Constant-Size Inner Product Arguments for Group-Scalar Relations, Dynamic Threshold VRFs, and More

Omid Mir[1]*, Octavio Perez-Kempner[2], Sebastian Ramacher[1], and Daniel Slamanig[3]

[1] AIT Austrian Institute of Technology, Vienna, Austria
{omid.mir, sebastian.ramacher}@ait.ac.at
[2] NTT Social Informatics Laboratories, Tokyo, Japan
octavio.perezkempner@ntt.com
[3] Research Institute CODE, Universität der Bundeswehr München, Germany
daniel.slamanig@unibw.de

Abstract. Linear algebraic relations, such as inner products $\langle \mathbf{a}, \mathbf{b} \rangle$, underlie a wide range of cryptographic constructions, including zero-knowledge proofs, SNARKs, polynomial commitment schemes, and more. In this work, we consider group-scalar relations, i.e., statements of the form $\langle \mathbf{A}, \mathbf{b} \rangle$, where $\mathbf{A}$ is a vector of group elements and $\mathbf{b}$ is a vector of field elements. In many cryptographic settings, it is necessary to prove relationships between group elements like public keys, or other cryptographic objects without access to the underlying discrete logarithms. Our results are as follows:

- At the protocol level, we introduce the first Inner Product Argument (IPA) that specifically focuses on group-scalar relations in bilinear groups. It achieves constant-size proofs and constant-time verification, maintaining commitments and arguments entirely in the source group. Our techniques enable new applications and significantly improve efficiency compared to state-of-the-art IPAs such as Dory (TCC '21) and GIPA (Asiacrypt '21), which rely on recursive folding techniques and thus have logarithmic proofs and verification time. We prove security in the Algebraic Group Model under the q-DHE and q-DL assumptions.
- At the primitive level, we present a new class of functional commitments for linear functions over group-scalar elements. It enables even more applications such as polynomial commitments for values hidden inside group exponentiations.
- To showcase our contributions, we demonstrate new applications—most notably, we introduce the notion of *dynamic* threshold verifiable random functions, which we believe to be a valuable tool for distributed randomness generation. We further present dynamic threshold signatures without random oracles, polynomial commitments over group-encoded inputs, and their applications to oblivious proofs.

Our results provide modular and efficient tools to build cryptographic protocols without typical SNARK frameworks, simplifying real-world deployments. To demonstrate the practicality of our contributions, we provide an implementation and related benchmarks.

Keywords: Functional Commitments, Inner Product Arguments, Threshold VRF, Threshold Signatures, Succinctness

1 Introduction

Group-scalar inner product relations of the form $\langle \mathbf{A}, \mathbf{b} \rangle$, where $\mathbf{A}$ is a vector of group elements and $\mathbf{b}$ a vector of field elements, naturally arise in many cryptographic settings that involve relationships between group elements and public keys. For instance, given $\{b_i\}$ and $\{h_i = P^{y_i}\}$, a prover may need to demonstrate that $a = \prod_i h_i^{b_i}$ is correctly computed—without knowing the exponents $\{y_i\}$. Efficient protocols for such a task enable (new) powerful applications and directly contribute to their efficiency.

A very popular primitive that benefits from inner produce relations like the above is a threshold signature. This primitive enables a t-out-of-n distributed signing process, whereby any subset of t out of n parties can jointly produce a signature. To generate the keys, a distributed key generation (DKG) algorithm is usually run to secret-share the private key so that it can only be reconstructed

* Corresponding author: omid.mir@ait.ac.at

from t valid shares. While substantial efforts have been put on efficiently instantiating DKGs for threshold signatures (*e.g.*, [1,34,26,35,42]), they represent a significant bottleneck with little flexibility, *i.e.*, re-execution is needed whenever the set of parties changes. Besides, as pointed out in [32], most of the threshold schemes are designed to deal with the *unweighted* case (*i.e.*, exactly t different signers are required to satisfy threshold t) and those who consider weights, like [58] and [20], are far less efficient.

