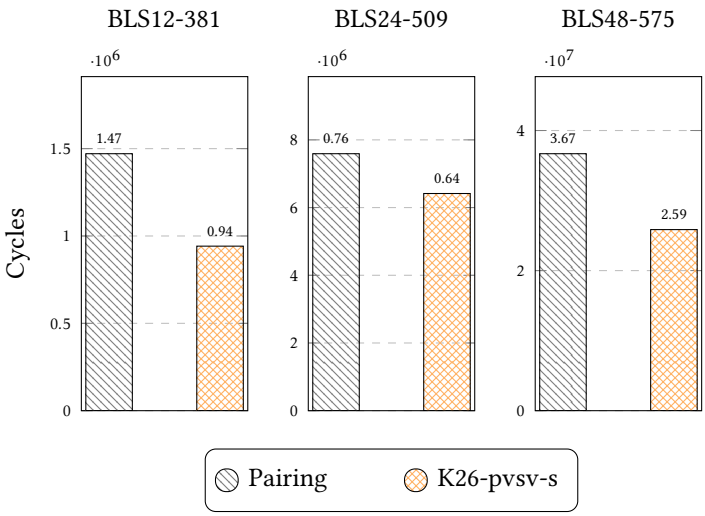


**Figure 5.7:** Single pairing delegation cost comparison for type PVSV.

**Batch Pairing Delegation** Figure 5.8 presents the average client cost per pairing for the proposed K26-pvsv-b protocol across batch sizes $\mathsf{M} \in \{2, 3, 10\}$. The protocol achieves efficiency gains ranging from 22.3% ($\mathsf{M} = 2$, BLS24-509) to 54.7% ($\mathsf{M} = 10$, BLS12-381) compared to local computation.

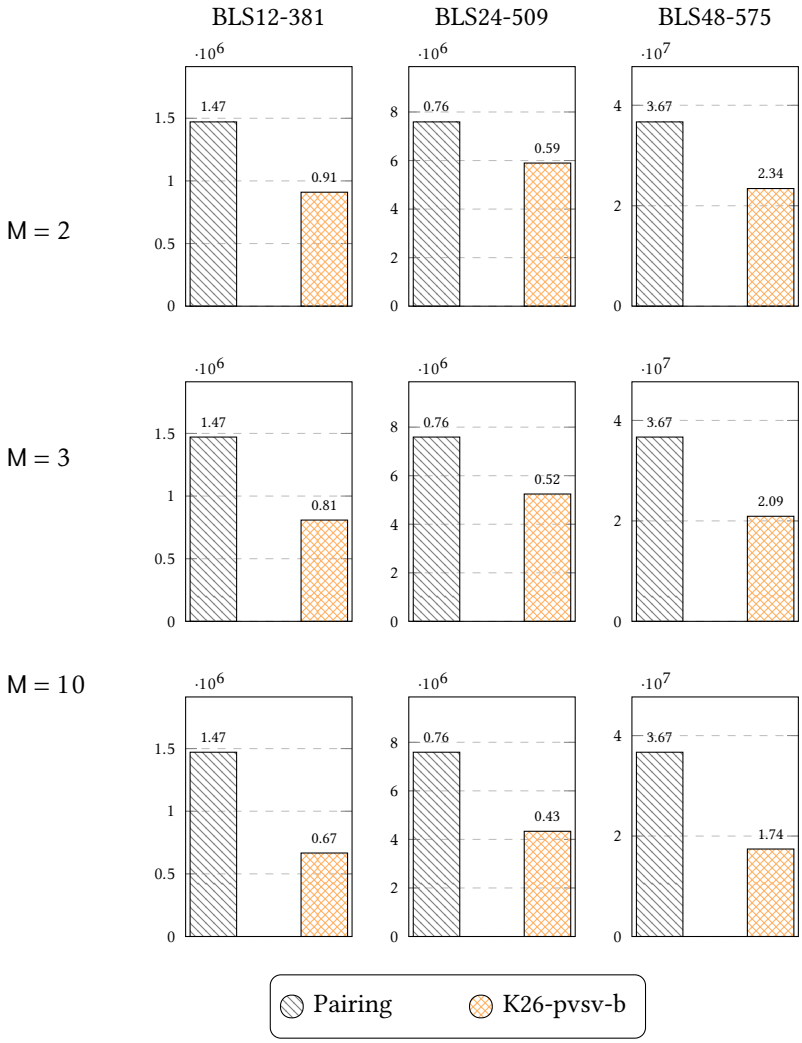


**Figure 5.8:** Batch pairing delegation cost comparison for type PVSV.

## 5.5 Comparison between KAPR25 and K26 Single Protocols

The following comparison is based on actual benchmark measurements from the protocol implementations [1]. Unlike the previous sections where costs were derived symbolically, these results reflect direct execution time measurements.

