This is the preprint (full) version of an article with the same title that will appear at ACM CCS 2017 on Oct. 30–Nov. 3, 2017, Dallas, TX, USA.
DOI: 10.1145/3133956.3134068

New Techniques for Structural Batch Verification in Bilinear Groups with Applications to Groth–Sahai Proofs

Gottfried Herold*
Ecole Normale Supérieure
LIP/AriC
Lyon, France
gottfried.herold@ens-lyon.fr

Max Hoffmann†
Ruhr-Universität Bochum
Horst Görtz Institute for IT-Security
Bochum, Germany
max.hoffmann@rub.de

Michael Klooß‡
Karlsruhe Institute of Technology
Department of Informatics
Karlsruhe, Germany
michael.klooss@kit.edu

Carla Ràfols§
Universitat Pompeu Fabra
DTIC
Barcelona, Spain
carla.rafols@upf.edu

Andy Rupp¶
Karlsruhe Institute of Technology
Department of Informatics
Karlsruhe, Germany
andy.rupp@kit.edu

ABSTRACT

Bilinear groups form the algebraic setting for a multitude of important cryptographic protocols including anonymous credentials, e-cash, e-voting, e-coupon, and loyalty systems. It is typical of such crypto protocols that participating parties need to repeatedly verify that certain equations over bilinear groups are satisfied, e.g., to check that computed signatures are valid, commitments can be opened, or non-interactive zero-knowledge proofs verify correctly. Depending on the form and number of equations this part can quickly become a performance bottleneck due to the costly evaluation of the bilinear map.

To ease this burden on the verifier, batch verification techniques have been proposed that allow to combine and check multiple equations probabilistically using less operations than checking each equation individually. In this work, we revisit the batch verification problem and existing standard techniques. We introduce a new technique which, in contrast to previous work, enables us to fully exploit the structure of certain systems of equations. Equations of the appropriate form naturally appear in many protocols, e.g., due to the use of Groth–Sahai proofs.

The beauty of our technique is that the underlying idea is pretty simple: we observe that many systems of equations can alternatively be viewed as a single equation of products of polynomials for which probabilistic polynomial identity testing following Schwartz–Zippel can be applied. Comparisons show that our approach can lead to significant improvements in terms of the number of pairing evaluations. Indeed, for the BeleniosRF voting system presented at CCS 2016, we can reduce the number of pairings (required for ballot verification) from $4k + 140$, as originally reported by Chaidos et al. [19], to $k + 7$. As our implementation and benchmarks demonstrate, this may reduce the verification runtime to only 5% to 13% of the original runtime.

CCS CONCEPTS

• **Security and privacy → Mathematical foundations of cryptography**; *Public key (asymmetric) techniques*; • **Theory of computation → Cryptographic protocols**;

KEYWORDS

Batch verification; bilinear maps; Groth-Sahai proofs; structure-preserving cryptography; Belenios; P-signatures.

1 INTRODUCTION

Elliptic curve groups equipped with a bilinear map, aka *pairing*, proved to be one of the most useful mathematical objects of modern cryptography. Since the first constructive use of pairings by Joux [38] countless pairing-based cryptographic schemes and protocols have been proposed. Identity-based encryption [13], identity-based signatures [46], group [31], ring [20], and aggregate signatures [14], anonymous credentials [18], e-cash [16], e-voting [21, 22], e-coupons [45], and loyalty programs [37] are just a few examples.

It is common, especially in more complex pairing-based systems, e.g., like e-cash and e-voting schemes, that parties need to repeatedly verify multiple equations involving the pairing. Since in terms of execution time, a pairing operation is usually much more expensive than an exponentiation in the elliptic curve groups (e.g., 5.52 ms vs. 1.13 ms for our smartphone implementation of a 254-bit order bilinear setting), those verifications can quickly become a performance bottleneck. In particular, this might be the case when targeting constrained devices or when huge numbers of equations need to be checked at a server.

The equations we consider typically come in the form of verification equations for digital signature schemes, commitment schemes, but first and foremost in the form of verification equations of zero-knowledge or witness-indistinguishable proofs. These proofs are a very common tool in cryptographic protocols to enforce that participating parties behave honestly. In pairing-based crypto the

*The author is supported by ERC Starting Grant 335086 Lattices: Algorithms and Cryptography (LattAC)
†The author is supported by DFG grant PA 587/10-1.
‡The author is supported by the Competence Center for Applied Security Technology (KASTEL).
§The author is supported by Marie Curie COFUND project “UPF Fellows” under grant agreement 600387.
¶The author is supported by DFG grant RU 1664/3-1 and the Competence Center for Applied Security Technology (KASTEL).

most famous instance of such a proof system which is employed in numerous schemes and protocols, e.g., [5, 7, 9, 15, 19, 21, 22, 30, 37] to cite only a few, is due to Groth and Sahai [35]. Groth–Sahai (GS) proofs are non-interactive witness-indistinguishable (NIWI) and zero-knowledge (NIZK) proofs of satisfiability of quadratic equations in bilinear groups. In many cases, these proofs are also extractable and can be used as a proof of knowledge. While under certain circumstances, there might be more efficient alternatives like Fiat-Shamir-based NIZKs [27] or SNARKs [10], there are many good reasons to stick with GS. For instance, GS proofs are secure in the CRS model under standard assumptions, whereas the security of FS-NIZKs is based on the highly idealized and controversial Random-Oracle-Model (ROM) and current instantiations of SNARKs make use of non-falsifiable assumptions. Also, GS proofs are rerandomizable, a very useful property exploited in privacy-preserving protocols like e-voting schemes [21, 22], which cannot be easily obtained in the ROM.

Related work. In order to speed up the verification of large systems of equations, several batch verification techniques have been proposed.

Bellare et al. [8] proposed batch verifiers for modular exponentiations, i.e., equations of the form $a_i = g^{x_i}$. The small exponent batching technique combines n such equations into a single one, i.e., $\prod_i a_i^{r_i} = g^{x_i r_i}$ which is a linear combination with random ℓ-bit coefficients r_i, where ℓ is a security parameter (typically $\ell = 80$). This approach comes at the cost of a small loss of soundness: even if the combined equation verifies, there is a probability of at most $2^{-\ell}$ that one of the original equations was false. Camenisch et al. [17] apply the small exponent technique to batch verify two short pairing-based signatures in scenarios with many different signers. Besides presenting batch verifiers for a variety of signature schemes over bilinear groups, Ferrara et al. [26] also give a general (informal) framework of how to securely and efficiently batch verify a set of pairing-based equations. They propose the use of small exponent batching and describe a set of rules to optimize the batched equations. These rules take the relative costs of pairing and exponentiations in the different groups of a pairing-based setting into account but might be conflicting. Thus, [11] looks at different types of equations arising from Groth–Sahai proofs to find an optimally batched equation for each case. As finding optimal expressions using the considered set of rules by hand quickly becomes tedious and error-prone, the authors of [2, 3] propose automated batch verification over bilinear groups.

If the higher-level structure which our batching exploits is *known* and made explicit for the algorithm, it should be easily integrated into (existing) automated batching tools. This is the case for GS proofs. We did not explore how to algorithmically *find* such higher-level structure, but believe it to be very expensive in general.

Our contributions. Many of the equations in bilinear groups one wants to batch verify are very structured (e.g. GS proofs [35], structure preserving cryptography [1, 41] or the protocols [33, 42]). For instance, many of them consist simply of matrix-vector multiplication in the exponent. In contrast, the examples considered by Ferrara et al. lack such structure.

Our work exploits this additional structure to get significant performance improvements. In particular, we note that verification of GS proofs and linear algebra operations can be written as checking a set of polynomial identities in the exponent.[1] Our batching algorithm simply performs the usual probabilistic identity checking algorithm in the exponent (evaluation at some point chosen from a large enough set S). Pointwise evaluation of the polynomial identity costs several exponentiations, but reduces the number of pairing operations significantly compared to other batching strategies. A nice feature of the polynomial view on batching is that it suggests other possibilities for S apart from small exponents, as was considered in [23] for batching of exponentiations. The fundamental insight which explains our efficiency gains is that pointwise evaluation corresponds to computing a random linear combination of all the individual equations with structured randomness (as opposed to independent). This structured randomness is compatible with the pairing and allows to compute most exponentiations in G_1 and G_2 *before* computing the pairing. One nice feature of our approach is that it is more systematic and less "hand-crafted".

We apply our techniques to two concrete examples, P-signatures [6] and the BeleniosRF voting scheme [19], and compare our savings with the small exponent batching approach. This demonstrates that in certain cases our technique can make a big difference: we are able to reduce the required number of pairing operations for ballot verification in BeleniosRF to $k + 7$, whereas Chaidos et al. report $4k + 140$ pairing applications in [19] using small exponent batching. Furthermore, our experimental results show the impact of saving pairings on performance.

Batching in bilinear groups: An illuminating example. In the following, we consider a pairing $e\colon G_1 \times G_2 \to G_T$ for cyclic groups G_1, G_2, G_T of order p. We use additive notation for all groups. Hence, bilinearity means that for all $\gamma_1, \gamma_2 \in \mathbb{Z}_p$ and $a_1 \in G_1, a_2 \in G_2$, we have $e(\gamma_1 a_1, \gamma_2 a_2) = \gamma_1\gamma_2 e(a_1, a_2)$. Furthermore, we use implicit notation for group elements, i.e., we write $[x]_\sigma \coloneqq x\mathcal{P}_\sigma \in G_\sigma$ with respect to some fixed group generator $\mathcal{P}_\sigma \in G_\sigma$, where $\sigma \in \{1, 2, T\}$. This notation naturally generalizes to matrices of group elements by applying it component-wise. Using this notation we can write $e([x]_1, [y]_2) = [xy]_T$ for $x, y \in \mathbb{Z}_p$.

Batch verification techniques typically consist of two phases: a first phase where all the equations are combined into a single one, and a second phase where the cost of verifying the combined claim is optimized. In the bilinear setting, the latter means reducing the number of pairing computations, which are the most expensive operations, but also trying to minimize the total cost taking into account the relative cost of other operations (exponentiations in G_T are the most expensive, followed by exponentiations in G_2, followed by exponentiations in G_1). For instance, for the second phase, one of the rules of Ferrara et al. [26] says "move the exponent into the pairing", i.e. one should replace terms of the form $\gamma e([x]_1, [y]_2)$ by $e(\gamma[x]_1, [y]_2)$ in the combined expression.

Following these lines, let us now illustrate the small exponent batching as well as our new technique by considering the following set of claims:

$$\begin{aligned} e([x_1]_1, [y_1]_2) &\overset{?}{=} [t_1]_T \qquad & e([x_1]_1, [y_2]_2) &\overset{?}{=} [t_2]_T \\ e([x_2]_1, [y_1]_2) &\overset{?}{=} [t_3]_T \qquad & e([x_2]_1, [y_2]_2) &\overset{?}{=} [t_4]_T \end{aligned} \tag{1}$$

[1] This is inspired by (but even simpler than) the polynomial framework of [36].

Applying small exponent batching, which is the standard approach [8, 11, 26], results in the following single combined claim in Phase 1:

$$\begin{array}{ll}\gamma_1 e([x_1]_1,[y_1]_2)+\gamma_2 e([x_1]_1,[y_2]_2)+ & \\ \gamma_3 e([x_2]_1,[y_1]_2)+\gamma_4 e([x_2]_1,[y_2]_2) & \overset{?}{=}\sum_{i=1}^4 \gamma_i[t_i]_T.\end{array}$$

for independent, random (small) γ_i. In Phase 2, one applies the rules from [26] to minimize the number of operations. This yields:

$$e(\gamma_1[x_1]_1+\gamma_3[x_2]_1,[y_1]_2)+e(\gamma_2[x_1]_1+\gamma_4[x_2]_1,[y_2]_2)\overset{?}{=}\sum_{i=1}^{4}\gamma_i[t_i]_T.$$

In particular, the total expression can be evaluated with only 2 pairings and 4 exponentiations in each of G_1 and G_T.

Our new approach works as follows: we introduce Phase 0 in which we try to rewrite Eq. (1) in terms of a higher-level bilinear map $\tilde{e}\colon G_1^2\times G_2^2\to G_T^4$ involving the underlying basic bilinear map $e\colon G_1\times G_2\to G_T$. In this case, we define

$$\tilde{e}([\vec{x}]_1,[\vec{y}]_2):=\begin{pmatrix}e(x_1,y_1) & e(x_1,y_2)\\ e(x_2,y_1) & e(x_2,y_2)\end{pmatrix}=\begin{pmatrix}[x_1y_1]_T & [x_1y_2]_T\\ [x_2y_1]_T & [x_2y_2]_T\end{pmatrix}$$
$$=[\vec{x}\vec{y}^\top]_T\,,$$

and write Eq. (1) we want to verify as

$$\tilde{e}([\vec{x}]_1,[\vec{y}]_2)\overset{?}{=}\begin{pmatrix}[t_1]_T & [t_2]_T\\ [t_3]_T & [t_4]_T\end{pmatrix}.$$

If we look at this equation in the exponent, the claim reads $\vec{x}\vec{y}^\top \overset{?}{=} \mathbf{T}$. We can, alternatively, interpret vectors and matrix as polynomials $\mathbf{f}=x_1R_1+x_2R_2$, $\mathbf{g}=x_1S_1+x_2S_2$ and $\mathbf{h}=t_1R_1S_1+t_2R_1S_2+t_3R_2S_1+t_4R_2S_2$ in variables R_1,R_2,S_1,S_2. So the equation in the exponent looks like $\mathbf{f}\cdot\mathbf{g}=\mathbf{h}$. Now, in Phase 1, we sample $\vec{r},\vec{s}\leftarrow\mathbb{Z}_p^2$ and check whether

$$\mathbf{f}(\vec{r})\cdot\mathbf{g}(\vec{s})\overset{?}{=}\mathbf{h}(\vec{r},\vec{s})\qquad\left(\iff(\vec{r}^\top\vec{x})(\vec{y}^\top\vec{s})\overset{?}{=}\vec{r}^\top\mathbf{T}\vec{s}\right),$$

holds, which is equivalent to verifying if

$$e([\vec{r}^\top\vec{x}]_1,[\vec{y}^\top\vec{s}]_2)\overset{?}{=}\vec{r}^\top\begin{pmatrix}[t_1]_T & [t_2]_T\\ [t_3]_T & [t_4]_T\end{pmatrix}\vec{s}$$

holds. To spell this out equation-wise, we are now checking if

$$e([r_1x_1+r_2x_2]_1,[s_1y_1+s_2y_2]_2)\overset{?}{=}$$
$$r_1s_1[t_1]_T+r_1s_2[t_2]_T+r_2s_1[t_3]_T+r_2s_2[t_4]_T,$$

which corresponds to a linear combination of each of the atomic claims (each equation in G_T) with coefficients of a special form, namely, $r_1s_1, r_1s_2, r_2s_1, r_2s_2$. Obviously, correctness holds because of basic linear algebra. It is not hard to see that the soundness error is at most $2/p$. The interesting observation is that we have reduced the number of pairing operations with respect to the small exponent batching approach by 50% (and by 75% compared to the naive approach).

Often, e.g. when verifying GS proofs, one has a set $\mathcal{Z}$ of sets $\mathcal{EQ}_i$ of equations with the property that each set $\mathcal{EQ}_i$ is amenable to our techniques. As a first step, we batch each set $\mathcal{EQ}_i$ into a single equation $\mathcal{E}_i$. We call this *internal* batching. Then we batch the equations $\mathcal{E}_i$ (for $i\in\mathcal{Z}$) into a single equation $\mathcal{E}$. This is called *external* batching.

For very structured equations, even external batching is amenable to our techniques, which yields extremely efficient batch verification. See App. C.3 for an example.

2 PRELIMINARIES

2.1 Bilinear Groups

The results of this paper are in the setting of (prime-order) bilinear groups.[2] We use the following formal definition.

Definition 2.1 (prime-order bilinear group generator). A *prime-order bilinear group generator* is a PPT algorithm Gen that on input of a security parameter 1^λ outputs a tuple of the form

$$\mathcal{BG}:=(G_1,G_2,G_T,e,p,\mathcal{P}_1,\mathcal{P}_2)\leftarrow\mathsf{Gen}(1^\lambda)$$

where G_1,G_2,G_T are descriptions of cyclic groups of prime order p, $\log p=\Theta(\lambda)$, $\mathcal{P}_1$ is a generator of G_1, $\mathcal{P}_2$ is a generator of G_2, and $e\colon G_1\times G_2\to G_T$ is a map which satisfies the following properties:

- **Bilinearity:** For all $Q_1,Q_1'\in G_1$, $Q_2,Q_2'\in G_2$, $\alpha\in\mathbb{Z}_p$, we have $e(\alpha Q_1,Q_2)=e(Q_1,\alpha Q_2)=\alpha e(Q_1,Q_2)$, as well as $e(Q_1+Q_1',Q_2)=e(Q_1,Q_2)+e(Q_1',Q_2)$ and likewise $e(Q_1,Q_2+Q_2')=e(Q_1,Q_2)+e(Q_1,Q_2')$.
- **Non-Degeneracy:** $\mathcal{P}_T:=e(\mathcal{P}_1,\mathcal{P}_2)$ generates G_T.

Following [36], we refer to e as a *basic* bilinear map. This is to distinguish it from a *composite* bilinear map $\tilde{e}\colon G_1^{n_1}\times G_2^{n_2}\to G_T^{n_T}$, for some $n_1,n_2,n_T\in\mathbb{N}$, which is built on top of e and whose evaluation requires several evaluations of the basic bilinear map.

Unless it is specifically stated, $\mathcal{BG}$ refers to a generic bilinear map and no assumption is made concerning the (non)-existence of an efficiently computable isomorphism between G_1 and G_2. Only in certain contexts we will specify if $\mathcal{BG}$ is an elliptic curve of Type I, II or III (according to the classification of [32]). In case of Type I (or symmetric) bilinear groups, $G_1=G_2$ and we drop the subindex to refer to group elements, i.e. $e\colon G\times G\to G_T$.

2.2 Notation: Implicit Representation

To represent elements in groups we use the notation introduced in [25]. Namely, given a cyclic group $\langle\mathcal{P}\rangle=G$ of order p, for $a\in\mathbb{Z}_p$ we define $[a]:=a\mathcal{P}$. Similarly, given a bilinear group $\mathcal{BG}:=(G_1,G_2,G_T,e,p,\mathcal{P}_1,\mathcal{P}_2)$, we generalize this notation to distinguish between the elements of the groups G_1,G_2,G_T as $[a]_\sigma:=a\mathcal{P}_\sigma$, for any $\sigma\in\{1,2,T\}$. Although all groups are written in additive notation, we still use the term exponentiation to refer to scalar multiplications $a\mathcal{P}_\sigma$ for $\mathcal{P}_\sigma\in G_\sigma$ and $a\in\mathbb{Z}_p$. We further extend our notation to refer to vectors and matrices of group elements, namely, for a vector $\vec{f}\in\mathbb{Z}_p^n$ and a matrix $\mathbf{A}=(a_{i,j})_{i,j}\in\mathbb{Z}_p^{n\times m}$, the implicit representation in the group G_σ is defined by $[\vec{f}]_\sigma:=([f_i]_\sigma)_i\in G_\sigma^n$ and $[\mathbf{A}]_\sigma:=([a_{i,j}]_\sigma)_{i,j}\in G_\sigma^{n\times m}$ for $\sigma\in\{1,2,T\}$.

2.3 Pairing-based Claims and Batch Verifiers

We define batch verifiers for claims over bilinear groups following [26]. Let $\mathcal{BG}:=(G_1,G_2,G_T,e,p,\mathcal{P}_1,\mathcal{P}_2)\leftarrow\mathsf{Gen}(1^\lambda)$. A *pairing-based verification equation* over $\mathcal{BG}$ is a boolean relation of the form

$$\sum_{i=1}^{m}\alpha_i\, e([a_i]_1,[b_i]_2)\overset{?}{=}[c]_T$$

[2] Using a simple generalization of the lemma of Schwartz–Zippel enables our techniques in composite order $N=p_1^{e_1}\cdot\ldots\cdot p_n^{e_n}$, as long as $\min_i(p_i)$ is large enough.

for $m \in \mathrm{poly}(\lambda)$, $[a_i]_1 \in G_1$, $[b_i]_2 \in G_2$, $[c]_T \in G_T$, and $\alpha_i \in \mathbb{Z}_p$. Let us denote this relation by C and call it a *claim.* A pairing-based *batch verifier* with soundness error (i.e. false positive error probability) ε is a PPT algorithm which on input of a representation of claims $C^{(1)}, \ldots, C^{(n)}$, where $n \in \mathrm{poly}(\lambda)$, always accepts if all claims are true, and rejects otherwise with probability $1 - \varepsilon$. The soundness error ε is usually set to $2^{-\ell}$, where ℓ is a statistical security parameter and a standard choice is $\ell = 80$.