Recent work proposed *Multiverse Threshold Signatures* (MTS) [3], enabling parties to use their long-term secret keys and thus eliminating the need for a DKG. The underlying idea is to instantiate threshold signatures from multi-signatures (the special case where $t = n$). This is done by carefully defining auxiliary information that parties can use to create different sets of signers (universes) and thresholds. Subsequent work [24,33,40], allowed for a *dynamic* choice of thresholds and signers after a *silent setup* (*i.e.*, parties do not exchange any message to generate keys), without requiring further interaction. In other words, the aggregated public key that identifies a set of n signers, supporting a threshold t, is computed as a deterministic function of their individual keys. All existing works follow the same blueprint, which requires the aggregator to compute proofs for inner-product relations (*e.g.*, for proving threshold and weights) but also a succinct non-interactive argument of knowledge (SNARK) to prove the well-formedness of the aggregated public key (that is pre-computed in [40]). This can be a significant performance bottleneck.

We identify distributed or threshold VRFs [30] as a closely related primitive to threshold signatures that can similarly benefit from group-scalar inner product arguments. A verifiable random function (VRF) [57] is a pseudorandom function whose evaluations are publicly verifiable, *i.e.*, given a public verification key vk and an evaluation proof, anyone can verify that a VRF output was correctly computed. An increasingly popular related notion is that of distributed or threshold VRF [22,43]. Here each party can compute a partial evaluation and any t valid partial evaluations can be combined to obtain the (final) global evaluation of the VRF. Threshold VRFs are widely used in practice in distributed applications such as Proof-of-Stake (PoS) consensus protocols and randomness beacons; see [22,43,25] for a comprehensive overview. However, existing threshold VRF constructions are too costly for these applications due to their reliance on expensive distributed key generation (DKG) protocols, which must be rerun frequently—particularly whenever a new committee is selected. For instance, in such systems, DKG protocols are often executed per epoch (e.g., Ethereum, Aptos) or even per block (e.g., Algorand), making repeated coordination a significant overhead. Note that, the evaluation threshold and the set of participating parties cannot be easily modified without re-running the entire DKG protocol. In contrast to dynamic threshold signatures, no analogous notion exists for VRFs that would resolve these limitations. Our goal is to address this gap by enabling flexible and low-overhead selection of the evaluator set in VRFs. We discuss these applications in more detail in Section 1.1.

More broadly, we show that group-scalar relations are valuable not only in threshold cryptography but also in scenarios where one needs to prove relationships—such as linear combinations—between group and scalar elements without access to the underlying discrete logarithms. This setting arises in many cryptographic protocols, such as ring signatures [45] or oblivious proofs [32], where proofs are generated without knowing the underlying secrets. Latter has further applications, as shown by Garg et al. [32], including verifiable private delegation of computation and private outsourcing of zkSNARKs to a single untrusted server—an efficient approach for delegating zkSNARK generation while preserving privacy.

To improve cryptographic primitives such as dynamic threshold signatures—and even enable new ones like dynamic threshold VRFs—we study the underlying primitives that make them possible. Towards that end, we begin by discussing the core building blocks to then switch the focus to how they can be improved, enabling a broader scope of applications.

Group-Scalar Inner Product Relations. As mentioned, inner-product relations underpin a broad range of constructions including zero-knowledge proofs, SNARKs, polynomial commitment schemes, and more. We focus on a specific class of the form $\langle \mathbf{A}, \mathbf{b} \rangle$, where $\mathbf{A}$ is a vector of group elements and $\mathbf{b}$ is a vector of field elements. Such *group-scalar* relations naturally arise in many settings, *e.g.*, when proving relationships between group elements such as public keys in the above threshold primitives without knowing their discrete logarithms (*i.e.*, secret keys).