For the privacy cases of K26, only the online phases are accounted for since the offline phase OneTimeSetup is designed to be run once for delegating an unlimited sequence of pairing and thus can be fully amortized. On the other hand, for the public construction KAPR25, due to the timeout parameter $\tau$, the timings are accounted for 25 delegation rounds, so OneTimeSetup is still included in the measurements.

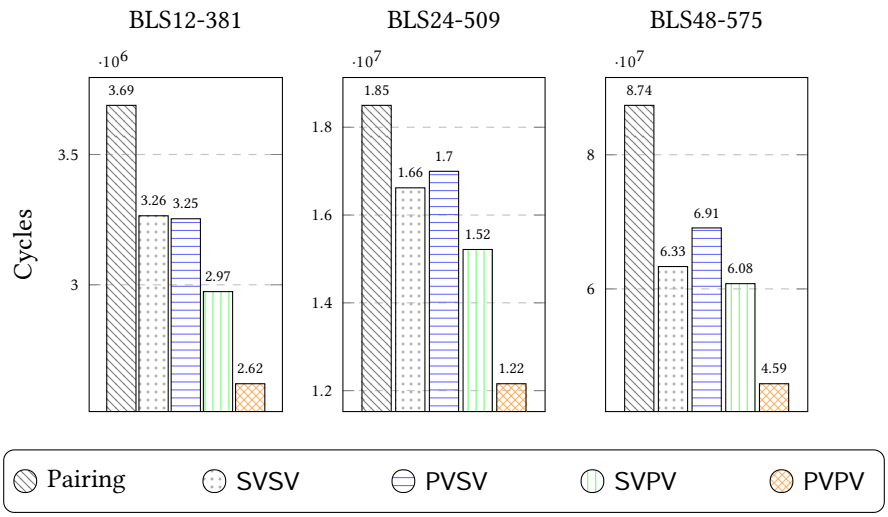


**Figure 5.9:** Cost comparison between the privacy delegation types SVSV, PVSV and SVPV of K26-single and the public case PVPV of KAPR25.

[1] `github.com/adrianperezkeilty/relic-privamore/tree/privamore-fix`

## 5.6 Comparison between KAPR25 and K26 for Batch Protocols

Notably, the K26-SVPV-b protocol outperforms KAPR25-PVPV on BLS12-381 for batch sizes $\mathsf{M} \in \{3, 10, 100\}$, demonstrating that input privacy can be achieved without additional cost in these particular configurations.