2.4 Computational Assumptions

Our improvements in batch verification are unconditional and depend only indirectly on the hardness of some cryptographic problem.[3] However, we will be referring to different computational assumptions in bilinear groups when describing the Groth–Sahai proof system and list them here for completeness.

The decisional Diffie-Hellman and Decisional Linear Assumptions are two of the most standard assumptions in the bilinear group setting and are described below.

- *Decisional Diffie-Hellman (*DDH*) Assumption.* It is hard to distinguish $(\mathcal{G}, [x], [y], [xy])$ from $(\mathcal{G}, [x], [y], [z])$, for $\mathcal{G} = (G, p, \mathcal{P}) \leftarrow \mathsf{Gen}(1^\lambda)$, $x, y, z \leftarrow \mathbb{Z}_p$.
- *Decisional* 2*-Linear (*2-Lin*) Assumption* [12]. It is hard to distinguish $(\mathcal{G}, [x_1], [x_2], [r_1x_1], [r_2x_2], [r_1 + r_2])$ from $(\mathcal{G}, [x_1], [x_2], [r_1x_1], [r_2x_2], [z])$, for $\mathcal{G} = (G, p, \mathcal{P}) \leftarrow \mathsf{Gen}(1^\lambda)$ and uniform $x_1, x_2, r_1, r_2, z \leftarrow \mathbb{Z}_p$.

In a bilinear group $\mathcal{BG}$, the SXDH *Assumption* says that DDH holds in G_1 and G_2.

2.5 Schwartz–Zippel Lemma

The Schwartz–Zippel Lemma is a well-known result [47, 50] which can be used for efficient testing of polynomial identities.

Lemma 2.2. *Let* $\mathbf{f} \in \mathbb{Z}_p[\vec{X}]$, $\vec{X} = (X_1, \ldots, X_\mu)$ *a non-zero polynomial of total degree d. Let* $S \subset \mathbb{Z}_p$ *be any finite set. If* $x_1, \ldots, x_\mu$ *are sampled independently and uniformly at random from S, then*

$$\Pr[\mathbf{f}(x_1, \ldots, x_\mu) = 0] \leq d/|S|.$$

2.6 Overview of Groth–Sahai Proofs

Groth and Sahai [35] constructed a proof system for satisfiability of quadratic equations over bilinear groups. The equations for which one can prove satisfiability can be written in a unified way as

$$\sum_{j=1}^{m_y} f(a_j, \mathsf{y}_j) + \sum_{i=1}^{m_x} f(\mathsf{x}_i, b_i) + \sum_{i=1}^{m_x}\sum_{j=1}^{m_y} f(\mathsf{x}_i, \gamma_{ij}\mathsf{y}_j) = t, \qquad (2)$$

where A_1, A_2, A_T are $\mathbb{Z}_p$-modules, $\vec{\mathsf{x}} \in A_1^{m_x}$, $\vec{\mathsf{y}} \in A_2^{m_y}$ are the *variables* (for which one shows that there is a satisfying assignment), $\vec{a} \in A_1^{m_y}$, $\vec{b} \in A_2^{m_x}$, $\Gamma = (\gamma_{ij}) \in \mathbb{Z}_p^{m_x \times m_y}$, $t \in A_T$ are the *constants* of the equation and $f\colon A_1 \times A_2 \to A_T$ is a bilinear map. More specifically, Groth and Sahai consider equations of the following types for $\mathcal{BG} := (G_1, G_2, G_T, e, p, \mathcal{P}_1, \mathcal{P}_2)$:

(1) Pairing product equations, with $A_1 = G_1, A_2 = G_2, A_T = G_T$, $f([\mathsf{x}]_1, [\mathsf{y}]_2) = e([\mathsf{x}]_1, [\mathsf{y}]_2) = [\mathsf{xy}]_T \in G_T$.
(2) Multi-scalar multiplication equations in G_1, with $A_1 = A_T = G_1$, $A_2 = \mathbb{Z}_p$, $f([\mathsf{x}]_1, \mathsf{y}) = [\mathsf{xy}]_1 \in G_1$.
(3) Multi-scalar multiplication equations in G_2, with $A_1 = \mathbb{Z}_p$, $A_2 = A_T = G_2$, $f(\mathsf{x}, [\mathsf{y}]_2) = [\mathsf{xy}]_2 \in G_2$.
(4) Quadratic equations in $\mathbb{Z}_p$, with $A_1 = A_2 = A_T = \mathbb{Z}_p$, $f(\mathsf{x}, \mathsf{y}) = \mathsf{xy} \in \mathbb{Z}_p$.

Essentially, the GS proof system is a commit-and-prove scheme in the sense that it works in two steps: first, the prover commits to a satisfying assignment for the equation and second, it proves that the equation is satisfied using the committed assignment.

2.6.1 *Specifying a GS Instantiation.* Groth and Sahai presented their proof system following a general algebraic framework and also gave two different instantiations of this framework in prime order bilinear groups. Such an instantiation is specified by the items described below.

a) A choice of the (prime order) bilinear group type, that is, specifying whether the group is symmetric or asymmetric.
b) A decisional assumption in $G_1^{n_1}$, $n_1 \in \mathbb{N}$ and some vectors $[\vec{u}_1]_1, \ldots, [\vec{u}_{n_1}]_1 \in G_1^{n_1}$. For each possible A_1 (cf. equation types), a map $\iota_1\colon A_1 \to G_1^{n_1}$ and some commitment scheme for elements in A_1, where the commitment keys are $[\vec{u}_1]_1, \ldots, [\vec{u}_{k_1}]_1 \in G_1^{n_1}$ for some $k_1 \leq n_1$ which depends on A_1.
c) A decisional assumption in $G_2^{n_2}$, $n_2 \in \mathbb{N}$ and some vectors $[\vec{v}_1]_2, \ldots, [\vec{v}_{n_2}]_2 \in G_2^{n_2}$. For each possible A_2, a map $\iota_2\colon A_2 \to G_2^{n_2}$ and some commitment scheme to elements of A_2, where the commitment keys are $[\vec{v}_1]_2, \ldots, [\vec{v}_{k_2}]_2 \in G_2^{n_2}$ for some $k_2 \leq n_2$, which depends on A_2.
d) A composite bilinear map $\tilde{e}\colon G_1^{n_1} \times G_2^{n_2} \to G_T^{n_T}$ with certain properties in the commitment space[4], and for each possible (A_1, A_2, A_T, f) a map $\iota_T\colon A_T \to G_T$, such that for all $a_1 \in A_1$, $a_2 \in A_2$ we have $\iota_T(f(a_1, a_2)) = \tilde{e}(\iota_1(a_1), \iota_2(a_2))$.

The most efficient instantiation of the GS proof system (cf. Sec. 2.6.5) is based on the SXDH Assumption.

2.6.2 *Prover.* A prover who wants to convince a verifier that an equation of the form (2) is satisfiable, holds a witness of satisfiability $(x_1, \ldots, x_{m_x}) \in A_1^{m_x}$, $(y_1, \ldots, y_{m_y}) \in A_2^{m_y}$, which is an assignment of the indeterminates $\vec{\mathsf{x}}, \vec{\mathsf{y}}$ which satisfies the equation. The prover sends to the verifier:

(1) $[\overline{x}_i]_1 \in G_1^{n_1}$, $i = 1, \ldots, m_x$, which are commitments to $x_1, \ldots, x_{m_x}$.
(2) $[\overline{y}_j]_2 \in G_2^{n_2}$, $j = 1, \ldots, m_y$, which are commitments to $y_1, \ldots, y_{m_y}$.
(3) Some vectors $[\vec{\pi}_1]_2, \ldots, [\vec{\pi}_{k_1}]_2 \in G_2^{n_2}$, and $[\vec{\theta}_1]_1, \ldots, [\vec{\theta}_{k_2}]_1 \in G_1^{n_1}$, called the proof.

Note that strictly speaking the three items together constitute the proof, but this terminology comes from thinking of GS proofs as a commit-and-prove scheme. The actual description of how the commitments and the proof are computed is irrelevant for our results, where we only care about saving operations for verification.

[3] We just assume that the order of the bilinear group is sufficiently large and chosen according to the security parameter.

[4] Following the terminology of [28, 36], the bilinear map has to be projecting.

2.6.3 *Verifier.* The verification runs in two steps.

Step 1 The verifier defines some values $[\overline{a}_i]_1 \in G_1^{n_1}$, $i = 1, \dots, m_y$, $[\overline{b}_i]_2 \in G_2^{n_2}$, $i = 1, \dots, m_x$, and $[\overline{t}]_T \in G_T^{n_T}$, which can be derived from the constants $a_1, \dots, a_{m_y} \in G_1$, $b_1, \dots, b_{m_x} \in G_2$ and $t \in G_T$ defining the equation.

Step 2 The verifier accepts a proof if and only if

$$\sum_{i=1}^{m_y} \tilde{e}([\overline{a}_i]_1, [\overline{y}_i]_2) + \sum_{i=1}^{m_x} \tilde{e}([\overline{x}_i]_1, [\overline{b}_i]_2) + \sum_{i=1}^{m_x} \sum_{j=1}^{m_y} \gamma_{ij} \tilde{e}([\overline{x}_i]_1, [\overline{y}_j]_2)$$
$$\overset{?}{=} [\overline{t}]_T + \sum_{i=1}^{k_1} \tilde{e}([\vec{u}_i]_1, [\vec{\pi}_i]_2) + \sum_{i=1}^{k_2} \tilde{e}([\vec{\theta}_i]_1, [\vec{v}_i]_2), \quad (3)$$

holds, with the rest of the elements defined as said before, namely: $[\vec{u}_i]_1$ (resp. $[\vec{v}_i]_2$) are the commitment keys to elements in A_1 (resp. A_2), and $[\vec{\pi}_i]_2, [\vec{\theta}_i]_1$ is the proof.

When the equation has no terms in $\vec{x}$ (resp. $\vec{y}$), the verification equation is also of this form with $k_1 = 0$ (resp. $k_2 = 0$).

2.6.4 *Reducing Verification Cost.* Note that even without batching, the verifier can optimize the verification equation (3) before evaluating it so that it requires less computations of $\tilde{e}$. In particular, the number of evaluations of $\tilde{e}$ can always be brought down to at most $m_x + m_y$ for the left-hand side of (3) by grouping all the terms with the same first or second argument, e.g. replacing $\tilde{e}([\overline{a}_j]_1, [\overline{y}_j]_2) + \sum_{i=1}^{m_x} \gamma_{ij} \tilde{e}([\overline{x}_i]_1, [\overline{y}_j]_2)$ by $\tilde{e}([\overline{a}_j + \sum_{i=1}^{m_x} \gamma_{ij} [\overline{x}_i]_1, [\overline{y}_j]_2)$. This is only an upper bound. In practice, the cost tends to be much lower: for instance, if there are no linear terms in $\vec{x}$ (i.e. $[\overline{b}_i]_2 = [\vec{0}]_2$), then the number of required evaluations of $\tilde{e}$ for the left-hand side is bounded by m_y. This trick can be applied to every GS instantiation and equation type.

For multiscalar and quadratic equations there are additional optimizations possible, due to the knowledge of discrete logarithms of some of the constants. For example, if we have a multiscalar equation in G_1, that is $A_1 = A_T = G_1$, $A_2 = \mathbb{Z}_p$, then $b_1, \dots, b_{m_x} \in \mathbb{Z}_p$. By definition, $[\overline{b}_i]_2 = \iota_2(b_i) = b_i \iota_2(1) \in G_2^{n_2}$. By bilinearity,

$$\sum_{i=1}^{m_x} \tilde{e}([\overline{x}_i]_1, [\overline{b}_i]_2) = \tilde{e}\Big(\sum_{i=1}^{m_x} b_i [\overline{x}_i]_1, \iota_2(1)\Big) \quad (4)$$

and the right-hand side of Eq. (4) can be computed by the verifier since b_i is a known constant describing the equation. Thus, the linear terms in $\vec{x}$ can be grouped in a single $\tilde{e}$-pairing, resulting in at most $\min(m_x + m_y, m_y + 1)$ evaluations of $\tilde{e}$ for the left-hand side of (3). Clearly, an analogous trick works if $A_1 = \mathbb{Z}_p$ and $A_2 = G_2$ and for quadratic equations.

2.6.5 SXDH *Instantiation.* The SXDH instatiation of GS proofs [35] is the most efficient setting in terms of proof size, in the number of group elements, and in the number of basic pairings for verification. Furthermore, asymmetric bilinear groups for which SXDH is assumed to hold, have more efficient arithmetic. To specify this instantiation we follow Sec. 2.6.1.

We use prime order asymmetric bilinear groups, for which the SXDH assumption holds (i.e. DDH in G_1 and G_2), and set $n_1 = n_2 = 2$. Moreover, the bilinear map $\tilde{e}$ is the tensor product, i.e. $\tilde{e}\colon G_1^{n_1} \times G_2^{n_2} \to G_T^{n_T}$ is defined by

$$\tilde{e}\left(\begin{pmatrix}[x_1]_1\\ [x_2]_1\end{pmatrix}, \begin{pmatrix}[y_1]_2\\ [y_2]_2\end{pmatrix}\right) = \begin{pmatrix}[x_1y_1]_T & [x_1y_2]_T\\ [x_2y_1]_T & [x_2y_2]_T\end{pmatrix}.$$

The common reference string includes $[\vec{u}_1]_1, [\vec{u}_2]_1 \in G_1^2$ and $[\vec{v}_1]_2, [\vec{v}_2]_2 \in G_2^2$. Since the distribution of the vectors does not affect the verification algorithm we omit its description. These parameters are common to all equation types. We define maps $\iota_\sigma^G : G_\sigma \to G_\sigma^2$, $\iota_\sigma^Z : \mathbb{Z}_p \to G_\sigma^2$, $\sigma \in \{1, 2\}$ as[5]

$$\iota_\sigma^G([a]_\sigma) = \begin{pmatrix}[a]_\sigma\\ [0]_\sigma\end{pmatrix} \qquad \iota_\sigma^Z(a) = \begin{cases}\sigma = 1: \ a[\vec{u}_2 + \vec{e}_1]_1\\ \sigma = 2: \ a[\vec{v}_2 + \vec{e}_1]_2\end{cases},$$

where $\vec{e}_1$ is the first vector of the canonical basis of $\mathbb{Z}_p^2$. The maps $\iota_1, \iota_2, \iota_T$ in Sec. 2.6.1 b) and c) and the number of commitment keys depend on the equation type:

(1) Pairing product equations: $k_1 = k_2 = 2$, $\iota_1 = \iota_1^G$, $\iota_2 = \iota_2^G$, $\iota_T := \vec{e}_1 \vec{e}_1^\top [a]_T$.
(2) Multi-scalar multiplication equations in G_1: $k_1 = 2$, $k_2 = 1$, $\iota_1 = \iota_1^G$, $\iota_2 = \iota_2^Z$, $\iota_T([a]_1) = \tilde{e}(\iota_1^G([a]_1), \iota_2^Z(1))$.
(3) Multi-scalar multiplication equations in G_2: $k_1 = 1$, $k_2 = 2$, $\iota_1 = \iota_1^Z$, $\iota_2 = \iota_2^G$, $\iota_T([a]_2) = \tilde{e}(\iota_1^Z(1), \iota_2^G([a]_2))$.
(4) Quadratic equations in $\mathbb{Z}_p$: $k_1 = k_2 = 1$, $\iota_1 = \iota_1^Z$, $\iota_2 = \iota_2^Z$, $\iota_T(a) = \tilde{e}(\iota_1^Z(1), \iota_2^Z(a))$.

3 POLYNOMIAL VIEW OF COMPOSITE PAIRINGS

Herold et al. [36] derived new instantiations of GS proofs by introducing a polynomial viewpoint on composite projecting[6] bilinear maps.

More specifically, given a basic prime-order bilinear group setting $\mathcal{BG} = (G_1, G_2, G_T, e, p, \mathcal{P}_1, \mathcal{P}_2)$ and a choice of decisional assumptions in G_1 and G_2 (e.g., SXDH), which determines parameters n_1, n_2, their work [36] identifies vectors from $G_1^{n_1}$ and $G_2^{n_2}$ with polynomials in some spaces $V_1, V_2 \subset \mathbb{Z}_p[X_1, \dots, X_\mu]$. Under this identification, the composite pairing $\tilde{e}\colon G_1^{n_1} \times G_2^{n_2} \to G_T^{n_T}$ becomes polynomial multiplication of polynomials in V_1 and V_2. In this case, n_T is the dimension of the space V_T, which is the vector space spanned by all $\{\mathbf{fg} : \mathbf{f} \in V_1, \mathbf{g} \in V_2\}$ and the map $\tilde{e}$ can be evaluated with n_T basic pairing operations. The interesting point is that [36] show that the map $\tilde{e}$ defined in this way is optimal in the sense that it is the one with minimal n_T compatible with a given decisional assumption. Thus, it also requires the minimal number of evaluations of the basic pairing e.

In this paper, we are not interested in the full power of the framework, but only in the polynomial interpretation of (projective) bilinear maps $\tilde{e}\colon G_1^{n_1} \times G_2^{n_2} \to G_T^{n_T}$ of [36]. The basic idea of the polynomial view is to interpret the $\mathbb{Z}_p$-vector spaces $G_1^{n_1}$, $G_2^{n_2}$ and $G_T^{n_T}$ as (coefficients of) polynomials in the variables $\vec{X} = (X_1, \dots, X_\mu)$. That is, we find subspaces $V_1, V_2, V_T \subseteq \mathbb{Z}_p[\vec{X}]$ with respective bases B_P, B_Q, B_R such that $G_\sigma \cong V_\sigma$ for $\sigma \in \{1, 2, T\}$ and the pairing $\tilde{e}$ corresponds to polynomial multiplication. The space V_T is the vector space generated by $\{\mathbf{fg} \mid \mathbf{f} \in V_1, \mathbf{g} \in V_2\}$.

[5] Note that in [35] the vector $\vec{e}_2$ is used in ι_σ^Z. Using $\vec{e}_1$ will simplify our notation.
[6] This is the essential property of the map $\tilde{e}$ necessary to construct GS proofs [28].

Definition 3.1. Let $B_P := \{\mathbf{p}_1, \dots, \mathbf{p}_{n_1}\}$, $B_Q := \{\mathbf{q}_1, \dots, \mathbf{q}_{n_2}\}$ be sets of linearly independent vectors in $\mathbb{Z}_p[X_1, \dots, X_\mu]$. Let $B_R := \{\mathbf{r}_1, \dots, \mathbf{r}_{n_T}\}$ be a basis of $V_T := \langle\{\mathbf{pq} : \mathbf{p} \in B_P, \mathbf{q} \in B_Q\}\rangle$. We say a bilinear map $\tilde{e}\colon G_1^{n_1} \times G_2^{n_2} \to G_T^{n_T}$ *admits a polynomial interpretation* given by B_P, B_Q, B_R iff the pairing $\tilde{e}$ satisfies[7]

$$\tilde{e}([\vec{f}]_1, [\vec{g}]_2) = [\vec{h}]_T \quad \Longleftrightarrow \quad (\sum_{i=1}^{n_1} f_i \mathbf{p}_i)(\sum_{j=1}^{n_2} g_j \mathbf{p}_j) = \sum_{k=1}^{n_T} h_k \mathbf{r}_k$$

for all $\vec{f} \in \mathbb{Z}_p^{n_1}$, $\vec{g} \in \mathbb{Z}_p^{n_2}$, $\vec{h} \in \mathbb{Z}_p^{n_T}$. In this case, we identify $[\vec{f}]_1$ with $[\mathbf{f}]_1 := \sum_{i=1}^{n_1} [f_i]_1 \mathbf{p}_i(\vec{X})$, $[\vec{g}]_2$ with $[\mathbf{g}]_2 := \sum_{j=1}^{n_2} [g_j]_2 \mathbf{q}_j(\vec{X})$ and $[\vec{h}]_T$ with $[\mathbf{h}]_T = \sum_{k=1}^{n_T} [h_k]_T \mathbf{r}_k(\vec{X})$, and write $\tilde{e}([\mathbf{f}]_1, [\mathbf{g}]_2) = [\mathbf{fg}]_T = [\mathbf{h}]_T$.

Note that this definition implies that the image $\operatorname{Im} \tilde{e}$ of $\tilde{e}$ spans all of $G_T^{n_T}$. When we will be using the polynomial interpretation in the following, it is most natural to assume that this is the case (e.g. in GS proofs, if the image of $\tilde{e}$ is contained in a subspace, one can define another map $\tilde{e}'\colon G_1^{n_1} \times G_2^{n_2} \to G_T^{n'_T}$ for some $n'_T < n_T$ which can also be used to verify the proof).