We initiate a comprehensive exploration of this setting by introducing new constructions and in particular a functional commitment and a polynomial commitment scheme tailored for group-

scalar inputs. Building on these foundations, we then present the first inner product argument for group-scalar relations that achieves *constant-size proofs* and *constant verification time*.

Functional Commitments. We recall that a functional commitment (FC) scheme allows a user to commit to a function f and later produce succinct, publicly verifiable proofs of evaluations $y_i = f(x_i)$. Commitments and proofs should be sublinear in the size of f, and it must be ensured that the commitment uniquely determines the function (called evaluation binding). The FC notion was first formally defined by Libert *et al.* in [51], where one can commit to vectors over a field $\mathbb{F}$ and later open linear functions $f : \mathbb{F}^m \to \mathbb{F}^n$, *i.e.,* inner products, of these vectors. The authors of [51] show that linear functions are sufficient to encompass earlier notions of specific functionalities, such as vector commitments [19,52], polynomial commitments [41,61,51], and linear map commitments [18,46]. These constructions have found numerous applications including verifiable outsourced storage [7], updatable zero-knowledge sets and databases [54], accumulators [8], succinct non-interactive arguments (of knowledge) [9,46] and many more. Due to their versatility and wide-ranging applications this is a very active field (cf. [60] for a detailed overview).

We introduce a new FC construction for linear functions, *i.e.,* group-scalar inner products, with *constant-size proofs* and *constant verification time*. Moreover, we show that it yields polynomial commitments for values hidden inside group exponentiations [32].

Inner Product Arguments. An Inner Product Argument (IPA) as introduced by Bootle et al. [13] allows a prover to convince a verifier that two vectors, committed using Pedersen commitments, satisfy a known inner product. In [13] the protocol recursively reduces the size of the vectors by half in each round through linear combinations, requiring logarithmic rounds relative to the vector length. Bulletproofs [14] improves this approach, consolidating commitments and reducing communication overhead and were later extended in [63,27]. However, they still have logarithmic communication and verification costs. In the realm of IPAs for vectors over a field, recent techniques have been developed that enable the construction of IPAs with constant-size proofs and verification time, such as the Linear Map Vector Commitment (LMVC) [18] and the Counting Vampires protocol [53]. We note that the works of [23,2] achieve constant-size proofs in groups of unknown order.

In contrast to the above schemes that focus on field elements, our work focuses on IPAs for *group elements*. When extending IPAs to scenarios involving group elements, the landscape shrinks significantly. The Generalized Inner Product Argument (GIPA) [15] and Dory [48] are among the few protocols specifically designed to handle inner product relations involving group elements. Another relevant work is [47], which presents a GIPA for groups elements as part of their succinct arguments for bilinear group arithmetic. For our comparison we focus on the schemes [15,48], which explicitly present GIPA constructions. Both schemes extend traditional IPAs, leveraging folding techniques to reduce proof size and verification complexity. Dory, in addition, provides a transparent setup and a slightly more efficient concrete construction, while remaining asymptotically similar to GIPA.

In this work, we introduce a novel IPA specifically designed for group-scalar relations. By achieving *constant-size proofs* and *constant verification time*, this marks a significant improvement over existing schemes like GIPA and Dory, which exhibit logarithmic complexities in both proof size and verification time. Moreover, both require pairing operations and involve elements in the target group $\mathbb{G}_T$, which are significantly larger and more expensive than elements and operations in the source groups. We provide a comparison with our scheme, which we call GSIPA, in Table 1. At this point we want to mention that the threshold signature scheme of [24] implicitly uses an IPA as part of a larger construction. However, we exclude it from our comparison for two reasons. First, their IPA is not a standalone primitive and not proven to be an IPA with independent knowledge-soundness guarantees. Second, recent analysis in [49] shows that the knowledge soundness of their proof statement used to prove unforgeability of their scheme is flawed. Our focus is on *general-purpose, standalone* IPA which can prevent this kind of issue.