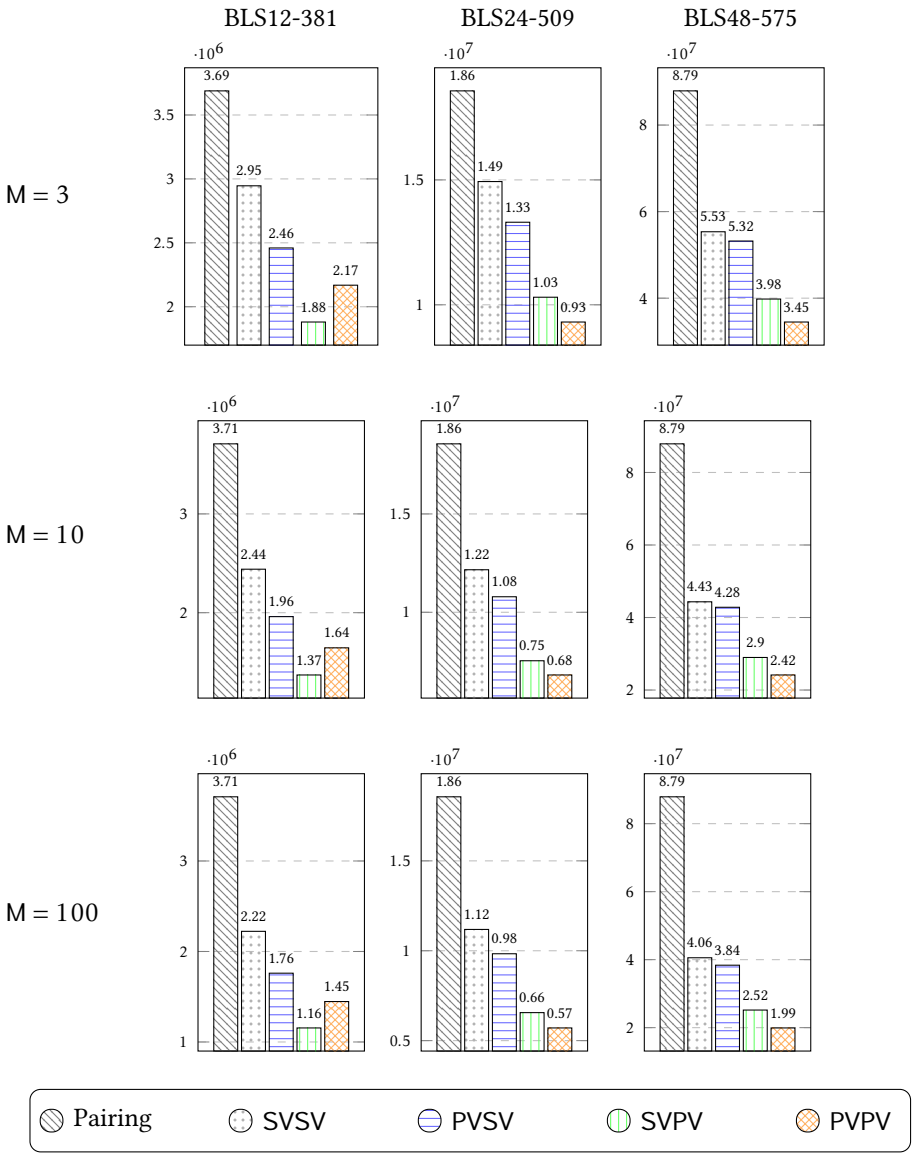


**Figure 5.10:** Cost comparison between the privacy delegation types SVSV, PVSV and SVPV of K26-batch and the public case PVPV of KAPR25.

# 6

# Future Directions

This chapter explores open research directions for pairing delegation protocols. Two main challenges are discussed: (1) achieving public verifiability, where commitment-based approaches using transparency logs are shown vulnerable to client-server collusion attacks due to an algebraic degree of freedom in verification equations, and (2) delegating pairing products, where many protocols (e.g., signature verification, zk-SNARKs) only require the product of multiple pairings rather than individual values, potentially enabling more efficient delegation strategies, though adapting KAPR25-style verification to protocols with identity products ($\rho = 1$) risks revealing longterm secrets.

The pairing delegation protocols presented in this thesis—particularly the KAPR25 framework and its private-input variants—achieve significant efficiency improvements and represent the first practical solutions for delegating pairings. Nevertheless, several important research directions remain open. This section explores the challenging problem of achieving public verifiability in the presence of client-server collusion, and the delegation of pairing products to speed up specific pairing based protocols (e.g., [5]) even more.

## 6.1 Public Verifiability of Delegated Pairings

In the KAPR25 protocol, verifiability is *verifier-designated*: only the client possessing the secret exponent $r$ can verify correctness using $\xi = \rho^r \cdot \gamma$. In blockchain and decentralized systems—particularly for zk-SNARK verification in resource-constrained nodes or verifiable computation auditing—*public verifiability* is desirable, where any third party can independently confirm that a delegated pairing was computed correctly.

**A Commitment-Based Approach** A natural idea is to use cryptographic commitments and transparency logs. The client commits to the delegation parameters $(\mathsf{pub}, r, \xi)$ *before* interacting with the server, publishing the commitment to an immutable, timestamped transparency log. After successful private verification, the client reveals these values. Third parties can then verify: (1) the commitment opens correctly, (2) the server's response came after the commitment (temporal ordering), and (3) the verification equation $\xi = \rho^r \cdot \gamma$ holds. However, this approach has a *fundamental flaw*.