To cover all GS instantiations from the literature, if the image of $\tilde{e}$ does not span $G_T^{n_T}$, we define a polynomial interpretation of $\tilde{e}$ as a polynomial interpretation of $\tilde{e}'\colon G_1^{n_1} \times G_2^{n_2} \to \langle \operatorname{Im} \tilde{e} \rangle \cong G_T^{n'_T}$.

Example 3.2. (Tensor Product) Let us consider the pairing given by $\tilde{e}\colon G_1^2 \times G_2^2 \to G_T^{2\times 2}$ with

$$\tilde{e}([\vec{x}]_1, [\vec{y}]_2) = [xy^\top]_T = \begin{pmatrix} [x_1y_1]_T & [x_1y_2]_T \\ [x_2y_1]_T & [x_2y_2]_T \end{pmatrix} \tag{5}$$

This is the pairing used for the SXDH instantiation of the GS proof system, which is sketched in Sec. 2.6.5. Some possible choices of polynomial views are:

(1) $B_P := \{1, X\}$, $B_Q := \{1, Y\}$ and $B_R := \{1, X, Y, XY\}$. In $G_T^{2\times 2}$, the (i, j)-th unit matrix $E_{i,j} = e_i e_j^\top$ corresponds to $X_i Y_j$.
(2) $B_P := \{X_1, X_2\}$, $B_Q := \{Y_1, Y_2\}$, $B_R := \{X_1Y_1, X_2Y_1, X_1Y_2, Y_1Y_2\}$.
(3) $B_P := \{1, X\}$, $B_Q := \{1, X^2\}$ and $B_R := \{1, X, X^2, X^3\}$.

These polynomial interpretations are easily adapted for higher dimensional tensor products $\tilde{e}\colon G_1^{n_1} \times G_2^{n_2} \to G_T^{n_1\times n_2}$, $\tilde{e}([\vec{x}]_1, [\vec{y}]_1) = [xy^\top]_T$ with $n_1, n_2 \in \mathbb{N}$. For example, to extend (1), one sets $B_P := \{1, X_1, \dots, X_{n_1}\}$, $B_Q := \{1, Y_1, \dots, Y_{n_2}\}$. In $G_T^{n_1\times n_2}$, the (i, j)-th unit matrix $E_{i,j} = e_i e_j^\top$ then corresponds to $X_i Y_j$.

The definition does not exclude $B_P = B_Q$. In fact, this is an important special case which appears naturally when instantiating GS proofs in symmetric bilinear groups, e.g. under the decisional linear assumption. We stress however that the type of bilinear group (asymmetric or symmetric) over which $\tilde{e}$ is defined is independent of whether the map $\tilde{e}$ is symmetric or not (in the sense that $\tilde{e}([\vec{a}]_1, [\vec{b}]_2) = \tilde{e}([\vec{b}]_1, [\vec{a}]_2)$ may or may not hold).

Example 3.3. (Symmetric Tensor Product) Let us consider the pairing $\tilde{e}\colon G_1^3 \times G_2^3 \to G_T^6$ with

$$\begin{aligned} \tilde{e}([\vec{x}]_1, [\vec{y}]_2) = ([x_1y_1]_T, [x_1y_2 + x_2y_1]_T, [x_1y_3 + x_3y_1]_T, \\ [x_2y_2]_T, [x_2y_3 + x_3y_2]_T, [x_3y_3]_T) \end{aligned} \tag{6}$$

A possible polynomial view is $B_P = B_Q := \{1, X_1, X_2\}$ and $B_R := \{1, X_1, X_2, X_1^2, X_1X_2, X_2^2\}$. This polynomial interpretation can be adapted to other dimensions in the obvious way.

With the polynomial view we can do the usual operations with polynomials "in the exponent". Most notably, one can evaluate the polynomials at any point $(x_1, \dots, x_\mu)$ via exponentiations in the respective group. For instance, to evaluate $[\mathbf{f}]_1$ at $\vec{x}$ means to compute $[\mathbf{f}(\vec{x})]_1 = \sum_{i=1}^{n_1} [f_i]_1 \mathbf{p}_i(\vec{x}) \in G_1$ and similarly for $[\mathbf{g}]_2$ and $[\mathbf{h}]_T$. Polynomial evaluation commutes with the pairing, i.e.

$$\tilde{e}([\mathbf{f}]_1, [\mathbf{g}]_2)(\vec{x}) = [\mathbf{fg}]_T(\vec{x}) = e([\mathbf{f}(\vec{x})]_1, [\mathbf{g}(\vec{x})]_2) \tag{7}$$

Note that for the evaluation, one can use multi-exponentiation algorithms (e.g., see [43] for an overview). However, if the point at which we need to evaluate has small coefficients (or is optimized for fast exponentiation in some other way), one can instead use Horner's rule to leverage that.

3.1 Polynomial Identity Checking in the Exponent

By the discussion above, it is obvious that one can verify any claim involving several evaluations of a pairing $\tilde{e}$ for which there is a polynomial view given by B_P, B_Q, B_R using the standard probabilistic algorithm for polynomial identity checking (cf. Schwartz–Zippel, Sec. 2.5) in the exponent: given a claim which can be interpreted as a polynomial equation

$$\sum_{i=1}^{r} \tilde{e}([\mathbf{f}_i]_1, [\mathbf{g}_i]_2) \stackrel{?}{=} [\mathbf{h}]_T$$

over some ring $\mathbb{Z}_p[X_1, \dots, X_\mu]$ for some $r \in \mathbb{N}$, the equation can be verified by sampling $\vec{s} \leftarrow S^\mu$, for some arbitrary set S and evaluating the polynomials at $\vec{s}$. The soundness error is at most $d/|S|$, where d is the maximal degree of the polynomials in B_R.

Expression (7) shows that this requires r basic pairing evaluations. On the other hand, note that evaluation at $\vec{s}$ for $[\mathbf{f}_i]_1$ requires to compute $\sum_{j=1}^{n_1} [f_{ij}]_1 \mathbf{p}_j(\vec{s}) \in G_1$ with coefficients given by $\mathbf{p}_j(\vec{s})$, $j = 1, \dots, n_1$ and similarly for G_2 with B_Q. In particular, the efficiency of the involved exponentiations depends on the choice of B_P, B_Q, B_R and S. A standard choice for S is $S = \{0, 1, \dots, 2^{80+\log_2 d} - 1\}$, where d is the maximum degree of a polynomial in B_R. The resulting soundness error is 2^{-80}.

Consider Example 3.2, fixing polynomial view (1). In this case, we have $d = 2$ and evaluating elements at $(X = s_1, Y = s_2)$ requires 1 small exponentiation in G_1 and 1 in G_2. Evaluating elements in G_T^4 requires 3 small exponentiations[8]. Obviously, the choice of polynomial view (2) is less interesting because it requires more randomness for checking equality and more exponentiations. On the other hand, choice (3) is interesting if one wants to save randomness, but some of the exponents are no longer small when evaluating in G_2.

[7] Since each set $\{\mathbf{p}_1, \dots, \mathbf{p}_{n_1}\}$, $\{\mathbf{q}_1, \dots, \mathbf{q}_{n_2}\}$, $\{\mathbf{r}_1, \dots, \mathbf{r}_{n_T}\}$ is a basis of $G_1^{n_1}$, $G_2^{n_2}$, $G_T^{n_T}$, respectively, this equivalence uniquely defines the pairing.

[8] We can compute this as $[\mathbf{f}(x, y)]_T = ([f_{1,1}] + x[f_{1,2}]) + y([f_{2,1}] + x[f_{2,2}]_T)$ by Horner's rule.

We note that other choices of S can lead to significant speed-ups, as reported in [23]. For batch verification of exponentiations, the authors suggest to choose the coefficients which define the combined claim as a linear combination of the atomic claims from different sets S. For instance, if S is the set of exponents with small Hamming weight (e.g. Hamming weight at most 19 for a soundness error of 2^{-80} is sufficient) the authors report a 2 to 3 speed-up factor for verification, the reason being that usually squaring is cheaper than multiplication.

4 BATCH VERIFICATION TECHNIQUES FOR PAIRING-BASED CLAIMS

Recall that we consider the following problem: Given group elements $\left[a_i^{(j)}\right]_1 \in G_1$, $\left[b_i^{(j)}\right]_2 \in G_2$, $\left[c^{(j)}\right]_T \in G_T$ and $\alpha_i^{(j)} \in \mathbb{Z}_p$, we wish to verify a set of N' claims $C^{(j)}$ of the form

$$\sum_{i=1}^{m^{(j)}} \alpha_i^{(j)}\, e\left(\left[a_i^{(j)}\right]_1, \left[b_i^{(j)}\right]_2\right) \stackrel{?}{=} \left[c^{(j)}\right]_T \qquad \text{with } 1 \le j \le N' \tag{8}$$

more efficiently than verifying each claim individually. Typically, the $a_i^{(j)}$ and similarly the $b_i^{(j)}$ are not all distinct; in particular, the same group element may appear in several claims. In fact, if there are no known relations among the $a_i^{(j)}$'s and $b_i^{(j)}$, neither previous work nor our batching technique will give any improvement compared to treating each equation separately. We make the following assumptions regarding computational costs: pairings are by far the most expensive operations, followed by (in this order): exponentiations in G_T, exponentiations in G_2, exponentiations in G_1, followed by group operations. This is justified by the state-of-art in elliptic curves. An exact estimation of this relative cost depends on several factors (representation of the points, choice of elliptic curve, underlying hardware, exponentiation algorithms, etc) which is out of the scope of this paper. Hence, our primary goal is to minimize the number of pairing applications.

In the following we provide an overview of existing batch verification and optimization techniques as well as present our own. Which (combination of) techniques will yield the best performance gain strongly depends on the concrete structure the system of claims. Hence, we will only compare the different approaches later when considering more concrete claims.

4.1 Previous Work

The basic idea of batching, as first introduced by Bellare et al. [8] in the context of batch verification of exponentiation equations, consists of two phases. First, consider a random linear combination of the equations. Second, minimize the computational cost of evaluating the single resulting equation.

In our setting, this means that, in the first phase, we combine the N' equations of the form (8) into a single equation of the form

$$\sum_{j=1}^{N'} r^{(j)} \sum_{i=1}^{m^{(j)}} \alpha_i^{(j)}\, e\left(\left[a_i^{(j)}\right]_1, \left[b_i^{(j)}\right]_2\right) \stackrel{?}{=} \sum_{j=1}^{N'} r^{(j)} \cdot \left[c^{(j)}\right]_T \ , \tag{9}$$

where $r^{(j)}$ are chosen randomly according to some distribution. This introduces a soundness error, i.e. there is some probability that (9) is satisfied even if not all of the individual claims hold. The exact soundness error depends on the choice of distribution for the $r^{(j)}$'s. In the second phase, one would try to minimize the cost of verifying (9), which according to our cost model means minimizing the number of evaluations of e.

First Phase: Small Exponent Batching. Ferrara et al. [26], following [8], choose $r^{(j)}$ uniformly independent in $S = \{1, \dots, 2^\ell\}$, which gives a soundness error of $2^{-\ell}$ (we assume $2^\ell < p$). This choice of S is called Small Exponent Batching and is meant to minimize the cost of exponentiations.

Second Phase: Transforming Equations. In our notation, Ferrara et al. [26] suggest to transform the resulting equation (9) by the following rules for the second phase:

(1) *Move scalar (exponent) into pairing.* As exponentiation in G_1 is less expensive than in G_T (and G_2), replace a term of the form $\alpha \cdot e([a]_1, [b]_2)$ with $[a]_1 \in G_1, [b]_2 \in G_2, \alpha \in \mathbb{Z}_p$ by $e(\alpha[a]_1, [b]_2)$.
(2) *Move sum into pairing.* When two applications of the pairing share the same first or second argument, use bilinearity to combine them. Say if $[b]_2 \in G_2$, one can replace the expression $\alpha \cdot e([a]_1, [b]_2) + \alpha' \cdot e([a']_1, [b]_2)$ by $e(\alpha[a]_1 + \alpha'[a']_1, [b]_2)$.
(3) *Switch two sums.* The number of pairings may also be reduced by moving a sum from the first to the second component (or vice versa). Consider $\sum_{j=1}^{t} e(\sum_{i=1}^{k} \alpha_{i,j}[a_i]_1, [b_j]_2)$. If $t < k$, then $k - t$ pairing applications can be saved by replacing this term by $\sum_{i=1}^{k} e([a_i]_1, \sum_{j=1}^{t} \alpha_{i,j}[b_j]_2)$.

Note that these rules are a heuristic and may be conflicting. Also, the number of pairing operations saved might come at the cost of increasing other operations, so the precise gains depend on the relative costs of these operations. These techniques apply generally and we will also use them for our second phase.

4.2 Our Techniques

0^{th} Phase. Our techniques are based on polynomial evaluation and we introduce an additional 0^{th} phase. In this phase, we rewrite the claims we wish to verify as polynomial equations. The extent to which this is possible, and consequently the amount by which we improve over previous results, depends strongly on the structure of the claims. Note that GS proofs are particularly amenable to our techniques, since the verification equation (3) uses a pairing $\tilde{e}$ that has a polynomial interpretation for every instantiation from the literature (cf. Appendix B).

To simplify the exposition, we assume at first that the claims can be directly written as a single polynomial equation; the more general case will be dealt with below in Sec. 4.3. This means that we have $N' = n_T$ claims (over G_T) that can be jointly written as

$$\sum_{i=1}^{m} \alpha_i \cdot \tilde{e}([\vec{a}_i]_1, [\vec{b}_i]_2) \stackrel{?}{=} [\vec{c}]_T \tag{10}$$

for $\alpha_i \in \mathbb{Z}_p$, where $\tilde{e}\colon G_1^{n_1} \times G_2^{n_2} \to G_T^{n_T}$ is a bilinear map which admits a polynomial interpretation. The $N' = n_T$ claims over G_T in (10) then correspond to a single claim of the multivariate polynomial identity $\sum_i \alpha_i \mathbf{a}_i \cdot \mathbf{b}_i \stackrel{?}{=} \mathbf{c}$. In the 0^{th} phase, we choose a particular polynomial interpretation (In general, there may be several choices that perform differently in our batching scheme.).

First Phase. We suggest to instantiate the first phase (in the language of Sec. 4.1) by using the classical probabilistic polynomial identity checking algorithm in the exponent, i.e. we verify

$$\sum_{i}^{m} \alpha_i \cdot e([\mathbf{a}_i(\vec{s})]_1, [\mathbf{b}_i(\vec{s})]_2) \overset{?}{=} [\mathbf{c}(\vec{s})]_T \tag{11}$$

for a uniformly randomly chosen $\vec{s} \in \mathbb{Z}_p^{\mu}$ (or more generally uniform from a sufficiently large subset $\vec{s} \in S^{\mu} \subset \mathbb{Z}_p^{\mu}$), where μ is the number of variables. We note that Eq. (11) can be evaluated by only m applications of the basic pairing e. The evaluations at $\vec{s}$ correspond to multi-exponentiations as explained in Sec. 3.1. For efficiency, we evaluate first at $\vec{s}$, then we multiply.

Let us write this out explicitly in terms of the basic pairing e. For this, assume that we have a polynomial interpretation given by $\mathbf{p}_i$'s, $\mathbf{q}_i$'s and $\mathbf{r}_i$'s. Then Equation (11) becomes

$$\sum_{i=1}^{m} \alpha_i \cdot e\Big(\sum_{j_1=1}^{n_1} \mathbf{p}_{j_1}(\vec{s})[a_{i,j_1}]_1, \sum_{j_2=1}^{n_2} \mathbf{q}_{j_2}(\vec{s})[b_{i,j_2}]_2\Big) \overset{?}{=} \sum_{j_T=1}^{n_T} \mathbf{r}_{j_T}(\vec{s})[c_{j_T}]_T. \tag{12}$$

Here, $[a_{i,j_1}]_1$, $[b_{i,j_2}]_2$, $[c_{j_T}]_T$ are the coefficients of $[\vec{a}_i]_1$, $[\vec{b}_i]_2$, and $[\vec{c}]_T$.

Soundness loss. We can easily compute the soundness loss in our first phase using the Schwartz–Zippel lemma:

Theorem 4.1. *Consider a set of n_T claims (over G_T) of the form Eq.* (10) *with some given polynomial interpretation. If all n_T individual claims are true, then Eq.* (11) *holds for every choice of $\vec{s} \in \mathbb{Z}_p^{\mu}$, so verification always succeeds. Conversely, assume that at least one of the n_T claims is false. Then, for a uniformly random choice of $\vec{s} \in S^{\mu}$, Eq.* (11) *only holds with probability (soundness error)*

$$\Pr_{\vec{s} \in S^{\mu}}[\text{Verification succeeds} \mid \text{Some claim does not hold}] \leq d/|S|$$

over the choice of $\vec{s} \in S^{\mu}$, where d is the maximal degree of the polynomials $\mathbf{r}_i$ in the polynomial interpretation.

Proof. Since Eq. (10) corresponds to an equation in the space of polynomials spanned by the $\mathbf{r}_i$'s, the claim is an immediate consequence of the Schwartz–Zippel lemma. □

As the soundness loss only depends on the size $|S|$ of S, we choose the set S such that it reduces the cost of exponentiations.

Second Phase. For the second phase (i.e. reorganizing the evaluation), we suggest to use the heuristic techniques from Ferrara et al. [26] as described in Sec. 4.1, starting from Eq. (11) (or, written more explicitly, Eq. (12)).

4.3 Systems of Equations

In many cases, it may not be obvious or natural to write the claims as a single polynomial equation. So consider the case that in the 0th phase, we do not write our claims as a single polynomial equation, but rather as a system of equations. Concretely, we assume that our N' claims (over G_T) can be written as a system of N equations

$$\sum_{i=1}^{m^{(j)}} \alpha_i^{(j)} \cdot \tilde{e}^{(j)}([\vec{a}_i^{(j)}]_1, [\vec{b}_i^{(j)}]_2) \overset{?}{=} [\vec{c}^{(j)}]_T \qquad \text{for } 1 \leq j \leq N \tag{13}$$

where the superscript ranges over the N individual (composite) equations. Each such composite equation uses a bilinear pairing $\tilde{e}^{(j)}\colon G_1^{n_1^{(j)}} \times G_2^{n_2^{(j)}} \to G_T^{n_T^{(j)}}$ which admits a polynomial interpretation given by bases $\{\mathbf{p}_1^{(j)}, \ldots\}$, $\{\mathbf{q}_1^{(j)}, \ldots\}$, $\{\mathbf{r}_1^{(j)}, \ldots\}$, which span vector spaces of polynomials $V_1^{(j)}$, $V_2^{(j)}$, $V_T^{(j)}$. So the jth equation corresponds to $n_T^{(j)}$ equations over G_T and we have $N' = \sum_j^N n_T^{(j)}$. Typically, $\tilde{e}^{(j)}$ and the polynomial interpretation do not actually depend on j. We emphasize that the "unstructured" case where each $\tilde{e}^{(j)}$ is just the basic pairing e is allowed and interesting. In particular, writing the claims in the form (13) is always possible.

Via the polynomial interpretation, the set of claims (13) may be viewed as a system of N polynomial equations

$$\sum_{i=1}^{m^{(j)}} \alpha_i^{(j)} \cdot \mathbf{a}_i^{(j)} \mathbf{b}_i^{(j)} \overset{?}{=} \mathbf{c}^{(j)} \qquad \text{for } 1 \leq j \leq N \tag{14}$$

for polynomials $\mathbf{a}_i^{(j)} \in V_1^{(j)}, \mathbf{b}_i^{(j)} \in V_2^{(j)}, \mathbf{c}^{(j)} \in V_T^{(j)}$.

To combine the equations, choose non-zero polynomials $\mathbf{f}^{(j)}$ such that the set of all $\mathbf{f}^{(j)}\mathbf{r}_i^{(j)}$ is linearly independent. Next, multiply the jth equation with $\mathbf{f}^{(j)}$, resulting in the equivalent system

$$\sum_{i=1}^{m^{(j)}} \alpha_i^{(j)} \cdot \mathbf{f}^{(j)} \cdot \mathbf{a}_i^{(j)} \mathbf{b}_i^{(j)} \overset{?}{=} \mathbf{f}^{(j)} \cdot \mathbf{c}^{(j)} \qquad \text{for } 1 \leq j \leq N. \tag{15}$$

The jth equation is an equation in the space $\mathbf{f}^{(j)} \cdot V_T^{(j)}$, which is spanned by the $\mathbf{f}^{(j)}\mathbf{r}_i^{(j)}$'s. Since all $\mathbf{f}^{(j)}\mathbf{r}_i^{(j)}$ are linearly independent, the system (14) is equivalent to the single polynomial equation

$$\sum_{j=1}^{N} \sum_{i=1}^{m^{(j)}} \alpha_i^{(j)} \mathbf{f}^{(j)} \cdot \mathbf{a}_i^{(j)} \mathbf{b}_i^{(j)} \overset{?}{=} \sum_{j=1}^{N} \mathbf{f}^{(j)} \mathbf{c}^{(j)}, \tag{16}$$

which we can then treat the same way as we described above.

We conclude that, in general, we need to choose in the 0th phase a decomposition of the claims into N such "polynomial equations", polynomial interpretations for each of them together with polynomials $\mathbf{f}^{(j)}$ such that all $\mathbf{f}^{(j)}\mathbf{r}_i^{(j)}$'s that appear are linearly independent. The efficiency of our batching strategy depends on these choices.

Indeed, a simple and natural way to ensure linear independence of the $\mathbf{f}^{(j)}\mathbf{r}_i^{(j)}$ is choosing $\mathbf{f}^{(j)} = Z_j$ for variables Z_j that did not appear before. Another option is $\mathbf{f}^{(j)} = Z^{j-1}$ for a new variable Z, which gives a slightly larger soundness error for $k \geq 3$, but requires less randomness. Depending on the structure of the equations, we may also multiply by variables that are already present in the $\mathbf{r}_i^{(j)}$, requiring even less randomness.

For GS proofs for systems of equations, the decomposition into polynomial equations is completely natural and we have polynomial interpretations in every case of interest. For the $\mathbf{f}^{(j)}$'s, we will always choose either fresh variables $\mathbf{f}^{(j)} = Z_j$ or $\mathbf{f}^{(j)} = \mathbf{1}$.

In the first and second phase, we proceed as before and test whether

$$\sum_{j=1}^{N} \sum_{i=1}^{m^{(j)}} \alpha_i^{(j)} \mathbf{f}^{(j)}(\vec{s}) \cdot e\Big([\mathbf{a}_i^{(j)}(\vec{s})]_1, [\mathbf{b}_i^{(j)}(\vec{s})]_2\Big) \overset{?}{=} \sum_{j=1}^{N} \mathbf{f}^{(j)}(\vec{s})[\mathbf{c}^{(j)}(\vec{s})]_T \tag{17}$$

holds for uniformly random $\vec{s} \in S^{\mu}$ for a sufficiently large $S \subset \mathbb{Z}_p$. Evaluating (17) can be done with $\sum_j m^{(j)}$ evaluations of e (or fewer, if we can perform further optimizations in phase 2). As before, the soundness error is bounded by $\frac{d}{|S|}$, where d is the maximum total degree of any $\mathbf{f}_j\mathbf{r}_i^{(j)}$ that appears in the polynomial interpretations.

Internal vs. external batching. Suppose we have N (composite) equations as in (13), each of which all admit some (typically the same) polynomial interpretation for its pairing $\tilde{e}^{(j)}$. In such a situation, we distinguish *internal* and *external* batching.

By internal batching, we refer to the strategy we use to evaluate a single equation using $\tilde{e}^{(j)}$ under our chosen polynomial interpretation. This corresponds to using our techniques in the way described in Sec. 4.2 to each equation individually.

By external batching, we refer to the strategy we use to batch the N equations into a single equation, i.e. the choice of the $\mathbf{f}^{(j)}$. The externally batched equation is then

$$\sum_{j=1}^{N}\sum_{i=1}^{m^{(j)}} \alpha_i^{(j)}\mathbf{f}^{(j)}(\vec{s}) \cdot \mathbf{a}_i^{(j)}\mathbf{b}_i^{(j)} \overset{?}{=} \sum_{j=1}^{N}\mathbf{f}^{(j)}(\vec{s})\mathbf{c}^{(j)}, \quad \text{or} \tag{18}$$

$$\sum_{j=1}^{N}\sum_{i=1}^{m^{(j)}} \alpha_i^{(j)}\mathbf{f}^{(j)}(\vec{s}) \cdot \mathbf{a}_i^{(j)}(\vec{s})\mathbf{b}_i^{(j)}(\vec{s}) \overset{?}{=} \sum_{j=1}^{N}\mathbf{f}^{(j)}(\vec{s})\mathbf{c}^{(j)}(\vec{s}), \tag{19}$$

depending on whether we externally batch before or after internal batching. Note that these operations commute. If we use independent variables for internal and external batching, as we will do in our examples, these steps are completely independent.

In other words: internal batching (essentially polynomial evaluation) checks if a single equation with $\tilde{e}$ holds.

External batching only combines the N equations into one (and does not touch the $\tilde{e}^{(j)}$'s).

4.4 Relation to Previous Work

Let us briefly discuss how our batching strategy fits into the framework given by previous work and vice versa. First, let us express previous previous work in our framework: The small-exponent batching of [26] that we explained in Sec. 4.1 is actually a special case of our approach if we choose $S = \{1, \dots, 2^{\ell}\}$ and only perform external batching. Namely, consider the case where every $\tilde{e}^{(j)} = e$ is just the basic pairing and the polynomial interpretation is trivial (i.e. all $\mathbf{p}_i^{(j)} = \mathbf{q}_i^{(j)} = \mathbf{r}_i^{(j)} = \mathbf{1}$). In this case, internal batching does not do anything and we only perform external batching. The $\mathbf{f}^{(j)}$ take the role of the random linear combination that combines the claims. In particular, (17) exactly corresponds to (9), where $\mathbf{f}^{(j)}(\vec{s})$ corresponds to $r^{(j)}$. This gives an alternative direct proof for the soundness error for the small exponent batching technique. In the second phase, our techniques are the same, since we just use theirs.

However, we would like to emphasize that our approach is a generalization even for the case of only external batching, because it allows to analyze the case where the $\mathbf{f}^{(j)}$ are algebraically dependent, such as $\mathbf{f}^{(j)} = Z^j$ for the j^{th} equation. Our approach also suggests an improvement over [26], which was already observed in [2]: we may set one of the $\mathbf{f}^{(j)}$'s to be $\mathbf{1}$ and the rest to be Z_j's. This still ensures linear independence, but we save on exponentiations for the equation where $\mathbf{f}^{(j)} = \mathbf{1}$.

Conversely, let us see how our polynomial evaluation batching can be expressed in the framework given by previous work. First, we remark that our batching strategy actually fits within the framework in terms of phase 1 and phase 2 that we laid out in Sec. 4.1.

Indeed, polynomial evaluation at $\vec{s}$ in case of a single polynomial equation is equivalent to the following: treat Eq. (10) as n_T individual equations over G_T, multiply the i^{th} equation by $\mathbf{r}_i(\vec{s})$ and add them up. For systems of polynomial equations, we need to multiply by $\mathbf{f}^{(j)}\mathbf{r}_i^{(j)}(\vec{s})$ instead. In this sense, polynomial evaluation fits phase 1. This also means that we may view our technique as a special case of the Ferrara et al. [26] technique, where the randomness used to combine equations is structured as $\mathbf{f}^{(j)}\mathbf{r}_i^{(j)}(\vec{s})$.

We caution that strictly following the blueprint from Sec. 4 and using Eq. (9) corresponds to verifying (in terms of polynomials)

$$\sum_j\sum_i \alpha_i^{(j)}\mathbf{f}^{(j)}(\vec{s}) \cdot \left(\mathbf{a}_i^{(j)}\mathbf{b}_i^{(j)}\right)(\vec{s}) \overset{?}{=} \sum_j \mathbf{f}^{(j)}(\vec{s})\mathbf{c}(\vec{s}),$$

i.e. to first multiplying the polynomials $\mathbf{a}_i^{(j)}$ and $\mathbf{b}_i^{(j)}$ (using $\tilde{e}$) and then evaluating. By contrast, we actually evaluate them first and then multiply (using e). These two different ways of computing the same expression are in fact a reorganization of the resulting equation, so our techniques add a step in the second phase as well. This step is precisely enabled by the fact that our internal batching strategy is compatible with the pairing and it is this step where our improvements stem from.

4.5 Probabilistic Homomorphic Batching

In this section, we take a more abstract view on our techniques. For brevity, we restrict our attention to internal batching.

Consider a (basic) bilinear group $\mathcal{BG} := (G_1, G_2, G_T, e)$, where for notational simplicity we drop the group order p and the generators $\mathcal{P}_1, \mathcal{P}_2, \mathcal{P}_T$ from our notation. Consider a system of n_T claims of the usual form $\sum_{i=1}^{m} \alpha_i \tilde{e}([\vec{a}_i]_1, [\vec{b}_i]_2) \overset{?}{=} [\vec{c}]_T$ for some $\tilde{e}\colon G_1^{n_1} \times G_2^{n_2} \to B_T^{n_T}$. Defining $B_1 := G_1^{n_1}, B_2 := G_2^{n_2}, B_T := G_T^{n_T}$, we may consider this as a single claim $\sum_{i=1}^{m} \alpha_i \tilde{e}(a_i, b_i) \overset{?}{=} c$, where $\alpha_i \in \mathbb{Z}_p, a_i \in B_1$, $b_i \in B_2$, $c \in B_T$ over the bilinear group $\mathcal{BG}' := (B_1, B_2, B_T, \tilde{e})$.[9]

The crucial property of our technique is that evaluation at a point respects the pairing, i.e. $\mathbf{f}(\vec{s})\mathbf{g}(\vec{s}) = (\mathbf{f} \cdot \mathbf{g})(\vec{s})$. In fancy terms, evaluation at $\vec{s}$ is a *homomorphism* of bilinear groups.

Definition 4.2 (bilinear homomorphism). By $\mathrm{Hom}(\mathcal{BG}', \mathcal{BG})$, we denote the set of triples $\phi_\bullet = (\phi_1, \phi_2, \phi_T)$ where $\phi_\sigma : B_\sigma \to G_\sigma$ is linear and $\phi_\bullet$ respects the pairing, i.e. $e(\phi_1(a), \phi_2(b)) = \phi_T(\tilde{e}(a, b))$ for all $a \in B_1$, $b \in B_2$. We call $\phi_\bullet$ a *bilinear homomorphism.*

$$\begin{tikzcd}
\mathcal{BG}': \arrow[d, "\phi_\bullet"] & B_1 \times B_2 \arrow[r, "\tilde{e}"] \arrow[d, "\phi_1\times\phi_2"] & B_T \arrow[d, "\phi_T"] \\
\mathcal{BG}: & G_1 \times G_2 \arrow[r, "e"] & G_T
\end{tikzcd}$$

Any bilinear homomorphism allows us to transfer the verification equation from $\mathcal{BG}'$ to $\mathcal{BG}$ while preserving the structure of the equations. The latter is the reason that verification in $\mathcal{BG}$ only requires m pairing evaluations. Note that, for GS proofs, $\tilde{e}$ needs

[9] We deliberately write a_i instead of $[\vec{a}_i]_1$, etc., to clarify that we do *not* want to treat elements in B_σ as vectors in $G_\sigma^{n_\sigma}$ for $i = 1, 2, T$.

to have the so-called projecting property [28], which exactly corresponds to the existence of non-trivial bilinear homomorphisms.

This allows us to use what we call *probabilistic homomorphic batching*: Randomly choose $\phi_\bullet \in \mathcal{H} \subset \mathrm{Hom}(\mathcal{BG}', \mathcal{BG})$ for appropriate $\mathcal{H} \subset \mathrm{Hom}(\mathcal{BG}', \mathcal{BG})$, use $\phi_\bullet$ to transfer the equation from $\mathcal{BG}'$ to $\mathcal{BG}$ and verify in $\mathcal{BG}$. For soundness, we need that for any fixed $0 \neq c \in B_T$, the probability $\mathbf{Pr}_{\phi_\bullet \in \mathcal{H}}[\phi_T(c) = 0]$ is small. This is only achievable for specific $\tilde{e}$, e.g. if $\tilde{e}$ admits a polynomial interpretation.

5 APPLICATIONS TO GROTH–SAHAI PROOFS

The verification equation (3) of GS proofs in a bilinear group as in Sec. 2.6 checks equality of n_T equations in G_T. In case we use GS proofs for a system of equations, we use (as explained in Sec. 4.3) internal batching to verify the n_T equations of a single GS proof and external batching to simultaneously verify the proofs of many GS equations.

For all instantiations of GS proofs from the literature, there exists a polynomial interpretation such that the verification equation can be written as polynomial identity checking. This is explained in more detail in Appendix B. Let $\{\mathbf{p}_1, \ldots, \mathbf{p}_{n_1}\}$, $\{\mathbf{q}_1, \ldots, \mathbf{q}_{n_2}\}$, and $\{\mathbf{r}_1, \ldots, \mathbf{r}_{n_T}\}$ be the chosen polynomial bases, and let $X_1, \ldots, X_\mu$ be the variables and $S \subseteq \mathbb{Z}_p$. We use this polynomial interpretation of $(G_1^{n_1}, G_2^{n_2}, G_T^{n_T}, \tilde{e})$ in each GS proof. For efficiency, we suggest to use $\mathbf{p}_1 = \mathbf{q}_1 = \mathbf{r}_1 = \mathbf{1}$. We also suggest that one should choose polynomial bases and ι_i in the GS instantiation that fit together well, e.g. so that $\iota_1^G([a]_1)(\vec{s}) = [a]_1$, i.e. ι_1^G corresponds to mapping (in the exponent) $a \in \mathbb{Z}_p$ to the constant polynomial $a \in \mathbb{Z}_p[\vec{X}]$ as in Ex. 5.1 below.

Thus, we are now in the situation of Sec. 4. We start explaining how to batch a single GS verification equation, that is, we start with internal batching. We use equation (20) where all pairings with $\vec{y}$ are grouped. This is only for explanation. Our technique can be applied to an equation optimized as in Sec. 2.6.4.

$$\begin{aligned}\sum_{j=1}^{m_y} \tilde{e}\Big([\overline{a}_i]_1 + \sum_{i=1}^{m_x}[\gamma_{ij}\overline{x}_i]_1, [\overline{y}_j]_2\Big) + \sum_{i=1}^{m_x} \tilde{e}([\overline{x}_i]_1, [\overline{b}_i]_2)\\ \stackrel{?}{=} [\overline{t}]_T + \sum_{i=1}^{k_1} \tilde{e}([\vec{u}_i]_1, [\vec{\pi}_i]_2) + \sum_{i=1}^{k_2} \tilde{e}([\vec{\theta}_i]_1, [\vec{v}_i]_2) \quad (20)\end{aligned}$$

We see that the number of $\tilde{e}$-pairings is bounded by $m_x + m_y + k_1 + k_2 + \delta_t$, where $\delta_t \in \{0, 1\}$, but in practice it tends to be (much) lower, since $\gamma_{i,j}$ and $\overline{a}_i$, $\overline{b}_j$ are 0 for many i, j. The term δ_t is 0 for pairing product equations and if $t = 0$. Otherwise, computing $\iota_T(t) = [\overline{t}]_T \in G_T^{n_T}$ from $t \in A_T$ may necessitate a pairing.

Now evaluate equation (20) at $\vec{s} \leftarrow S^\mu$. For this, we introduce intermediate variables for the evaluation at $\vec{s}$, namely

$$[\widehat{a}_i]_1 = [\overline{\mathbf{a}}_i(\vec{s})]_1 \qquad [\widehat{x}_i]_1 = [\overline{\mathbf{x}}_i(\vec{s})]_1 \qquad [\widehat{u}_i]_1 = [\mathbf{u}_i(\vec{s})]_1 \quad (21)$$

and likewise for $[\widehat{\theta}_i]_1$, $[\widehat{b}_i]_2$, $[\widehat{y}_i]_2$, $[\widehat{v}_i]_2$, $[\widehat{\pi}_i]_2$, and $[\widehat{t}_i]_T$.

The resulting, internally batched equation then looks like:

$$\begin{aligned}\sum_{j=1}^{m_y} e\Big([\widehat{a}_i]_1 + \sum_{i=1}^{m_x}[\gamma_{ij}\widehat{x}_i]_1, [\widehat{y}_j]_2\Big) + \sum_{i=1}^{m_x} e([\widehat{x}_i]_1, [\widehat{b}_i]_2)\\ \stackrel{?}{=} [\widehat{t}]_T + \sum_{i=1}^{k_1} e([\widehat{u}_i]_1, [\widehat{\pi}_i]_2) + \sum_{i=1}^{k_2} e([\widehat{\theta}_i]_1, [\widehat{v}_i]_2) \quad (22)\end{aligned}$$

Beyond efficiency improvements, an advantage of our technique is that the resulting verification equation matches very closely the original GS equation, which makes it conceptually easy to use and analyze. For example, Blazy et al. [11] have different formulas for each equation type. In our case, only the number of basic pairing computations depends indirectly on the equation type, in the sense that k_1 and k_2 are dependent on the equation type.

Another interesting observation is that the number of basic pairings required to verify does not grow much with the size of the underlying decisional assumptions, which only affects the terms $k_1 + k_2$. Concretely, this means that only two additional pairings are required to verify a proof in the SXDH setting compared to the 2-Lin setting. Indeed, one composite pairing maps to one basic pairing, regardless of the arguments. In comparison, the technique of [11] needs roughly $\min(n_1, n_2)$ basic pairings per composite pairing for generic arguments. However, [11] exploits composite pairings of vectors with [0] components to save basic pairings.

For concreteness, let us exemplify our techniques in the SXDH-case, which is also the most important one in practice.

Example 5.1. Consider the SXDH instantiation in Sec. 2.6.5. For the polynomial interpretation of $(G_1^2, G_2^2, G_T^{2\times 2}, \tilde{e})$, choose the one in Ex. 3.2 (1), i.e. $\mathbf{p}_1 = \mathbf{q}_1 = \mathbf{1}$, $\mathbf{p}_2 = X$, $\mathbf{q}_2 = Y$. This means

$$\begin{aligned}&[(a_1, a_2)]_1^\top \mathrel{\widehat{=}} [x_1 + x_2 X]_1 \quad \text{and} \quad [(y_1, y_2)]_2^\top \mathrel{\widehat{=}} [y_1 + y_2 Y]_2\\ &\text{and} \quad \begin{pmatrix}[c_{1,1}]_T & [c_{1,2}]_T\\ [c_{2,1}]_T & [c_{2,2}]_T\end{pmatrix} \mathrel{\widehat{=}} [c_{1,1} + c_{2,1}X + c_{1,2}Y + c_{2,2}XY]_T\end{aligned}$$

The constants $[\overline{a}_i]_1 \in G_1^2$ are computed as $[(a_i, 0)]_1^\top = \iota_1^G([a_i]_1)$ if $A_1 = G_1$ or $a_i[\vec{u}_2 + \vec{e}_1]_1 = \iota_1^Z(a_i)$ if $A_1 = \mathbb{Z}_p$. Likewise for A_2 and $[\overline{b}_j]_2$. Remember that A_1, A_2, A_T describe the equation type (pairing product, multi-scalar, quadratic), cf. Sec. 2.6. For batching, choose $\rho, \sigma \in \mathbb{Z}_p$ and evaluate Eq. (20) at $\vec{s} = (\rho, \sigma)$, i.e. $X = \rho$, $Y = \sigma$. The intermediaries are

- $[\widehat{x}_i]_1 = [\overline{x}_{i,1} + \rho\overline{x}_{i,2}]_1 = [\overline{\mathbf{x}}_i(\vec{s})]_1$ and likewise for $[\vec{u}_i]_1$, $[\vec{\theta}_i]_1$, and $[\overline{a}_i]_1$.
- $[\widehat{y}_j]_2 = [\overline{y}_{j,1} + \sigma\overline{y}_{j,2}]_2 = [\overline{\mathbf{y}}_j(\vec{s})]_2$ and likewise for $[\vec{v}_j]_2$, $[\vec{\pi}_j]_2$ and $[\overline{a}_j]_2$
- $[\widehat{t}]_T = [\overline{t}_{1,1} + \rho\overline{t}_{2,1} + \sigma\overline{t}_{1,2} + \rho\sigma\overline{t}_{2,2}]_T$. For some A_T and $t \in A_T$, this may need a pairing.[10]

We have the following useful special cases:

- If $A_\sigma = G_\sigma$, then $[\widehat{a}_i]_\sigma = [a_i]_\sigma$.
- If $A_1 = \mathbb{Z}_p$ then only $\iota_1(1)(\vec{s}) = [\vec{u}_{2,1} + \rho\vec{u}_{2,2}]_1$ needs to be computed (as explained in Sec. 2.6.4) and the same holds for $A_2 = \mathbb{Z}_p$.

[10] This pairing corresponds to the term δ_t in the estimate for the number of pairings.

Symmetric instantiations. For symmetric GS instantiations we proceed as above. The differences to the asymmetric instantiations are more possibilities to optimize evaluation and shorter proofs. The right-hand-side of the GS verification equation only needs $\max(k_1, k_2)$ pairings instead of $k_1 + k_2$.