**A Collusion Attack** If the client and server collude—specifically, if the client shares $r$ with the server before the delegation—they can forge arbitrary pairing values that pass third-party verification:

1. The client commits to $(\mathsf{pub}, r, \xi)$ in the transparency log.

2. The client secretly shares $r$ with the server.

3. For any $\eta \in \mathbb{G}_T$ with $\eta \neq 1$, the server returns:

$$\rho^* = \rho \cdot \eta$$
$$\gamma^* = \gamma \cdot \eta^{-r}$$

   which passes verification since

$$(\rho^*)^r \cdot \gamma^* = (\rho \cdot \eta)^r \cdot \gamma \cdot \eta^{-r} = \rho^r \cdot \gamma = \xi\,.$$

It was already highlighted in Lemma 3.3.1 that the verification equation has an algebraic degree of freedom: for any $\eta \in \mathbb{G}_T$, the pair $(\rho \cdot \eta, \gamma \cdot \eta^{-r})$ satisfies the equation. The commitment and temporal ordering only prove the client committed before receiving the response but an additional tool is needed for the client to convince external verifiers that its interaction with the server was carried out honestly.

**Economic Approaches via Auditing and Staking** A way to circumvent this issue could be to force the client to have a financial stake associated to the honesty of the delegation, but still relying on third-party auditors to detect fraud. A possible approach could be as follows:

- The client locks a financial stake before each delegation, committing to honest behavior.

- A set of *auditor nodes*—resource-rich entities capable of computing pairings locally—randomly sample published delegations for verification.

- Auditors recompute $e(A, B)$ for the revealed inputs and compare against the published $\rho$. If $\rho \neq e(A, B)$, fraud is detected.

- Upon detecting fraud, the auditor proves the discrepancy (e.g., by publishing both $\rho$ and $e(A, B)$ along with a validity proof), triggering stake slashing.

- The slashed stake is split: part goes to the auditor as a reward, and part is potentially burned to prevent collusion between clients and auditors.

Unfortunately this approach does not achieve the fully trustless public verifiability desired for decentralized applications.

## 6.2 Product Pairing Delegation

Many pairing-based protocols require the computation of the product of multiple pairings without needing their individual outputs (e.g., [24, 54, 76]). A notable example is the verification phase of the multi-key linearly homomorphic signature scheme by Aranha and Pagnin [5], where the verifier must run the verification check

$$e(\gamma, g_2) \stackrel{?}{=} \prod_{j=1}^{t} e\left( g_1^{\mu_j} \cdot \prod_{i \in I_j} H(\ell_i)^{f_i}, \mathrm{pk}_{id_j} \right). \tag{6.1}$$

The right-hand side requires computing $t$ pairings and multiplying them in $\mathbb{G}_T$. The verifier only needs this final product, not the individual pairing values (symbol descriptions are spared for the sake of brevity).

**Simplified Verification for Products** If a client intends to delegate the product $\prod_{i=1}^{n} e(A_i, B_i)$, the client does not need to verify each of the $n$ individual pairings separately. Instead, the client can focus on verifying the collective product as a whole. It becomes inconsequential if the server manipulates individual pairings or shuffles their values, as long as the final product is computed accurately.

**Security Issue with Identity Products** A critical limitation arises when adapting the KAPR25 verification approach to protocols where the expected pairing product equals the identity element $1 \in \mathbb{G}_T$ (a common case in many verification equations). Recall that in KAPR25, the client verifies using $\xi = \rho^r \cdot \gamma$, where $\xi$ is the longterm secret. If the expected result is $\rho = 1$, this reduces to:

$$\xi = 1^r \cdot \gamma = \gamma \tag{6.2}$$

Thus, the server learns the longterm secret $\xi$ which obliges the protocol to terminate. Protocols requiring pairing products that equal 1 therefore need fundamentally different verification strategies that do not expose longterm secrets through trivial algebraic simplification.

# A

# Appendix

## A.1 Estimating Concrete Costs from Symbolic Ones

Given the specification of a pairing delegation protocol, one can express symbolically the overall client cost by identifying the bilinear group operations involved and later obtain their corresponding cost in clock cycles by running the respective benchmarks in RELIC. For obtaining the concrete cost on, say, BLS12-381, cd-into the RELIC repository and run the following commands to build the benchmarking executable and run it:

```
$ ./relic/preset/x64-pbc-bls12-381.sh relic-target
$ cd /relic/relic-target && make -j$(nproc)
$ ./relic/relic-target/bin/bench_pc
```