5.1 Batching Systems of GS Equations

For batching systems of N GS verification equations (with each such equation consisting of n_T equations over G_T), we use the techniques from Sec. 4.3. Concretely, this means that we apply internal batching to each of the N GS verifications individually (with the same $\vec{s}$) as explained in last section, resulting in N equations over G_T. We then batch the resulting equations by multiplying them with random $r_i \in S$ for $i > 1$ and $r_1 = 1$, and add them together. This corresponds to choosing $\mathbf{f}^{(i)} = Z_i$ for $i = 2 \ldots, N$ and $\mathbf{f}^{(1)} = \mathbf{1}$ in Sec. 4.3.[11]

Since internal batching preserves the structure, each of the N equations is of the form (22), whose left-hand side has the same shape as the equation (2) for which we want to prove satisfiability. Consequently, if some variables or coefficients are shared between those equations, the same is true for the internally batched one and this may allow us to combine some pairings, depending on the structure of the system. However, we can always bound the number of basic pairing evaluations by the total number of variables which define the system of equations.

For the right-hand sides, since every equation has the same $\vec{u}_i$ and $\vec{v}_j$, we can always group them together. Note that the N equations might be of different types, i.e. the values $k_1^{(j)}, k_2^{(j)}$ can vary for each GS proof. Consequently, for the right-hand side we need at most $\max_j(k_1^{(j)}) + \max_j(k_2^{(j)})$ evaluations of e for the terms with the proofs plus the pairings necessary[12] to compute $\sum_{j=1}^{N} [r_i \bar{t}^{(j)}(\vec{s})]_T$, independent of N and the number of variables or constants.

The left-hand side of our batched formula (22) matches the original equation (*not* the GS verification equation) closely. Thus, one can first externally batch the *original* equation and optimize the expression, maybe with with an automated tool such as [2]. Then, after essentially replacing the variables with the corresponding intermediate values, (e.g. replacing $[a_i]_1$ by $[\widehat{a}_i]$), and taking care of the right-hand side, this yields a valid batch-verifier for GS proofs.

Precomputations. The evaluation point $\vec{s}$ (and also the r_i's) can be chosen before the verifier receives the proof. The verifier may reuse the same $\vec{s}$ in the internal batching of several equations; we do this anyway. In fact, the prover does not learn anything about s from whether the verifier accepts a given proof (unless a dishonest prover managed to cheat in that proof), because we can simulate the verifier by doing a non-batched verification. Consequently, the verifier can choose a (secret) $\vec{s}$ ahead of time and precompute $[\widehat{u}_i]_1$'s and $[\widehat{v}_i]_2$'s. If the constants in the equations are known a priori, it may also precompute $[\widehat{a}_i]_1$, $[\widehat{b}_i]_2$ and $[\widehat{t}]_T$'s. After $\vec{s}$ was chosen, the verifier can replace $[\vec{u}_i]_1$ and $[\vec{v}_i]_2$ by $[\widehat{u}_i]_1$ and $[\widehat{v}_i]_2$, thereby saving memory. We caution that $\vec{s}$ needs to be kept secret and this optimization may lead to potential side-channel attacks.

5.2 Comparison with Blazy et al.

Blazy et al. [11] use the techniques of Ferrara et al. [26], working out the second phase for the case of GS verification equations. The batching technique of Blazy et al. does not preserve the structure of the equations, as it is essentially small exponents batching. Therefore, there are fewer opportunities to group elements, and consequently, we obtain improvements over their results. A brief comparison of the techniques is in Table 1. More details can be found in Appendix C. Note that, if external batching is again amenable to our techniques, higher efficiency gains are possible (w.r.t. small exponents batching as in [11]). See App. C.3 for an example.

		Blazy et al. [11]	this work
SXDH	PPE	$m_x + 2m_y + 8$	$m_x + m_y + 4$
	ME$_1$	$\min(2m_y + 9, 2m_x + m_y + 7)$	$\min(m_x, m_y) + 3$
	ME$_2$	$\min(2m_x + 9, 2m_y + m_x + 7)$	$\min(m_x, m_y) + 3$
	QE	$2\min(m_x, m_y) + 8$	$\min(m_x, m_y) + 2$
2-Lin	PPE	$3m + 6$	$2m + 3$
	ME	$3m_x + 3m_y + 6$	$\min(m_x, m_y) + 3$
	QE	$3m + 6$	$m + 3$

Table 1: Number of pairings to verify an (internally) batched GS proof. For simplicity, we assume evaluating $[\bar{t}(\vec{s})]_T$ needs no pairings ($\delta_T = 0$), e.g. since $t = 0$. For PPE and QE in the symmetric 2-Lin GS instance, we write $m = m_x = m_y$.

5.3 Example: P-signatures

Here we apply our techniques to the P-signatures of Belenkiy et al. [6]. P-signatures were introduced in [6] and used to construct anonymous credentials. For our purposes, it is only relevant that they offer NIZK proofs of signature possession, which in [6] were realized by GS proofs.

Blazy et al. choose P-signatures to exemplify the improvements of their batching techniques in [11] to speed up verification of proofs of P-signatures. Now we batch using our technique, and compare the results. In our notation, the verification equations of a proof of a P-signature in the SXDH instantiation are three PPEs

$$\tilde{e}([\overline{x}_1]_1, [\overline{b}_2 + \overline{y}_1 + \overline{y}_2]_2) - [\overline{1}]_T \overset{?}{=} \sum_{i=1}^{2} \tilde{e}([\vec{u}_i]_1, [\vec{\pi}_i^{(1)}]_2) + \sum_{i=1}^{2} \tilde{e}([\vec{\theta}_i^{(1)}]_1, [\vec{v}_i]_2)$$

$$\tilde{e}([\overline{x}_2]_2, [\overline{1}]_1) - \tilde{e}([\overline{a}_2]_1, [\overline{y}_2]_2) \overset{?}{=} \sum_{i=1}^{2} \tilde{e}([\vec{u}_i]_1, [\vec{\pi}_i^{(2)}]_2) + \sum_{i=1}^{2} \tilde{e}([\vec{\theta}_i^{(2)}]_1, [\vec{v}_i]_2)$$

$$\tilde{e}([\overline{x}_3]_1, [\overline{b}_3]_2) - \tilde{e}([\overline{a}_2]_1, [\overline{y}_1]_2) \overset{?}{=} \sum_{i=1}^{2} \tilde{e}([\vec{u}_i]_1, [\vec{\pi}_i^{(3)}]_2) + \sum_{i=1}^{2} \tilde{e}([\vec{\theta}_i^{(3)}]_1, [\vec{v}_i]_2)$$

We batch exactly as described in Sec. 5.1. That is, we batch the (resulting) verification equations by multiplying the i^{th} equation with r_i, where $r_1 = 1$ and $r_2, r_3 \leftarrow S$, and we evaluate each individual equation at $\vec{s} \leftarrow S^{\mu}$ using a polynomial interpretation. Note that for optimization, we use $[\widehat{1}]_T = \iota_T^G([1]_T)(\vec{s}) = [1]_T = e([1]_1, [1]_2)$.

[11] We can use a different choice of $\mathbf{f}^{(i)}$, but this one works well with small exponents.

[12] This depends on the equation types, but it is at most two. For quadratic equations, precomputing $\iota_T(1)$ allows to "replace" the pairing by $n_1 n_2$ exponentiations in G_T.

The formulas for the 2-Lin instantiation are essentially the same, except that the sums range from 1 to 3 and the right-hand side of the batched equation reflects this. For illustration, we show how a batched P-signature proof for a single signature looks like using our technique under the SXDH instantiation of GS proofs. In the notation of (21), with polynomial interpretation being that of Ex. 5.1, the batched equation is, on the left-hand side

$$e(-r_1[1]_1 + r_3[\widehat{x}_2]_1, [1]_2) + e(r_1[\widehat{x}_1]_1, [b_2 + \widehat{y}_1 + \widehat{y}_2]_2) \\ + e(r_3[\widehat{x}_3]_1, [b_3]_2) + e(-[a_2]_1, [r_3\widehat{y}_1 + r_2\widehat{y}_2]_2)$$

and on the right-hand side

$$e([\widehat{u_1}]_1, [r_1\widehat{\pi}_1^{(1)} + r_2\widehat{\pi}_1^{(2)} + r_3\widehat{\pi}_1^{(3)}]_2) + e([\widehat{u_2}]_1, [r_1\widehat{\pi}_2^{(1)} + r_2\widehat{\pi}_2^{(2)} + r_3\widehat{\pi}_2^{(3)}]_2) \\ + e([r_1\widehat{\theta}_1^{(1)} + r_2\widehat{\theta}_1^{(2)} + r_3\widehat{\theta}_1^{(3)}]_1, [\widehat{v}_1]_2) + e([r_1\widehat{\theta}_2^{(1)} + r_2\widehat{\theta}_2^{(2)} + r_3\widehat{\theta}_2^{(3}]_1, [\widehat{v}_2]_2).$$

The batched equation in [11] needs $2N + 11$ pairings to verify N signatures of the same signer. Our technique only needs $N + 7$ pairings. Moreover, we need about $1/3$ fewer exponentiations in G_1 and about $1/2$ fewer in G_2. Sec. 7 contains timings of both verification algorithms running on a modern smartphone. In App. D, we give more details about the formulas and estimates we used.

6 APPLICATIONS TO MATRIX MULTIPLICATION

Our techniques can be applied to do probabilistic checking of matrix multiplication "in the exponent" with fewer pairing operations. Since the introduction of the MDDH framework [25, 44], in which standard decisional assumptions are written in terms of deciding membership in the image of a matrix $\mathbf{A}$, several protocols are written in algebraic terms. For instance, in [41] the verification equation of several constructions of structure preserving signature schemes are written all in terms of vector-matrix operations in the exponent. Other examples using this formulation are recent very efficient constructions of (quasi-adaptive) NIZK proofs of membership in linear spaces [33, 42].

6.1 Batch Verification of Matrix Multiplication

We first show how to interpret a matrix product claim $\mathbf{AB} \overset{?}{=} \mathbf{C}$ as a polynomial identity claim. This follows from a well known result of linear algebra, which writes matrix multiplication as a sum of tensor products.

Lemma 6.1. *Let* $\mathbf{A} \in \mathbb{Z}_p^{\ell \times m}, \mathbf{B} \in \mathbb{Z}_p^{m \times n}$. *For* $i = 1, \ldots, m$, *let* $\vec{a}_i \in \mathbb{Z}_p^{\ell}$ *denote the* i^{th} *column of* $\mathbf{A}$ *and* $\vec{b}_i \in \mathbb{Z}_p^n$ *the transpose of the* i^{th} *row of* $\mathbf{B}$. *Then,* $\mathbf{AB} = \sum_{k=1}^{m} \vec{a}_k \vec{b}_k^{\top}$.

Proof. Let $\mathbf{A} = (a_{ij}), \mathbf{B} = (b_{ij})$. The (i, j)-th coordinate of $\vec{a}_k \vec{b}_k^{\top}$ is $a_{ik}b_{kj}$. Taking the sum over $k = 1, \ldots, m$, we obtain the formula for matrix multiplication. □

As we have seen in Sec. 3, the tensor product, in this case from $\mathbb{Z}_p^{\ell} \times \mathbb{Z}_p^n \to \mathbb{Z}_p^{\ell \times n}$ can be written as polynomial product. Now, we use this view to address the following problem: given matrices $[\mathbf{A}_j] \in G_1^{\ell \times m_j}$, $[\mathbf{B}_j] \in G_2^{m_j \times n}$, $[\mathbf{C}]_T \in G_T^{\ell \times n}$, verify if the claim

$$\sum_{j=1}^{N} \tilde{e}'([\mathbf{A}_j]_1, [\mathbf{B}_j]_2) \overset{?}{=} [\mathbf{C}]_T, \tag{23}$$

holds, where $\tilde{e}'([\mathbf{A}]_1, [\mathbf{B}]_2) = [\mathbf{AB}]_T$. By the previous explanation, this amounts to checking if $\sum_{j=1}^{N} \sum_{i=1}^{m_j} \tilde{e}([\vec{a}_i^j]_1, [\vec{b}_i^j]_1) = [\mathbf{C}]_T$, where $\tilde{e}\colon G_1^{\ell} \times G_2^n \to G_T^{\ell \times n}$ is the tensor product, and $\vec{a}_i^j$ is the i-th column of $\mathbf{A}_j$ and $\vec{b}_i^j$ is the i-th row of $\mathbf{B}_j$. Now, we take some polynomial interpretation of the asymmetric tensor product, for instance, $B_P = (\mathbf{p}_1, \ldots, \mathbf{p}_\ell) = (1, X_2, \ldots, X_\ell)$, $B_Q = (\mathbf{q}_1, \ldots, \mathbf{q}_n) = (1, Y_2, \ldots, Y_n)$ and $\mathbf{r}_{ij} := \mathbf{p}_i\mathbf{q}_j$ as B_R. Thus we interpret (23) as

$$\sum_{j=1}^{N} \sum_{i=1}^{m_j} \tilde{e}([\mathbf{a}_i^j]_1, [\mathbf{b}_i^j]_1) \overset{?}{=} [\mathbf{c}]_T$$

where $\tilde{e}$ is polynomial multiplication in the exponent, and $\mathbf{a}_i^j$, $\mathbf{b}_i^j$ and $\mathbf{c}$, are, respectively, the polynomials associated to $\vec{a}_i^j, \vec{b}_i^j$, $\mathbf{C}$ w.r.t. B_P, B_Q, B_R, e.g. $\mathbf{c} = \sum_{i=1}^{\ell} \sum_{j=1}^{n} [c_{ij}]_T \mathbf{r}_{ij}$.

To batch verify, we evaluate the polynomials at a random point in the set $S^{\ell+n-2}$. The required number of pairing operations is $\sum_{j=1}^{N} m_j$. The number of exponentiations (assuming $\mathbf{p}_1 = \mathbf{q}_1 = \mathbf{1}$) is $(\ell - 1)\sum_{j=1}^{N} m_j$ in G_1, $(n - 1)\sum_{j=1}^{N} m_j$ in G_2 and $\ell n - 1$ in G_T[13]

6.2 Comparison with Previous Work

Probabilistic matrix product verification is a well studied problem. For instance, there is a well known probabilistic matrix verification algorithm due to Freivalds [29]. It checks $\mathbf{AB} = \mathbf{C}$ by testing $\mathbf{AB}\vec{r} = \mathbf{C}\vec{r}$ for $r \leftarrow \{0, 1\}^n$ and has soundness error $1/2$. Note however that the goal of these classical algorithms is to minimize the number of multiplications in the finite field. When we look at the problem in a bilinear group, we want to minimize the number of basic pairing operations, even at the cost of increasing the number of exponentiations. This means we do not care for the total number of multiplications but only those which include multiples of the entries of $\mathbf{A}$ and $\mathbf{B}$.

Note that polynomial evaluation can be written as vector matrix multiplication. Thus, an alternative description of our algorithm is as follows: to verify $\mathbf{AB} \overset{?}{=} \mathbf{C}$, check if $\vec{u}^{\top}\mathbf{AB}\vec{v} \overset{?}{=} \vec{u}^{\top}\mathbf{C}\vec{v}$, for some $\vec{u}, \vec{v}$ sampled from some distribution given by the polynomial interpretation. On the other hand, the standard small exponents batching computes a linear combination of the equations and it is not hard to see that both methods give essentially the same result when $\min(\ell, n) = 1$.

Applying the small exponents test results in $\min(\ell, n)\left(\sum_{j=1}^{N} m_j\right)$ pairing operations (e.g. if $\ell = \min(\ell, n)$ one moves the small exponents as exponentiations in G_2 and groups all terms with the same first argument in $\tilde{e}$). If $\ell = \min(\ell, n)$, the test needs $n\sum_{j=1}^{N} m_j$ exponentiations in G_2 and if $n = \min(\ell, n)$, it needs $\ell\sum_{j=1}^{N} m_j$ exponentiations in G_1.

[13] The sum in G_T should be computed as $\mathbf{c}(\vec{s}) = \sum_{i=1}^{\ell} \mathbf{p}_i(\vec{s})(\sum_{j=1}^{n} [c_{ij}]_T \mathbf{q}_j(\vec{s}))$ according to Horner's rule. This way, all exponentiations in G_T use exponents from S.

6.3 Application Examples

In most application examples $\min(\ell, n) = 1$. For instance, in the structure preserving signature schemes of [41], $\ell = 1$ and $n = k + 1$, where k depends on the security parameter, e.g. $k = 1$ if the construction is based on the SXDH Assumption and $k = 2$ for the Decisional Linear Assumption.[14] This is also the case for the quasi-adaptive NIZK of linear spaces [33, 39, 42]. In all these cases, our batching techniques make the number of pairing computations independent of k.

In these examples, the concrete improvements w.r.t. small exponent batching are small (we save on exponentiations by taking $\mathbf{p}_1 = \mathbf{q}_1 = \mathbf{1}$) but we think it is still interesting to give a general method to batch these equation types, as opposed to manually handcrafting the verification equation. Also, our approach inherits the advantages of the polynomial view analysis regarding the choice of S.

We give an example of concrete improvement from the work of González et al. [33, Sec. 5.1]. Let $\mathbf{I}$ be the identity matrix in $\mathbb{Z}_p^{2\times 2}$. The authors construct a NIZK proof that a set of commitments satisfy a quadratic relation. The verification algorithm checks several linear algebra equations which can be improved with our techniques. Among them, the equation:

$$\tilde{e}([\vec{c}]_1, [\vec{d}]_2) \stackrel{?}{=} \tilde{e}'([\boldsymbol{\theta}]_1, [\mathbf{I}]_2) + \tilde{e}'([\mathbf{I}]_1, [\boldsymbol{\pi}]_2). \tag{24}$$

for vectors $[\vec{c}]_1 \in G_1^2$, $[\vec{d}]_2 \in G_2^2$ and matrices $[\boldsymbol{\theta}]_1 \in G_1^{2\times 2}$, $[\boldsymbol{\pi}]_2 \in G_2^{2\times 2}$. This case is particularly efficient since the discrete logarithm of $\mathbf{I}$ is known, so all the exponentiations needed to evaluate the right-hand side of (24) can be moved into a single argument of $\tilde{e}'$:

$$\begin{aligned} e([c_1{+}r_1c_2]_1, [d_1{+}s_1c_2]_2) \stackrel{?}{=} {} & e([\theta_{11}{+}r_1\theta_{21}{+}s_1\theta_{12}{+}r_1s_1\theta_{22}]_1, [1]_2) \\ & + e([1]_1, [\pi_{11} + r_1\pi_{21} + s_1\pi_{12} + r_1s_1\pi_{22}]_2). \end{aligned}$$

Thus, our techniques require only 3 basic pairings to verify the claim. On the other hand, for the small exponent test, even if using the same trick we require 6 basic pairings.

7 PERFORMANCE EVALUATION

In order to evaluate the benefits of our techniques in a real-world application, we implemented the verification of P-Signatures [6] and of a BeleniosRF vote [19]. We chose a modern smartphone as our target platform to exemplarily evaluate the impact of our techniques. For P-Signatures, we also implemented an optimization proposed by Blazy et al. [11] and compare the results with our novel techniques. In case of the BeleniosRF system, the comparison is more intricate. Chaidos et al. [19] suggest the use of batching techniques and provide the amount of pairings for their batched result; but they do not provide the actual batched equations.

7.1 Implementation Details

We implemented the verification algorithms of all schemes in C using the RELIC toolkit v.0.4.1 [4]. We configured RELIC to utilize 254-bit order Baretto-Naehrig curves Fp254BNb and Fp254n2Bnb by Aranha et al. [40], *optimal Ate parings* for maximum speed, and the left-to-right windowed NAF multiplication.

Operation	Execution Time
Scalar multiplication in G_1 (small exp)	0.73 ms
Scalar multiplication in G_2 (small exp)	1.73 ms
Scalar multiplication in G_1	1.13 ms
Scalar multiplication in G_2	2.41 ms
Pairing evaluation	5.48 ms

Table 2: Average execution times of the basic operations.

Verification Algorithm	Execution Time [ms]
Blazy et al. (original)	167.43
Blazy et al. (optimized)	154.96
This work	105.77
This work ($t_1 = 1$)	100.96

Table 3: Execution times of our implemented P-Signature verification algorithms.

Our target platform is a OnePlus 3 smartphone featuring a Snapdragon 820 Quad-Core CPU (2×2.15 GHz & 2×1.6 GHz) and 6GB RAM, running Android 7.1.1 (Nougat).

Table 2 shows the average execution times of basic operations in the groups G_1 and G_2 and of a pairing computation. Since we use elliptic curves the measured operation in G_1 and G_2 is scalar point multiplication The scalars used for the measurements were random 82-bit for the small exponents (as used for batching) and random 254-bit exponents, indicating the performance gain in small exponents. Note that computing a pairing takes only about the 8-fold time of a *small* scalar multiplication in G_1, and merely the 3-fold time of a *small* scalar multiplication in G_2. Thus, the number of pairings alone is not a good runtime estimate, as a high amount of group operations can outweigh the pairing computations.

Our implementation uses random inputs, since we are only interested in the execution times of the verification algorithms.

7.2 P-Signatures

Table 3 shows the average execution times of the implemented algorithms. P-Signatures optimized with our techniques are in the best case about 40% faster than P-Signatures optimized with the techniques of Blazy et al. Note that we slightly improved Blazy's approach as shown in App. D.2.

7.3 Belenios Receipt Free Voting Scheme

The BeleniosRF system immensely benefits from our new techniques. For details on the batching process, see Appendix E. In Table 4, we show the number of pairings required for different choices of the parameter k as well as the average execution times of the verification algorithm. The percentual improvement is based on the mathematical comparison, i.e. the number of pairings. However, getting comparable execution times for this application is difficult. Simply scaling the average execution time of a single pairing does

[14] To be precise, k depends on the decisional assumption that one uses, following the Matrix Diffie-Hellman framework of [25].

k	1	5	10	25
#pairings ([19])	144	160	180	240
#pairings (this work)	8	12	17	32
Improvement	94.4%	92.5%	90.5%	86.7%
Min. exec. time for [19]	789	876	986	1315
Est. exec. time for [19]	1482	2284	3174	4971
Exec. time (this work)	82.36	171.32	299.82	662.85

Table 4: Comparison of the different versions of the BeleniosRF voting scheme verification algorithm. All execution times are in milliseconds.

not result in a directly comparable timing. As we showed in Section 7.1, the basic group operations are, compared to a pairing, faster by only a small factor, but numerous group operations are required to compute the inputs for the remaining pairing functions.

In addition, Chaidos et al. use GS proofs along with QANIZK proofs, which cost significantly less group operations.[15] Therefore, Table 4 contains the minimum execution time on our platform and an estimation for a more realistic execution time. The minimum execution time is computed by multiplying the plain amount of pairings with the average execution time of a pairing. Note that this is an overly optimistic approximation, since it represents an implementation which does not have to compute anything beyond the pairings themselves. We estimate a more realistic execution time by scaling our execution times according to the percentual improvements. This estimation is still not perfect due to the QANIZK proofs, however, we expect this estimation to be much closer to a realistic value than the lower bound.

Regardless of the aforementioned difficulties, optimizing the BeleniosRF scheme highlights the application-specific improvement potential. We are able to improve from $4k + 140$ to $k + 7$ pairings, a gain of up to 86-95% in execution time regarding the parameter set $k \in \{1, 5, 10, 25\}$.

ACKNOWLEDGMENTS

We would like to thank the authors of BeleniosRF for providing information about their proof and batching techniques. We also would like to thank the anonymous reviewers for their constructive comments.

REFERENCES

[1] Masayuki Abe, Georg Fuchsbauer, Jens Groth, Kristiyan Haralambiev, and Miyako Ohkubo. 2010. Structure-Preserving Signatures and Commitments to Group Elements. In *CRYPTO 2010 (LNCS)*, Tal Rabin (Ed.), Vol. 6223. Springer, Heidelberg, Germany, Santa Barbara, CA, USA, 209–236.

[2] Joseph A. Akinyele, Gilles Barthe, Benjamin Grégoire, Benedikt Schmidt, and Pierre-Yves Strub. 2014. Certified Synthesis of Efficient Batch Verifiers. (2014), 153–165.

[3] Joseph A. Akinyele, Matthew Green, Susan Hohenberger, and Matthew W. Pagano. 2012. Machine-generated algorithms, proofs and software for the batch verification of digital signature schemes. In *the ACM Conference on Computer and Communications Security, CCS'12, Raleigh, NC, USA, October 16-18, 2012*, Ting Yu, George Danezis, and Virgil D. Gligor (Eds.). ACM, 474–487. https://doi.org/10.1145/2382196.2382248

[4] D. F. Aranha and C. P. L. Gouvêa. [n. d.]. RELIC is an Efficient LIbrary for Cryptography. https://github.com/relic-toolkit/relic. ([n. d.]).

[5] Mira Belenkiy, Jan Camenisch, Melissa Chase, Markulf Kohlweiss, Anna Lysyanskaya, and Hovav Shacham. 2009. Randomizable Proofs and Delegatable Anonymous Credentials. In *CRYPTO 2009 (LNCS)*, Shai Halevi (Ed.), Vol. 5677. Springer, Heidelberg, Germany, Santa Barbara, CA, USA, 108–125.

[6] Mira Belenkiy, Melissa Chase, Markulf Kohlweiss, and Anna Lysyanskaya. 2008. P-signatures and Noninteractive Anonymous Credentials. In *TCC 2008 (LNCS)*, Ran Canetti (Ed.), Vol. 4948. Springer, Heidelberg, Germany, San Francisco, CA, USA, 356–374.

[7] Mira Belenkiy, Melissa Chase, Markulf Kohlweiss, and Anna Lysyanskaya. 2009. Compact E-Cash and Simulatable VRFs Revisited. In *PAIRING 2009 (LNCS)*, Hovav Shacham and Brent Waters (Eds.), Vol. 5671. Springer, Heidelberg, Germany, Palo Alto, CA, USA, 114–131.

[8] Mihir Bellare, Juan A. Garay, and Tal Rabin. 1998. Fast Batch Verification for Modular Exponentiation and Digital Signatures. In *EUROCRYPT'98 (LNCS)*, Kaisa Nyberg (Ed.), Vol. 1403. Springer, Heidelberg, Germany, Espoo, Finland, 236–250.

[9] John Bethencourt, Elaine Shi, and Dawn Song. 2010. Signatures of Reputation. In *FC 2010 (LNCS)*, Radu Sion (Ed.), Vol. 6052. Springer, Heidelberg, Germany, Tenerife, Canary Islands, Spain, 400–407.

[10] Nir Bitansky, Ran Canetti, Alessandro Chiesa, Shafi Goldwasser, Huijia Lin, Aviad Rubinstein, and Eran Tromer. 2014. The Hunting of the SNARK. *IACR Cryptology ePrint Archive* 2014 (2014), 580. http://eprint.iacr.org/2014/580

[11] Olivier Blazy, Georg Fuchsbauer, Malika Izabachène, Amandine Jambert, Hervé Sibert, and Damien Vergnaud. 2010. Batch Groth-Sahai. In *ACNS 10 (LNCS)*, Jianying Zhou and Moti Yung (Eds.), Vol. 6123. Springer, Heidelberg, Germany, Beijing, China, 218–235.

[12] Dan Boneh, Xavier Boyen, and Hovav Shacham. 2004. Short Group Signatures. In *CRYPTO 2004 (LNCS)*, Matthew Franklin (Ed.), Vol. 3152. Springer, Heidelberg, Germany, Santa Barbara, CA, USA, 41–55.

[13] Dan Boneh and Matthew K. Franklin. 2001. Identity-Based Encryption from the Weil Pairing. In *CRYPTO 2001 (LNCS)*, Joe Kilian (Ed.), Vol. 2139. Springer, Heidelberg, Germany, Santa Barbara, CA, USA, 213–229.

[14] Dan Boneh, Craig Gentry, Ben Lynn, and Hovav Shacham. 2003. Aggregate and Verifiably Encrypted Signatures from Bilinear Maps. In *EUROCRYPT 2003 (LNCS)*, Eli Biham (Ed.), Vol. 2656. Springer, Heidelberg, Germany, Warsaw, Poland, 416–432.

[15] Jan Camenisch, Maria Dubovitskaya, Kristiyan Haralambiev, and Markulf Kohlweiss. 2015. Composable and Modular Anonymous Credentials: Definitions and Practical Constructions. In *Advances in Cryptology - ASIACRYPT 2015 - 21st International Conference on the Theory and Application of Cryptology and Information Security, Auckland, New Zealand, November 29 - December 3, 2015, Proceedings, Part II (Lecture Notes in Computer Science)*, Tetsu Iwata and Jung Hee Cheon (Eds.), Vol. 9453. Springer, 262–288. https://doi.org/10.1007/978-3-662-48800-3_11

[16] Jan Camenisch, Susan Hohenberger, and Anna Lysyanskaya. 2005. Compact E-Cash. In *EUROCRYPT 2005 (LNCS)*, Ronald Cramer (Ed.), Vol. 3494. Springer, Heidelberg, Germany, Aarhus, Denmark, 302–321.

[17] Jan Camenisch, Susan Hohenberger, and Michael Østergaard Pedersen. 2007. Batch Verification of Short Signatures. In *EUROCRYPT 2007 (LNCS)*, Moni Naor (Ed.), Vol. 4515. Springer, Heidelberg, Germany, Barcelona, Spain, 246–263.

[18] Jan Camenisch and Anna Lysyanskaya. 2004. Signature Schemes and Anonymous Credentials from Bilinear Maps. In *CRYPTO 2004 (LNCS)*, Matthew Franklin (Ed.), Vol. 3152. Springer, Heidelberg, Germany, Santa Barbara, CA, USA, 56–72.

[19] Pyrros Chaidos, Véronique Cortier, Georg Fuchsbauer, and David Galindo. 2016. BeleniosRF: A Non-interactive Receipt-Free Electronic Voting Scheme. In *ACM CCS 16*. ACM Press, 1614–1625.

[20] Nishanth Chandran, Jens Groth, and Amit Sahai. 2007. Ring Signatures of Sublinear Size Without Random Oracles. In *ICALP 2007 (LNCS)*, Lars Arge, Christian Cachin, Tomasz Jurdzinski, and Andrzej Tarlecki (Eds.), Vol. 4596. Springer, Heidelberg, Germany, Wroclaw, Poland, 423–434.

[21] Melissa Chase, Markulf Kohlweiss, Anna Lysyanskaya, and Sarah Meiklejohn. 2012. Malleable Proof Systems and Applications. In *EUROCRYPT 2012 (LNCS)*, David Pointcheval and Thomas Johansson (Eds.), Vol. 7237. Springer, Heidelberg, Germany, Cambridge, UK, 281–300.

[22] Melissa Chase, Markulf Kohlweiss, Anna Lysyanskaya, and Sarah Meiklejohn. 2013. Verifiable Elections That Scale for Free. In *PKC 2013 (LNCS)*, Kaoru Kurosawa and Goichiro Hanaoka (Eds.), Vol. 7778. Springer, Heidelberg, Germany, Nara, Japan, 479–496. https://doi.org/10.1007/978-3-642-36362-7_29

[23] Jung Hee Cheon and Dong Hoon Lee. 2006. Use of Sparse and/or Complex Exponents in Batch Verification of Exponentiations. *IEEE Trans. Comput.* 55, 12 (Dec 2006), 1536–1542.

[24] Alex Escala and Jens Groth. 2014. Fine-Tuning Groth-Sahai Proofs. In *PKC 2014 (LNCS)*, Hugo Krawczyk (Ed.), Vol. 8383. Springer, Heidelberg, Germany, Buenos Aires, Argentina, 630–649. https://doi.org/10.1007/978-3-642-54631-0_36

[25] Alex Escala, Gottfried Herold, Eike Kiltz, Carla Ràfols, and Jorge Villar. 2013. An Algebraic Framework for Diffie-Hellman Assumptions. In *CRYPTO 2013, Part II (LNCS)*, Ran Canetti and Juan A. Garay (Eds.), Vol. 8043. Springer, Heidelberg, Germany, Santa Barbara, CA, USA, 129–147. https://doi.org/10.1007/

[15] See App. E.5 for details.

978-3-642-40084-1_8

[26] Anna Lisa Ferrara, Matthew Green, Susan Hohenberger, and Michael Østergaard Pedersen. 2009. Practical Short Signature Batch Verification. In *CT-RSA 2009 (LNCS)*, Marc Fischlin (Ed.), Vol. 5473. Springer, Heidelberg, Germany, San Francisco, CA, USA, 309–324.

[27] Amos Fiat and Adi Shamir. 1987. How to Prove Yourself: Practical Solutions to Identification and Signature Problems. In *CRYPTO'86 (LNCS)*, Andrew M. Odlyzko (Ed.), Vol. 263. Springer, Heidelberg, Germany, Santa Barbara, CA, USA, 186–194.

[28] David Mandell Freeman. 2010. Converting Pairing-Based Cryptosystems from Composite-Order Groups to Prime-Order Groups. In *EUROCRYPT 2010 (LNCS)*, Henri Gilbert (Ed.), Vol. 6110. Springer, Heidelberg, Germany, French Riviera, 44–61.

[29] Rusins Freivalds. 1977. Probabilistic Machines Can Use Less Running Time.. In *IFIP congress*, Vol. 839. 842.

[30] Georg Fuchsbauer, David Pointcheval, and Damien Vergnaud. 2009. Transferable Constant-Size Fair E-Cash. In *CANS 09 (LNCS)*, Juan A. Garay, Atsuko Miyaji, and Akira Otsuka (Eds.), Vol. 5888. Springer, Heidelberg, Germany, Kanazawa, Japan, 226–247.

[31] Jun Furukawa and Hideki Imai. 2005. An Efficient Group Signature Scheme from Bilinear Maps. In *ACISP 05 (LNCS)*, Colin Boyd and Juan Manuel González Nieto (Eds.), Vol. 3574. Springer, Heidelberg, Germany, Brisbane, Queensland, Australia, 455–467.

[32] S. D. Galbraith, K. G. Paterson, and N. P. Smart. 2008. Pairings for cryptographers. *Discrete Applied Mathematics* 156(16) (2008), 3113?3121.

[33] Alonso González, Alejandro Hevia, and Carla Ràfols. 2015. QA-NIZK Arguments in Asymmetric Groups: New Tools and New Constructions. In *ASIACRYPT 2015, Part I (LNCS)*, Tetsu Iwata and Jung Hee Cheon (Eds.), Vol. 9452. Springer, Heidelberg, Germany, Auckland, New Zealand, 605–629. https://doi.org/10.1007/978-3-662-48797-6_25

[34] Jens Groth and Amit Sahai. 2008. Efficient Non-interactive Proof Systems for Bilinear Groups. In *EUROCRYPT 2008 (LNCS)*, Nigel P. Smart (Ed.), Vol. 4965. Springer, Heidelberg, Germany, Istanbul, Turkey, 415–432.

[35] Jens Groth and Amit Sahai. 2012. Efficient Noninteractive Proof Systems for Bilinear Groups. *SIAM J. Comput.* 41, 5 (2012), 1193–1232.

[36] Gottfried Herold, Julia Hesse, Dennis Hofheinz, Carla Ràfols, and Andy Rupp. 2014. Polynomial Spaces: A New Framework for Composite-to-Prime-Order Transformations. In *CRYPTO 2014, Part I (LNCS)*, Juan A. Garay and Rosario Gennaro (Eds.), Vol. 8616. Springer, Heidelberg, Germany, Santa Barbara, CA, USA, 261–279. https://doi.org/10.1007/978-3-662-44371-2_15

[37] Tibor Jager and Andy Rupp. 2016. Black-Box Accumulation: Collecting Incentives in a Privacy-Preserving Way. *PoPETs* 2016, 3 (2016), 62–82. http://www.degruyter.com/view/j/popets.2016.2016.issue-3/popets-2016-0016/popets-2016-0016.xml

[38] Antoine Joux. 2004. A One Round Protocol for Tripartite Diffie-Hellman. *Journal of Cryptology* 17, 4 (Sept. 2004), 263–276.

[39] Charanjit S. Jutla and Arnab Roy. 2013. Shorter Quasi-Adaptive NIZK Proofs for Linear Subspaces. In *ASIACRYPT 2013, Part I (LNCS)*, Kazue Sako and Palash Sarkar (Eds.), Vol. 8269. Springer, Heidelberg, Germany, Bengalore, India, 1–20. https://doi.org/10.1007/978-3-642-42033-7_1

[40] Yuto Kawahara, Tetsutaro Kobayashi, Michael Scott, and Akihiro Kato. 2016. *Barreto-Naehrig Curves.* Internet-Draft draft-kasamatsu-bncurves-02. Internet Engineering Task Force. https://datatracker.ietf.org/doc/html/draft-kasamatsu-bncurves-02 Work in Progress.

[41] Eike Kiltz, Jiaxin Pan, and Hoeteck Wee. 2015. Structure-Preserving Signatures from Standard Assumptions, Revisited. In *CRYPTO 2015, Part II (LNCS)*, Rosario Gennaro and Matthew J. B. Robshaw (Eds.), Vol. 9216. Springer, Heidelberg, Germany, Santa Barbara, CA, USA, 275–295. https://doi.org/10.1007/978-3-662-48000-7_14

[42] Eike Kiltz and Hoeteck Wee. 2015. Quasi-Adaptive NIZK for Linear Subspaces Revisited. In *EUROCRYPT 2015, Part II (LNCS)*, Elisabeth Oswald and Marc Fischlin (Eds.), Vol. 9057. Springer, Heidelberg, Germany, Sofia, Bulgaria, 101–128. https://doi.org/10.1007/978-3-662-46803-6_4

[43] Bodo Möller. 2001. *Algorithms for Multi-exponentiation.* Springer Berlin Heidelberg, Berlin, Heidelberg, 165–180. https://doi.org/10.1007/3-540-45537-X_13

[44] Paz Morillo, Carla Ràfols, and Jorge Luis Villar. 2016. The Kernel Matrix Diffie-Hellman Assumption. In *ASIACRYPT 2016, Part I (LNCS).* Springer, Heidelberg, Germany, 729–758. https://doi.org/10.1007/978-3-662-53887-6_27

[45] Lan Nguyen. 2006. Privacy-Protecting Coupon System Revisited. In *Financial Cryptography and Data Security, 10th International Conference, FC 2006, Anguilla, British West Indies, February 27-March 2, 2006, Revised Selected Papers (Lecture Notes in Computer Science)*, Giovanni Di Crescenzo and Aviel D. Rubin (Eds.), Vol. 4107. Springer, 266–280. https://doi.org/10.1007/11889663_22

[46] Kenneth G. Paterson and Jacob C. N. Schuldt. 2006. Efficient Identity-Based Signatures Secure in the Standard Model. In *ACISP 06 (LNCS)*, Lynn Margaret Batten and Reihaneh Safavi-Naini (Eds.), Vol. 4058. Springer, Heidelberg, Germany, Melbourne, Australia, 207–222.

[47] J. T. Schwartz. 1980. Fast Probabilistic Algorithms for Verification of Polynomial Identities. *J. ACM* 27, 4 (Oct. 1980), 701–717. https://doi.org/10.1145/322217.322225

[48] Jae Hong Seo. 2012. On the (Im)possibility of Projecting Property in Prime-Order Setting. In *ASIACRYPT 2012 (LNCS)*, Xiaoyun Wang and Kazue Sako (Eds.), Vol. 7658. Springer, Heidelberg, Germany, Beijing, China, 61–79. https://doi.org/10.1007/978-3-642-34961-4_6

[49] Hovav Shacham. 2007. A Cramer-Shoup Encryption Scheme from the Linear Assumption and from Progressively Weaker Linear Variants. Cryptology ePrint Archive, Report 2007/074. (2007). http://eprint.iacr.org/2007/074.

[50] Richard Zippel. 1979. Probabilistic algorithms for sparse polynomials. In *Symbolic and Algebraic Computation, EUROSAM '79, An International Symposiumon Symbolic and Algebraic Computation, Marseille, France, June 1979, Proceedings (Lecture Notes in Computer Science)*, Edward W. Ng (Ed.), Vol. 72. Springer, 216–226. https://doi.org/10.1007/3-540-09519-5_73

APPENDIX

A COMPUTATIONAL ASSUMPTIONS

As we mentioned, our improvements in batch verification are unconditional and depend only indirectly of the hardness of some cryptographic problem. However, we will be referring to different computational assumptions in bilinear groups when describing the different instantiations of Groth-Sahai NIZK proof system and we list them here for completeness.

The DDH and 2-Lin Assumptions were generalized ([12, 49]) to the k-linear family of assumptions, which is a family of increasingly weaker assumptions (where the two previous assumptions corresponded to $k = 1, 2$. A generalization of these assumptions was introduced by Escala et al. [25]. They propose a family of assumptions parameterized by a matrix distribution $\mathcal{D}_k$.

Definition A.1. We call $\mathcal{D}_k$ a matrix distribution if it outputs a matrix $\mathbf{A} \in \mathbb{Z}_p^{(k+1)\times k}$ such that any $k \times k$ submatrix of $\mathbf{A}$ has rank k. (in poly time, with overwhelming probability) matrices in $\mathbb{Z}_p^{(k+1)\times k}$ of full rank k.

We define the $\mathcal{D}_k$-matrix problem as to distinguish the two distributions $([\mathbf{A}], [\mathbf{A}\vec{w}])$ and $([\mathbf{A}], [\vec{z}])$, where $\mathbf{A} \leftarrow \mathcal{D}_k$, $\vec{w} \leftarrow \mathbb{Z}_p^k$, and $\vec{u} \leftarrow \mathbb{Z}_p^{k+1}$. In other words, the assumption states that it is hard to decide membership in the image of $[\mathbf{A}]$.