This will output among other benchmarks the following relevant ones:

```
-- Curve B12-P381:

-- Group G_1:

BENCH: g1_add                  = 1198 cycles
BENCH: g1_mul                  = 155767 cycles
BENCH: g1_rand                 = 73058 cycles

-- Group G_2:

BENCH: g2_add                  = 3031 cycles
BENCH: g2_mul                  = 242298 cycles
```

```
BENCH: g2_rand                 = 179805 cycles

-- Group G_T:

BENCH: gt_is_valid             = 120322 cycles
BENCH: gt_mul                  = 3778 cycles
BENCH: gt_exp                  = 400800 cycles

-- Pairing:

BENCH: pc_map                  = 1470968 cycles
```

where the costs of the group operations can be read off as:

| Operation | Cost (cycles) |
|---|---|
| $\mathfrak{m}_1$ (g1_mul) | 155,767 |
| $\mathfrak{m}_2$ (g2_mul) | 242,298 |
| $\mathfrak{m}_1^{\$}$ (g1_rand) | 73,058 |
| $\mathfrak{m}_2^{\$}$ (g2_rand) | 179,805 |
| $\mathfrak{m}_T$ (gt_exp) | 400,800 |
| $\in_T$ (gt_is_valid) | 120,322 |
| $\mathfrak{g}_1$ (g1_add) | 1,198 |
| $\mathfrak{g}_2$ (g2_add) | 3,031 |
| $\mathfrak{g}_T$ (gt_mul) | 3,778 |

**Table A.1:** Correspondence between symbolic operations and benchmark costs on BLS12-381.

### A.1.1 Methodology

The concrete computational costs presented in the histograms throughout this thesis are derived from the symbolic expressions in Table 2.3 by mapping each symbolic operation to its corresponding benchmark measurement from the RELIC library [4]. The elliptic curves BLS12-381, BLS24-509 and BLS48-575 were selected, each yielding different pairing groups with an increasing security level. RELIC was chosen for benchmarking since it includes all the aforementioned parameter sets with similar levels of optimization, including

assembly acceleration for Intel 64-bit platforms and the best-known formulas for arithmetic in pairing groups, allowing for a comprehensive and fair comparison.

Short scalar multiplications and exponentiations (denoted $\mathfrak{m}_1^{\mathsf{sh}}$, $\mathfrak{m}_2^{\mathsf{sh}}$, and $\mathfrak{m}_T^{\mathsf{sh}}$) use 40-bit scalars and have therefore correspondingly lower costs than their full-length counterparts. The concrete costs were obtained by multiplying the locally obtained full-length costs ($\mathfrak{m}_1$, $\mathfrak{m}_2$, and $\mathfrak{m}_T$ from Table A.1) by the ratios $\mathfrak{m}_i^{\mathsf{sh}}/\mathfrak{m}_i$ reported in [66, Table 2]. For example, for BLS12-381, the cost of $\mathfrak{m}_1^{\mathsf{sh}}$ is estimated as:

$$\mathfrak{m}_1^{\mathsf{sh}} = \mathfrak{m}_1 \cdot \frac{\mathfrak{m}_1^{\mathsf{sh}}}{\mathfrak{m}_1} = 155767 \cdot \frac{161}{373} \approx 67,235 \text{ cycles}$$

For single pairing delegation protocols, the total cost is computed by evaluating the symbolic expression with $\mathsf{N} = 25$, yielding the cost for 25 sequential delegations. This total is then divided by 25 to obtain the *average cost per pairing*. For batch delegation protocols, the symbolic expression is evaluated for batch sizes $\mathsf{M} \in \{2, 3, 10\}$ with $\mathsf{N} = 25$, and the total cost is divided by $\mathsf{N} \cdot \mathsf{M}$ to obtain the average cost per pairing. This normalization enables direct comparison with the baseline cost of local pairing computation (`pc_map`).

**Example:** Consider the K26-svsv-b protocol for batch delegation with $\mathsf{M} = 10$ on BLS12-381. The symbolic expression is:

$$\mathfrak{m}_T + \mathsf{N}\left(\mathfrak{m}_1 + \mathfrak{m}_1^{\$} + 2\mathfrak{m}_2 + \mathsf{M}\left(2\mathfrak{m}_1 + \frac{1}{2}\mathfrak{m}_2 + \frac{1}{2}\mathfrak{m}_T + \in_T + \mathfrak{g}_1 + \mathfrak{g}_2 + \mathfrak{g}_T\right)\right)$$