Definition A.2 ($\mathcal{D}_k$-Matrix Diffie-Hellman Assumption). Let $\mathcal{D}_k$ be a matrix distribution. We say that the $\mathcal{D}_k$-Matrix Diffie-Hellman ($\mathcal{D}_k$-MDDH) Assumption holds relative to Gen if for all PPT adversaries D,

$$\mathcal{A}_{\mathcal{D}_k,\mathrm{Gen}}(\mathrm{D}) = \Pr[\mathrm{D}(\mathcal{G}, [\mathbf{A}], [\mathbf{A}\boldsymbol{w}]) = 1] - \Pr[\mathrm{D}(\mathcal{G}, [\mathbf{A}], [\boldsymbol{u}]) = 1]$$

is negligible, where the probability is taken over $\mathcal{G} = (G, q, \mathcal{P}) \leftarrow \mathrm{Gen}(1^\lambda)$, $\mathbf{A} \leftarrow \mathcal{D}_k, \vec{w} \leftarrow \mathbb{Z}_p^k, \vec{u} \leftarrow \mathbb{Z}_p^{k+1}$ and the coin tosses of adversary D.

In particular, we will use or make mention of the following examples:

$$\mathcal{L}_k : \mathbf{A} = \begin{pmatrix} a_1 & 0 & 0 & \dots & 0 \\ 0 & a_2 & 0 & \dots & 0 \\ \vdots & \vdots & \ddots & & \vdots \\ 0 & 0 & 0 & \dots & a_k \\ 1 & 1 & 1 & \dots & 1 \end{pmatrix} \quad \mathcal{U}_k : \mathbf{A} = \begin{pmatrix} a_{1,1} & \dots & a_{1,k} \\ \vdots & & \vdots \\ a_{k+1,1} & \dots & a_{k+1,k} \end{pmatrix}$$

$$\mathcal{SC}_k : \mathbf{A} = \begin{pmatrix} a & 0 & 0 & \dots & 0 \\ 1 & a & 0 & \dots & 0 \\ \vdots & \vdots & \ddots & & \vdots \\ 0 & 0 & 0 & \dots & a \\ 0 & 0 & 0 & \dots & 1 \end{pmatrix},$$

where $a_i, a_{ij} \leftarrow \mathbb{Z}_p$. The $\mathcal{L}_k$-MDDH Assumption is the k-Lin family assumptions which we introduced for $k = 1, 2$, the second one is the $\mathcal{U}_k$-MDDH or Uniform Assumption, and the last one is the *Symmetric Cascade* Assumption introduced in [25].

Matrix Diffie-Hellman Assumptions are a practical framework for designing protocols, but for practical choices one usually chooses in asymmetric groups the SXDH Assumption (which is the assumption that the DDH Assumption holds in the groups G_1 and G_2 for a bilinear group) and in symmetric bilinear groups one usually chooses the 2-Lin-Assumption.

B A POLYNOMIAL VIEWPOINT FOR ANY GS INSTANTIATION

Herold et al. [36] introduced the polynomial view to derive instantiations of GS proofs with a more efficient map $\tilde{e}$, and they introduced new instantiations of GS proofs which were already given in the polynomial language. In this section, we show that, for all instantiations of GS proofs in the literature, the pairing $\tilde{e}\colon G_1^{n_1} \times G_2^{n_2} \to G_T^{n_T}$ can be interpreted as polynomial multiplication.

Groth and Sahai [35] originally proposed two prime order instantiations, one in asymmetric bilinear groups based on the SXDH Assumption and the other one in symmetric groups under the decisional linear (or 2-Lin) assumption. These were generalized to different assumptions in bilinear groups and different bilinear maps $\tilde{e}$ in several works [25, 48]. In Table 5 below we give a list of all the instantiations of GS proofs given in the cryptographic literature. The table specifies the dimensions of the commitment spaces n_1, n_2, the decisional assumption they are based on and the choice of bilinear map $e\colon G_1^{n_1} \times G_2^{n_2} \to G_T^{n_T}$.

The tensor product and the symmetric tensor product have already been discussed in Examples 3.2 and 3.3. The last two maps were already given in polynomial language. On the other hand, $\tilde{e}([\boldsymbol{x}], [\boldsymbol{y}]) = [\frac{1}{2}(\boldsymbol{x}\boldsymbol{y}^\top + \boldsymbol{y}\boldsymbol{x}^\top)]_T$ is like the symmetric tensor product except that the image of the map $\tilde{e}$ does not span $G_T^{n_T}$. However, this just means that some elements in the image always appear twice and are redundant. After we remove them (as is allowed by our definition of polynomial interpretations), this is exactly the symmetric tensor product again with $B_P = B_Q = \{\mathbf{p}_1 = \mathbf{1}, \mathbf{p}_2 = X_1, \mathbf{p}_3 = X_2\}$ and $B_R \coloneqq \{\mathbf{r}_{ij} = \mathbf{p}_i\mathbf{p}_j\}$ (in which case, $\mathbf{r}_{ij} = \mathbf{r}_{ji}$).

As we saw in Sec. 3, the choice of polynomial view affects the number of exponentiations necessary to compute batching. We note that for the maps of [36] given in polynomial language, the choices of B_P, B_Q and B_R are not always the most efficient for our applications. For instance, for the 2-Lin Assumption, [36] sets $B_P = B_Q = \{-X_2, -X_1, X_1X_2\}$, but an alternative and more efficient polynomial view compatible with this assumption is the usual choice for the symmetric tensor product, namely, $B_P = B_Q = \{\mathbf{1}, X_1, X_2\}$.

C COMPARISON WITH BLAZY et al.

In this section, we compare batching GS proofs as in [11] with our technique. We use the notation of Sec. 5.2. Instead of considering all strategies of Blazy et al., we restrict to the strategy which only uses exponentiations in G_1.[16]

C.1 SXDH instantiation

In this section, we consider the SXDH instantiation of GS proofs from Sec. 2.6.4. Our notation is as in Section 2.6. A commitment is an element $[(c_1, c_2)]_1 \in G_1^2$, so we view $[\overline{x}_i]_1$ lying in G_1^2. and $[\overline{y}_j]_2 \in G_2^2$. In [11] the elements in $G_T^{2\times 2}$ are treated as mere vectors and they do not have a polynomial interpretation.

Their batched term for the left-hand side of Eq. (3) for a pairing product equation is

$$\sum_{j=1}^{m_y} e\Bigg(\sum_{k=1}^{2}\Bigg(r_{k,1}(\sum_{i=1}^{m_x}\gamma_{i,j}[\overline{x}_{i,k}]_1) + [\overline{a}_{i,k}]_1, [\overline{y}_{j,1}]_2\Bigg)\Bigg)$$
$$+ \sum_{j=1}^{m_y} e\Bigg(\sum_{k=1}^{2}\Bigg(r_{k,1}(\sum_{i=1}^{m_x}\gamma_{i,j}[\overline{x}_{i,k}]_1) + [\overline{a}_{i,k}]_1, [\overline{y}_{j,2}]_2\Bigg)\Bigg)$$
$$+ \sum_{i=1}^{m_x} e\Bigg(\sum_{k=1}^{2} r_{k,2}[\overline{x}_{i,k}]_1, [\overline{b}_{i,2}]_2\Bigg).$$

Note that $[\overline{a}_{i,2}]_1$ and $[\overline{b}_{i,2}]_2$ are zero, see Ex. 5.1.[17] Thus, the left-hand side needs $m_x + 2m_y$ pairings for "internal" batching à la [11], while we need $m_x + m_y$. The numbers and expressions are different for multiscalar and quadratic equations, since Blazy et al. [11] use knowledge of discrete logarithms as noted in Sec. 2.6.4.

For our batching, we consider the (internally batched) equation

$$\sum_{j=1}^{m_y} e\Bigg([\widehat{a}_i]_1 + \sum_{i=1}^{m_x}[\gamma_{ij}\widehat{x}_i]_1, [\widehat{y}_j]_2\Bigg) + \sum_{i=1}^{m_x} e([\widehat{x}_i]_1, [\widehat{b}_i]_2)$$
$$\overset{?}{=} [\widehat{t}]_T + \sum_{i=1}^{k_1} e([\widehat{u}_i]_1, [\widehat{\pi}_i]_2) + \sum_{i=1}^{k_2} e([\widehat{\theta}_i]_1, [\widehat{v}_i]_2) \qquad (25)$$

as our evaluation strategy. We summarize the cost estimates in Table 6. Note that different evaluation strategies yield different results.

The numbers in Table 6 consist of following parts.

- The costs for computing the intermediates, i.e. evaluating at $\vec{s} \leftarrow S^2$. This is 1 exponentiation (in G_1 or G_2) per evaluation. Note that constants in G_1 (or G_2) cost nothing, and for constants in $\mathbb{Z}_p$ we only need $(\iota_i^Z(1))(\vec{s})$, see Ex. 5.1.
- The costs for computing the right-hand side of Eq. (25)
- The costs for computing the left-hand side of Eq. (25) which is $|\boldsymbol{\Gamma}| + m_x$.
- If $A_1 = \mathbb{Z}_p$, then Eq. (25) needs additional m_x exponentiations to compute $\iota_1^Z(a_i)(\vec{s}) = a_i\iota_1^Z(1)(\vec{s})$.[18]

Note that we estimate costs concerning $\boldsymbol{\Gamma}$ very precisely, and do not count zeroes as exponentiations. However, for a_i and b_j we use rough worst-case estimates.

C.2 2-Lin instantiation

Now we consider the *symmetric* 2-Lin instantiation with symmetric pairing $\tilde{e}$. We suppose that $k_1 \geq k_2$ and $\vec{u}_i = \vec{v}_i$ for $i = 1, \ldots, k_2$, and we use the (shorter) symmetric proof. The comparison of [11]

[16] There are cases where Blazy et al. save pairings by exponentiating in G_2. Essentially, this is done by grouping all pairings with x_i first (contrary to Eq. (25)). Thus our techniques are also improved in such cases. Note that, according to the timings in Sec. 7, Table 2, changing $5\frac{1}{2}$ exponentiations from G_1 to G_2 is as expensive as a pairing.

[17] Blazy et al. [11] use the map $\iota_i^G([a]) = [a\vec{e}_2]$ instead of $[a\vec{e}_1]$. But the influence on the formulas is trivial and estimates are unchanged.

[18] Actually, we could save one G_1-exponentiation by computing $\iota_1^Z(a_i)(\vec{s})$ directly (without $\iota_1^Z(1)(\vec{s})$).

Scheme	Bilinear Group Type and Decisional Assumption	n_1, n_2, n_T	Bilinear Map
[GS08] [34]	Type III SXDH	$n_1 = n_2 = 2$ $n_T = 4$	Tensor Product $\tilde{e}([\vec{x}]_1, [\vec{y}]_1) = [\vec{x}\vec{y}^\top]_T$
[GS08] [34]	Type I 2-Lin	$n_1 = n_2 = 3$ $n_T = 9, n'_T = 6$	$\tilde{e}([\vec{x}], [\vec{y}]) = [\frac{1}{2}(\vec{x}\vec{y}^\top + \vec{y}\vec{x}^\top)]_T$
[EHKRV13] [25]	Type I $\mathcal{D}_k$-MDDH	$n_1 = n_2 = k + 1$ $n_T = (k+1)^2, n'_T = \frac{k(k+1)}{2}$	$\tilde{e}([\vec{x}], [\vec{y}]) = [\frac{1}{2}(\vec{x}\vec{y}^\top + \vec{y}\vec{x}^\top)]_T$
[Seo12] [48]	Type I $\mathcal{U}_2$-MDDH	$n_1 = n_2 = 3$ $n_T = 6$	Symmetric Tensor Product
[HHHRR14] [36]	Type I $\mathcal{S}_2$-MDDH	$n_1 = n_2 = 3, n_T = 5$	Polynomial Multiplication $B_P = B_Q = \{\mathbf{1}, X, X^2\}$
[HHHRR14] [36]	Type I $\mathcal{D}_k$-MDDH	$n_1 = n_2 = k + 1,$ n_T (depending on $\mathcal{D}_k$)	Polynomial Multiplication $B_P = B_Q$ (depending on $\mathcal{D}_k$)

Table 5: List of GS instantiations in the cryptographic literature. In the new instantiations in [36] the pairing $\tilde{e}$ is already expressed as polynomial multiplication. n'_T denotes the dimension of the span of the image if $\tilde{e}$ is not surjective.

Strategy	Type	G_1	G_2	e_G
[11]	PPE	$2\lvert\mathbf{\Gamma}\rvert + 2m_x + 4m_y$ (+16)	0	$m_x + 2m_y$ (+8)
	ME$_1$	$2\lvert\mathbf{\Gamma}\rvert + 2m_x + 4m_y + 2$ (+14)	0	$2m_y + 2$ (+7)
	QE	$2\lvert\mathbf{\Gamma}\rvert + 2m_x + 6m_y + 4$ (+12)	0	$2m_y + 2$ (+6)
ours	PPE	$\lvert\mathbf{\Gamma}\rvert + m_x + m_y$ (+4)	m_y (+4)	$m_x + m_y$ (+4)
	ME$_1$	$\lvert\mathbf{\Gamma}\rvert + 2m_x$ (+2)	$m_y + 1$ (+4)	$m_y + 1$ (+3 + 1)
	QE	$\lvert\mathbf{\Gamma}\rvert + 3m_x + 1$ (+2 + 1)	$m_y + 1$ (+2)	$m_y + 1$ (+2 + 1)
Difference: [11] − ours	PPE	$\lvert\mathbf{\Gamma}\rvert + m_x + 3m_y$ (+12)	$-[m_y - 1$ (+4)$]$	m_y (+4)
	ME$_1$	$\lvert\mathbf{\Gamma}\rvert + 4m_y + 2$ (+8)	$-[m_y - 1$ (+4)$]$	$m_y + 1$ (+4)
	QE	$\lvert\mathbf{\Gamma}\rvert - m_x + 6m_y + 2$ (+7)	$-[m_y - 1$ (+2)$]$	$m_y + 1$ (+3)

Table 6: Comparison of [11] with our techniques for the SXDH instantiation. The constant terms in parenthesis are for the batched Groth–Sahai verification equation's right-hand side. We compare pairing product equations (PPE), multiscalar equations in G_1 (ME$_1$), and quadratic equations in $\mathbb{Z}_p$ (QE). Using a different evaluation for QEs, we can get $|\mathbf{\Gamma}| + 2m_x + m_y$ exponentiations in G_1. The "$(\ldots + 1)$" in the pairing column indicates that for $[\bar{t}]_T = \iota_T(t)$, we need to compute a pairing. If $[\bar{t}]_T$ is stored, then "only" exponentiations in G_T are necessary, which however are not small anymore since they correspond to $\mathrm{r}_1(\vec{s}), \ldots, \mathrm{r}_{n_T}(\vec{s})$. So depending on the instantiation, computing a pairing may be more efficient.

with our techniques is in Table 7. Note that for pairing product equations and quadratic equations, we have $m = m_x = m_y$.

From Tables 6 and 7, one property becomes visible: The "larger" $(G_1^{n_1}, G_2^{n_2}, G_T^{n_T}, \tilde{e})$ is, the higher the penalty for not respecting the pairing, as the factor in front of $|\Gamma|$ shows. Basically, the batching of [11] needs roughly $\min(n_1, n_2)$ pairings for every $\tilde{e}$-pairing between variables. This is where the $2m_y$ term in $m_x + 2m_y + 8$ comes from (for SXDH). For pairings between constants and variables, Blazy et al. use optimizations, which explains the term m_x (instead of $2m_x$) for pairing product equations.

C.3 The impact of structure: An example

In this section, we reconsider the pairing product equation from the introduction, namely

$$\begin{aligned} e([x_1]_1, [y_1]_2) &\overset{?}{=} [t_1]_T \qquad & e([x_1]_1, [y_2]_2) &\overset{?}{=} [t_2]_T \\ e([x_2]_1, [y_1]_2) &\overset{?}{=} [t_3]_T \qquad & e([x_2]_1, [y_2]_2) &\overset{?}{=} [t_4]_T \,. \end{aligned} \tag{26}$$

But instead of verifying equations (26), we assume that we want to verify a GS proof of it. We will see that our techniques are applicable twice, since our GS batching preserves structure. We compare this to standard small exponents batching of GS proofs (as employed in [11]).

Now suppose we have GS proofs proving that the openings $[x_1]_1$, $[x_2]_1$, $[y_1]_2$, $[y_2]_2$ of the respective commitments $[\bar{x}_1]_1, [\bar{x}_2]_1 \in G_1^2$ and $[\bar{y}_1]_2, [\bar{y}_2]_2 \in G_2^2$ satisfy equation (26). Since, as noted in Sec. 5.1, the batched right-hand side (RHS) has constant costs (i.e. a

Strategy	Type	G	e_G
[11] (★)	PPE	$3\lvert\boldsymbol{\Gamma}\rvert + 3m$ (+20)	$3m$ (+6)
	ME$_2$	$3\lvert\boldsymbol{\Gamma}\rvert + 3m_x + 9m_y$ (+18)	$3m_x + 3m_y$ (+6)
	QE	$3\lvert\boldsymbol{\Gamma}\rvert + 9m$ (+18)	$3m$ (+6)
ours	PPE	$\lvert\boldsymbol{\Gamma}\rvert + 2m$ (+12)	$2m$ (+3)
	ME$_2$	$\lvert\boldsymbol{\Gamma}\rvert + 2m_x + 3m_y + 2$ (+12)	$m_y + 1$ (+3 + 1)
	QE	$\lvert\boldsymbol{\Gamma}\rvert + 3m + 2$ (+10 + 1)	$m + 1$ (+2 + 1)
Difference: [11] − ours	PPE	$2\lvert\boldsymbol{\Gamma}\rvert + m$ (+8)	m (+4)
	ME$_2$	$2\lvert\boldsymbol{\Gamma}\rvert + m_x + 6m_y - 2$ (+8)	$3m_x + 2m_y - 1$ (+4)
	QE	$2\lvert\boldsymbol{\Gamma}\rvert + 6m - 2$ (+7)	$2m - 1$ (+3)

Table 7: Comparison of [11] and our techniques for the 2-Lin instantiation. The notation is as in Table 7. The number of exponentiations not given in [11] were counted by us.

constant number of pairings), independent of the number of batched equations,[19] we will only consider the left-hand side (LHS).

In this example, our batching techniques can be applied twice: First, we apply internal batching, as in Sec. 5. This yields intermediate variables $[\widehat{x}_1]_1, [\widehat{x}_2]_2 \in G_1$, and $[\widehat{y}_1]_2, [\widehat{y}_2]_2 \in G_2$, and $[\widehat{t}_{i,j}]_T \in G_T$, which essentially satisfy the original equations (except for the GS proof terms on the RHS). Using the batching technique from the introduction as external batching, we only need a single pairing to check the LHS of the fully batched equation.

On the other hand, Blazy et al. [11] propose to batch each equation individually, and sum them up. In our language, they batch the full set of (basic pairing) equations externally, using the standard short exponent batching. A simple computation shows, that this costs 4 pairings (for the LHS).

Now suppose that multiple equations of the form (26) have to be verified. Then the costs of the LHS dominate, and our technique is about 4 times faster than that of Blazy et al.

D P-SIGNATURES

We use the following definitions and naming conventions: The number of *systems* of equations (i.e. the number of P-signature proofs) we want to batch is N. A constant, variable, (external) batching randomness, etc. which depends the i^{th} system of equations (e.g. the i^{th} P-signature proof) is denoted by adding a superscript (i). For example we write $x_2^{(i)}$ for a variable in the i-th P-signature and $r_1^{(i)}, r_2^{(i)}, r_3^{(i)}$ for external batching randomness used to obtain the batched equation in Sec. 5.3 (i.e. randomness to "externally" batch the i-th P-signature proof). We use the "hat"-notation for the intermediate variables as in Eq. (21).

We present the most general formulas and mention specializations, but leave them to the reader. In particular, one can set $r_1^{(1)} = 1$ to optimize.

D.1 Our technique

In our notation, the constants $b_2, b_3 \in G_2$ are part of the verification key of P-signatures, and the constant $a_2 \in G_1$ is part of the common reference string. Thus we have $b_2^{(i)}, b_3^{(i)}$ if we batch with N distinct signature keys.

$$
\begin{aligned}
& e\Big(\Big(\sum_{i=1}^{N} -r_1^{(i)}\Big)[1]_1 + \sum_{i=1}^{N} r_3^{(i)}[\widehat{x}_2^{(i)}]_1, [1]_2\Big) \\
& + \sum_{i=1}^{N} e\Big(r_1^{(i)}[\widehat{x}_1^{(i)}]_1, [b_2^{(i)} + \widehat{y}_1^{(i)} + \widehat{y}_2^{(i)}]_2\Big) \\
& + \sum_{i=1}^{N} e\Big(r_3^{(i)}[\widehat{x}_3^{(i)}]_1, [b_3]_2^{(i)}\Big) \\
& + e\Big(-[a_2]_1, \sum_{i=1}^{N}[r_3\widehat{y}_1^{(i)} + r_2\widehat{y_2}]_2^{(i)}\Big) \\
& \overset{?}{=} e\Big([\widehat{u_1}]_1, [t_1\widehat{\pi}_1^{(1)} + t_2\widehat{\pi}_1^{(2)} + t_3\widehat{\pi}_1^{(3)}]_2\Big) \\
& + e\Big([\widehat{u_2}]_1, [t_1\widehat{\pi}_2^{(1)} + t_2\widehat{\pi}_2^{(2)} + t_3\widehat{\pi}_2^{(3)}]_2\Big) \\
& + e\Big([t_1\widehat{\theta}_1^{(1)} + t_2\widehat{\theta}_1^{(2)} + t_3\widehat{\theta}_1^{(3)}]_1, \widehat{v_1}]_2\Big) \\
& + e\Big([t_1\widehat{\theta}_2^{(1)} + t_2\widehat{\theta}_2^{(2)} + t_3\widehat{\theta}_2^{(3}], [\widehat{v_2}]_2\Big)
\end{aligned}
$$

If we consider the same-signer situation, we can "remove" the superscripts and write b_2, b_3. This is used to simplify the third sum of the left-hand side, which then needs a single pairing instead of N.

D.2 The formula of Blazy et al.

We state the left-hand side for batch-evaluating as in [11] in the notation of [11]. This term differs from [11, Appendix C]. Firstly, we correct an index mistake, the underlined $d_{2,1}$. Secondly, we make evaluation more efficient by moving $e(g,h)^{r_{2,2}}$ from the right-hand side to the left-hand side. Thirdly, we group pairings to get the claimed performance. To keep this readable, we did not apply the superscript (i) to the variables. The variables $c_{k,\ell}, d_{k,\ell}, r_{k,\ell}, s_{k,\ell}, t_{k,\ell}$ have this superscript. If the verification key is not fixed, then the constants v and w also depend on (i).

$$
\begin{aligned}
& \prod e(c_{1,1}, (vd_{1,2}d_{2,2})^{r_{1,2}}(d_{1,1}d_{2,1})^{r_{1,1}}) \\
\cdot & \prod e(c_{1,2}, (vd_{1,2}d_{2,2})^{r_{2,2}}(d_{1,1}\underline{d_{2,1}})^{r_{2,1}}) \prod e(c_{3,1}^{s_{1,2}} c_{3,2}^{s_{2,2}}, w^{-1}) \\
& \cdot e(\prod c_{2,1}^{t_{1,2}} c_{2,2}^{t_{2,2}} \underline{g^{r_{2,2}}}, h^{-1}) \cdot e(f, \prod d_{2,1}^{s_{2,1}} d_{2,2}^{s_{2,2}} d_{1,1}^{t_{2,1}} d_{1,2}^{t_{2,2}}) \qquad (27)
\end{aligned}
$$

[19] This holds true for batching as in [11], with higher constant costs.

D.3 The formulas in numbers

In Table 8 we give the precise number of exponentiations and pairings for one or multiple P-signature proof verifications.

Strategy	G_1	G_2	e_G	G	e_G
[11]	24	27	13	51	12
ours	17	16	8	34	8
[11]	$29N$	$32N$	$2N + 11$	$51N$	$3N + 9$
ours	$18N + 3$	$16N + 2$	$N + 7$	$30N + 7$	$N + 7$

Table 8: Comparison of P-signature batching under a single verification key. The left part of the table is for the SXDH instantiation, the right part for the symmetric 2-Lin instantiation. When batching a single signature (upper half), we use $r_1 = 1$. For batching N signatures (lower half), we choose $r_1^{(1)} \leftarrow S$, since this simplifies the estimates. See Appendix D for details. For multiple verification keys, (only) the number of pairings changes: Using [11] needs $3N + 10$ resp. $4N + 8$ pairings. Our technique needs $2N + 6$ (in both cases).

E BELENIOS RECEIPT FREE VOTING SYSTEM

E.1 General Remarks

The whole system in [19] depends on a parameter k. (This parameter is not to be confused with k_1, k_2 associated to GS instantiations.) A vote is a message $m = (m_1, \ldots, m_k) \in \{0, 1\}^k$. The parameter k should be set according to the number of candidates supported, which is k or 2^k depending on how it is used in a bigger protocol, more specifically, depending on whether votes are homomorphically added or shuffled. The paper implements several values of k (namely $k = 1, 5, 10, 25$) on different platforms (see Table 1, p. 1623). However, the timings are encrypting, signing and for computing GS proofs, not their verification. The authors claim that the number of pairings for verification is $4(k + 35)$ using suitable batching techniques. However, see Sec. E.5 for more on these numbers.

The basic verification process works in the following way: The verification algorithm receives $[(c_1, c_2, c_3)]_1$, $\mathbf{C}$, π, $[(\sigma_1, \sigma_2, \sigma_3)]_1$ where $[(c_1, c_2, c_3)]_1 \in G_1^3$ is the ciphertext, and $\mathbf{C}$ are a set of GS commitments, and π is a set of GS proofs that these commitments satisfy certain relations. All other elements in the verification equations below are public parameters. They will be named essentially as in [19] for comparability (except that the generator of G_i is not g_i as in [19] but $\mathcal{P}_i = [1]_i$). We define them as they appear. The GS instantiation is the SXDH instantiation from Sec. 2.6.5.

We split the verification process into two parts:

- Part I : Verification of GS proofs.
- Part II: Verification of a signature.

E.2 Part I

In an honest execution, the commitments

$$\mathbf{C} = \left(\{[\overline{c}_{1,m,i}]_1, [\overline{c}_{2,m,i}]_2\}_{i=1,\ldots,k}, [\overline{c}_T]_1, [\overline{c}_r]_2\}\right)$$

should open to the following values:

- For $i = 1, \ldots, k$, $[\overline{c}_{1,m,i}]_1 = [(\overline{c}^1_{1,m,i}, \overline{c}^2_{1,m,i})]_1 \in G_1^2$, and $[\overline{c}_{2,m,i}]_2 = [(\overline{c}^1_{2,m,i}, \overline{c}^2_{2,m,i})]_2 \in G_2^2$, are commitments to $m_i \in \{0, 1\}$ (in different groups).
- $[\overline{c}_r]_1 = [(c_{r,1}, c_{r,2})]_1 \in G_2^2$ is a commitment in G_2 to r, where $r = \mathrm{dlog}_{\mathcal{P}_1}([c_1]_1) = c_1$.
- $[\overline{c}_T]_1 = [(c_{T,1}, c_{T,2})]_1 \in G_1^2$ is a commitment in G_1 to $[rX_1]_1$, where $[X_1]_1$ is some public element.

The proof

$$\pi = \Big(\{[\vec{\theta}_{i,\mathrm{eq}}]_1, [\vec{\pi}_{i,\mathrm{eq}}]_2\}_{i=1,\ldots,k}, \{[\vec{\theta}_{i,\mathrm{quad}}]_1, [\vec{\pi}_{i,\mathrm{quad}}]_2\}_{i=1,\ldots,k},$$
$$[\vec{\theta}_{\mathrm{hash}}]_1, \{[\vec{\theta}_T]_1, [\vec{\pi}_T]_2\}, [\vec{\theta}_V]_1[\vec{\theta}_r]_1\Big)$$

consists of several subproofs and shows that the commitments are defined as described above. Namely, it consists of GS proofs of the following statements. For convenience, we use $[\vec{u'}]_1 := \iota_1^Z(1) = [\vec{u}_2 + \vec{e}_1]_1$ and $[\vec{v'}]_1 := \iota_2^Z(1) = [\vec{v}_2 + \vec{e}_1]_1$

(1) For all $i = 1, \ldots, k$, the commitments $[\overline{c}_{1,m,i}]_1$ and $[\overline{c}_{2,m,i}]_2$ open to the same value $\overline{m}_i \in \mathbb{Z}_p$. This is proven with GS proofs. For this, the verifier checks for each $i = 1, \ldots, k$ if:

$$\tilde{e}([\overline{c}_{1,m,i}]_1, [\vec{v'}]_2) - \tilde{e}([\vec{u'}]_1, [\overline{c}_{2,m,i}]_2)$$
$$\stackrel{?}{=} \tilde{e}([\vec{u}_1]_1, [\vec{\pi}_{i,\mathrm{eq}}]_2) + \tilde{e}([\vec{\theta}_{i,\mathrm{eq}}]_1, [\vec{v}_1]_2)$$

(2) For all $i = 1, \ldots, k$, the commitment $[\overline{c}_{1,m,i}]_1$ opens to x_i and $[\overline{c}_{2,m,i}]_2$ opens to y_i such that $x_i(y_i - 1) = 0$. For each $i = 1, \ldots, k$ the verifier checks if:

$$\tilde{e}([\overline{c}_{1,m,i}]_1, [\overline{c}_{2,m,i} - \vec{v'}]_2)$$
$$\stackrel{?}{=} \tilde{e}([\vec{u}_1]_1, [\vec{\pi}_{i,\mathrm{quad}}]_2) + \tilde{e}([\theta_{i,\mathrm{quad}}]_1, [\vec{v}_1]_2)$$

The previous two proofs together imply that $m_i \in \{0, 1\}$.

(3) Prove that

$$c_2 = \overline{r}[P]_1 + [\gamma_0]_1 + \sum_{i=1}^{k} m_i[\gamma_i]_1 \ ,$$

where the $[P]_1, [\gamma_i]_1 \in G_1$ are public values (this is essentially Waters' hash function) and $\overline{r}$ is the value committed in $[\overline{c}_r]_2$. This corresponds to the following verification equation:

$$\tilde{e}\left(\begin{pmatrix}[P]_1\\ [0]_1\end{pmatrix}, [\overline{c}_r]_2\right) + \tilde{e}\left(\begin{pmatrix}[\gamma_0]_1\\ [0]_1\end{pmatrix}, [\vec{v'}]_2\right) + \sum_{i=1}^{k} \tilde{e}\left(\begin{pmatrix}[\gamma_i]_1\\ [0]_1\end{pmatrix}, [\vec{c}_{1,m,i}]_2\right)$$
$$\stackrel{?}{=}\tilde{e}\left(\begin{pmatrix}[c_2]_1\\ [0]_1\end{pmatrix}, [\vec{v'}]_2\right) + \tilde{e}([\vec{\theta}_{\mathrm{hash}}]_1, [\vec{v}_1]_2)$$

This equation is one-sided linear. Thus, we used an optimized GS proof which has no proof term in G_2. Further, $[\vec{\theta}_{\mathrm{hash}}]_1$ is of a special form, namely $[\vec{\theta}_{\mathrm{hash}}]_1 = \begin{pmatrix}[\theta_{\mathrm{hash},1}]_1\\ [0]_1\end{pmatrix}$.

(4) Prove that $[\overline{c}_T]_1$ opens to a value $\overline{w}$ and $\overline{c}_r$ to $\overline{r}$, such that $\overline{w} = X_1^{\overline{r}}$. The verification equation is of the following form:

$$\tilde{e}\left(\begin{pmatrix}[X_1]_1\\ [0]_1\end{pmatrix}, [\overline{c}_r]_2\right) - \tilde{e}([\overline{c}_T]_1, [\vec{v'}]_2)$$
$$\stackrel{?}{=} \tilde{e}\left([\vec{u}_1]_1, [\vec{\pi}_T]_2\right) + \tilde{e}([\vec{\theta}_T]_1, [\vec{v}_1]_2)$$

(5) Prove that $[c_3]_1 = \overline{r}[H(\mathrm{vk})]_1$, where H is a collision resistant hash function and vk the verification key of the signature

scheme. The verification algorithm checks if:

$$\tilde{e}\left(\begin{pmatrix}[H(\mathrm{vk})]_1\\ [0]_1\end{pmatrix}, [\bar{c}_r]_2\right) \overset{?}{=} \tilde{e}\left(\begin{pmatrix}[c_3]_1\\ [0]_1\end{pmatrix}, [\vec{v}']_2\right) + \tilde{e}([\vec{\theta}_V]_1, [\vec{v}_1]_2)$$

Further, $[\vec{\theta}_V]_1$ is of a special form, namely $[\vec{\theta}_V]_1 = \begin{pmatrix}[\theta_{V,1}]_1\\ [0]_1\end{pmatrix}$. (See item 3 for the explanation.)

(6) Prove that $\bar{r}[1]_1 = [c_1]_1$, i.e. $\bar{r} = c_1$, where $\bar{r}$ is the value committed in $\bar{c}_r$. The verification algorithm checks if:

$$\tilde{e}\left(\begin{pmatrix}[1]_1\\ [0]_1\end{pmatrix}, [\bar{c}_r]_2\right) = \tilde{e}\left(\begin{pmatrix}[c_1]_1\\ [0]_1\end{pmatrix}, [\vec{v}']_2\right) + \tilde{e}([\vec{\theta}_r]_1, [\vec{v}_1]_2)$$

Further, $[\vec{\theta}_r]_1$ is of a special form, namely $[\vec{\theta}_r]_1 = \begin{pmatrix}[\theta_{r,1}]_1\\ [0]_1\end{pmatrix}$. (See item 3 for the explanation.)

Remark E.1. *The proof of item 4 is the only equation where $[\vec{c}_T]_1$ appears. We do not use dual mode commitments for $[\vec{c}_T]_1$ but simply ElGamal encryption because the equation is "right-simulatable", see e.g. [24] for justification.*

E.3 Part II

This part of verification is to check a signature. We have to check

$$\begin{aligned} e([\sigma_1]_1, [g_2]_2) &\overset{?}{=} e([c_1]_1, \sigma_4)\\ e([\sigma_2]_1, [g_2]_2) &\overset{?}{=} e([z]_1, [X_2]_2) + e([c_2]_1, [\sigma_4]_2)\\ e([\sigma_3]_1, [g_2]_2) &\overset{?}{=} e([[g_1]_1, [\sigma_4]_2)\\ e([\sigma_5]_1, [g_2]_2) &\overset{?}{=} e([[P]_1, [\sigma_4]_2) \end{aligned}$$

This can be written as

$$\tilde{e}_2\left(\begin{pmatrix}[\sigma_1]_1\\ [\sigma_2]_1\\ [\sigma_3]_1\\ [\sigma_5]_1\end{pmatrix}, [g_2]_2\right) \overset{?}{=} \tilde{e}_2\left(\begin{pmatrix}[c_1]_1\\ [c_2]_1\\ [g_1]_1\\ [P]_1\end{pmatrix}, [\sigma_4]_2\right) + \tilde{e}_2\left(\begin{pmatrix}[0]_1\\ [z]_1\\ [0]_1\\ [0]_1\end{pmatrix}, [X_2]_2\right), \tag{28}$$

where $\tilde{e}_2\colon G_1^4 \times G_2 \to G_T^4$ is defined as

$$\tilde{e}_2\left(\begin{pmatrix}[a]_1\\ [b]_1\\ [c]_1\\ [d]_1\end{pmatrix}, [f]_2\right) = \begin{pmatrix}e([a]_1, [f]_2)\\ e([b]_1, [f]_2)\\ e([c]_1, [f]_2)\\ e([d]_1, [f]_2)\end{pmatrix} .$$

E.4 Batching

Part I. Choose small exponents for internal batching, namely $r_1 = s_1 = 1$ and random $r_2, s_2 \leftarrow \{0, \dots, 2^{82} - 1\}$. Use the polynomial interpretation of Ex. 5.1. That is, apply the linear maps $G_1^2 \ni [(x_1, x_2)]_1 \mapsto [x_1 + r_2 x_2]_1 \in G_1$, and $[(y_1, y_2)]_2 \mapsto [y_1 + s_2 y_2]_2 \in G_2$, and $G_T^{2\times 2} \ni ([z_{i,j}]_T)_{i,j} \mapsto \sum_{i,j=1}^{2} r_i s_j [z_{i,j}]_T$. This yields the intermediate variables we write as $[\widehat{c}_{1,m,i}]_1$, and so on. (Note that if the second component is zero then the linear map does nothing, e.g. for $[\theta_V]_1$ we get $[\widehat{\theta}_V]_1 = [\theta_{V,1}]_1$.)

In total, for Part I, we need to verify $2k + 3$ equations. We do not batch the equations for Parts I and II together, because they have disjoint variables and this does not help to reduce the number of pairings. In fact, batching them together only incurs additional exponentiations.

Using external batching, we combine all equations of Part I into a single equation. Namely we let $S := \{0, \dots, 2^{82} - 1\}$. and we sample random small coefficients, where for item 1 we use $a_i \leftarrow S$, for item 2 we use $b_i \leftarrow S$, $i = 1, \dots, k$ and for items 3,4,5 and 6 we use $t_1 = 1$, and $t_2, t_3, t_4 \leftarrow S$. Then we compute the random linear combination of these claims and optimize the evaluation by applying the rules of Ferrara et al. The result is following equation:

$$\begin{aligned} &\sum_{i=1}^{k} e\Big([-a_i\widehat{u'} + b_i\widehat{c}_{1,m,i} + \gamma_i]_1,\ [\widehat{c}_{2,m,i}]_2\Big)\\ &+ e\Big(\Big[\sum_{i=1}^{k}(a_i - b_i)\widehat{c}_{1,m,i} + \gamma_0 + c_2 - t_2 c_T + t_3 c_3\Big]_2,\ [\widehat{v'}]_2\Big)\\ &+ e\Big([P + t_2 X_1 + t_3 H(\mathrm{vk} + t_4)]_1, [\widehat{c}_r]_2\Big)\\ &\overset{?}{=} e\Big([\widehat{u}_1]_1,\ \Big[\sum_{i=1}^{k} a_i\widehat{\pi}_{i,\mathrm{eq}} + \sum_{i=1}^{k} b_i\widehat{\pi}_{i,\mathrm{quad}} + t_2\widehat{\pi}_T\Big]_2\Big)\\ &+ e\Big(\Big[\sum_{i=1}^{k} a_i\widehat{\theta}_{i,\mathrm{eq}} + \sum_{i=1}^{k} b_i\widehat{\theta}_{i,\mathrm{quad}} + \widehat{\theta}_{\mathrm{hash}} + t_2\widehat{\theta}_T + t_3\widehat{\theta}_V + t_4\widehat{\theta}_r\Big]_1,\\ &\qquad [\widehat{v}_1]_2\Big). \end{aligned} \tag{29}$$

Part II. Choose additional small exponents $r_3, r_4 \leftarrow S$. Here we use the technique of Sec. 6. That is, we essentially multiply equation (28) by $\vec{\rho} = (r_2, 1, r_3, r_4)$. (We use this instead of $(1, r_2, r_3, r_4)$ for efficiency). Thus we compute $[\widehat{\sigma}]_1 = \vec{\rho}[(\sigma_1, \sigma_2, \sigma_3, \sigma_5)]_1^\top$, and $[\widehat{x}]_2 = \vec{\rho}[(c_1, c_2, g_1, P)]_2^\top$ and $[\widehat{z}]_1 = [z]_1$. We check if

$$e([\widehat{\sigma}]_1, [g_2]_2) \overset{?}{=} e([\widehat{x}]_1, [\sigma_4]_2) + e([z]_1, [X_2]_2) \tag{30}$$

holds.

Total number of pairings. For Part I we need $k + 4$ pairings. for Part II we need 3. Thus in total, we need $k + 7$ pairings.

Soundness. Soundness loss is $3/2^{82} < 2^{-80}$ for this choice of S.

E.5 A note on the numbers

We contacted the authors of [19] to learn the details of their verification optimizations. They use the QANIZK proofs from [33] rather than GS proofs for proving that the commitments $[\bar{c}_{i,m,1}]_1$ are bit commitments. The remaining equations are proven with GS proofs and verification is improved with the batching techniques of Blazy et al. [11].

The authors communicated a better batching formula for the GS equations improving the total cost from $4k + 140$ to $3k + 86$ pairings. Building on their result and adding external batching, we could improve it to $3k + 63$ pairings. Here, the GS proofs cost $k + 11$ and the QANIZKs $2k + 52$.

These numbers are derived without using any batching techniques for the verification of the QANIZK proof. If we apply our results, we can reduce the verification costs of the proof of [33] that $[\bar{c}_{i,m,1}]_1$ are bit commitments from $2k + 52$ down to $2k + 22$ pairing evaluations. This is still worse than our results with only GS proofs, which needs $k + 7$ pairings in total. It is interesting that while the QANIZK proof has very small size (constant), it is less amenable to batch verification and should be avoided when the goal is to minimize verification complexity.

We also evaluated how well batching as in [11] performs when only GS proofs and no QANIZKs are used. After several attempts, and carefully using the "linear properties" of [11], we achieved $2k + 11$ pairings. This is still better than the QANIZK approach. Arguably, this illustrates how difficult it is to find the right verification formula with the approach of [11] without automated tools.