Substituting the RELIC benchmark values and $\mathsf{N} = 25$, $\mathsf{M} = 10$:

$$\begin{aligned}\text{Cost} &= 400{,}800 + 25 \times (155{,}767 + 73{,}058 + 2 \times 242{,}298 \\ &\quad + 10 \times (2 \times 155{,}767 + 0.5 \times 242{,}298 + 0.5 \times 400{,}800 \\ &\quad + 120{,}322 + 1{,}198 + 3{,}031 + 3{,}778)) \\ &= 208{,}589{,}325 \text{ cycles}\end{aligned}$$

Dividing by $\mathsf{N} \times \mathsf{M} = 250$ yields an average of 834,357 cycles per pairing, representing a 43.3% efficiency gain over the baseline pairing cost of 1,470,968 cycles.

### A.1.2 Histogram Generation Scripts

The performance histograms were generated from the symbolic cost formulas in Table 2.3 and matching them against the RELIC benchmarks as earlier described.

**Cost Computation:** `https://github.com/adrianperezkeilty/lic/blob/main/performance/new-histograms/compute_costs.py`

**Histogram Generation:** `https://github.com/adrianperezkeilty/lic/blob/main/performance/new-histograms/generate_histograms.py`

## A.2 Running Benchmarks for KAPR25 and K26

This section describes how to run benchmarks for the KAPR25 and K26 protocols. The benchmarks measure the client-side computational cost of the delegation protocols.

**Preparing the Benchmarking Environment.** To obtain reliable and reproducible benchmark results, the system should be configured to minimize variance from CPU frequency scaling, Turbo Boost, and hyper-threading. The following script applies these settings:

```
$ chmod +x bench-mode.sh undo-bench-mode.sh
$ sudo ./bench-mode.sh
```

This script performs the following:

- Disables Intel Turbo Boost to prevent frequency fluctuations
- Offlines hyper-threading sibling cores to eliminate resource contention
- Sets the Energy Performance Preference (EPP) to `performance`

**Running the Benchmarks.** After applying the benchmarking settings, run the protocol benchmarks pinned to a specific CPU core:

```
$ taskset -c 2 ./relic/relic-target/bin/bench_cp
```

The `taskset -c 2` command pins the benchmark process to CPU core 2, which should be a physical core with its hyper-threading sibling offlined.

**Configuring Batch Size.** For batch delegation protocols, the batch size M is controlled by the `AGGS` variable in the source code. Before compiling the benchmarks, update this variable to the desired batch size:

- `AGGS = 3` for $M = 3$
- `AGGS = 10` for $M = 10$
- `AGGS = 100` for $M = 100$

After modifying the variable, recompile the benchmark executable with `make -j$(nproc)` before running.

**Restoring Normal Settings.** After benchmarking is complete, restore the system to normal operation:

```
$ sudo ./undo-bench-mode.sh
```

This re-enables Turbo Boost, brings hyper-threading cores back online, and restores the original EPP values.

## A.3 Parametrization Verification Scripts

The symbolic parametrizations for the private delegation protocols (SVSV, SVPV, and PVSV) presented in this thesis have been verified using Python scripts with the SymPy library. These scripts validate that the proposed parametrizations satisfy the required algebraic constraints for both single and batch delegation modes.

All verification scripts are available in the GitHub repository:

### A.3.1 Single Pairing Delegation

- PVSV (**$A$ public, $B$ secret):**
  `https://github.com/adrianperezkeilty/lic/blob/main/appendix/verify_pvsv_parametrization.py`
- SVPV (**$A$ secret, $B$ public):**
  `https://github.com/adrianperezkeilty/lic/blob/main/appendix/verify_svpv_parametrization.py`

- SVSV (**$A$ secret, $B$ secret**):
  `https://github.com/adrianperezkeilty/lic/blob/main/appendix/verify_svsv_parametrization.py`

### A.3.2 Batch Pairing Delegation

- PVSV (**$A$ public, $B$ secret**):
  `https://github.com/adrianperezkeilty/lic/blob/main/appendix/verify_pvsv_batch_parametrization.py`
- SVPV (**$A$ secret, $B$ public**):
  `https://github.com/adrianperezkeilty/lic/blob/main/appendix/verify_svpv_batch_parametrization.py`
- SVSV (**$A$ secret, $B$ secret**):
  `https://github.com/adrianperezkeilty/lic/blob/main/appendix/verify_svsv_batch_parametrization.py`