1.1 Applications

Now we want to provide a brief overview of the applications enabled by our group-scalar inner-product argument GSIPA (cf. Sec. 4) and the PCS (cf. Sec. 3.2) that follows from our FC scheme for linear functions GSFC (cf. Sec. 3).

Dynamic Threshold (Weighted) VRFs. A variant of VRFs that is receiving a lot of attention recently

Table 1. Comparison between our scheme GSIPA, GIPA, and Dory in terms of transparent setup (TS), communication, and time complexity. $\mathbb{G}_i$, $i \in \{1, 2, T\}$ denotes an asymmetric (type-3) pairing-group and $\mathbb{P}$ a pairing.

Scheme	TS	Comm. (Com / Eval)	Time Com	Time Eval ($\mathcal{P}$)	Time Eval ($\mathcal{V}$)
GIPA	$\times$	$1\,\mathbb{G}_T$ / $\log n\,\lvert\mathbb{G}_T\rvert$	$n\,\mathbb{G}_1$	$\sqrt{n}\,\mathbb{P}$	$\log n\,\mathbb{G}_T$
Dory	✓	$1\,\mathbb{G}_T$ / $\log n\,\lvert\mathbb{G}_T\rvert$	$n\,\mathbb{G}_1$	$\sqrt{n}\,\mathbb{P}$	$\log n\,\mathbb{G}_T$
GSIPA	$\times^\star$	$1\,\mathbb{G}_1 + 1\,\mathbb{G}_2$ / same	$\mathcal{O}(1)^\dagger$	$n^2\,\mathbb{G}^\blacklozenge$	$\mathcal{O}(1)$

* Although our scheme requires a trusted setup like GIPA, it remains compatible with standard trust-reduction techniques such as MPC-based or updatable CRS; see Sec.4 for details.

† Our commitment phase involves $\mathcal{O}(n)$ group additions, which for practical sizes of n is insignificant compared to group exponentiations; see the implementation section for details.

$^\blacklozenge$ Although this is asymptotically more expensive, in practice we only require scalar multiplications and group operations in the source group, which are significantly faster and cheaper than pairing operations used in other schemes.

are distributed or threshold VRFs. Here, the evaluation key is secret-shared among n parties, and a quorum of at least t parties is required to evaluate the VRF. Threshold VRFs provide enhanced fault tolerance, decentralization, and robustness compared to standard VRFs. By distributing the secret key, they eliminate single points of failure, ensuring that no single party can control or bias the VRF output. Threshold VRFs [28,30,55] are used widely in practice in distributions applications such as in Proof-of-Stake (PoS) consensus protocols, where they serve as a randomness beacon [22,25,31]. However, this model introduces several challenges and in particular the reliance on distributed key generation (DKG) adds significant overhead due to its interactivity, making it expensive in large-scale deployments. Moreover, once the system is initialized, the evaluation threshold and participating parties cannot be easily modified without rerunning the entire DKG protocol again. These limitation can be prohibitive in PoS blockchain applications, where a DKG must be executed frequently and whenever a new committee is selected. We note that while recent work on non-interactive distributed verifiable random functions (NI-DVRFs) [55] reduce the round complexity of the DKG to a single round, i) it still has to be rerun frequently and ii) the only existing construction in [55] relies on zk-SNARKs with significant memory consumption and prover times. Moreover, the approach requires to prove pre-images of random oracles where they are simultaneously treated as an explicit circuit (and due to this paradox leading to unclear security guarantees).

Inspired by silent threshold signatures, we put forth the notion of *dynamic* threshold VRFs (DT-VRFs). This new primitive enables every party to use an extended public key, consisting of additional values that are locally computed using the secret key and made public (this is referred as a hint in [33]), removing the need for a DKG. It also supports a dynamic selection of thresholds and parties where parties can join in an *ad-hoc* manner (up to the upper bound of participants q), publishing their public keys asynchronously. Moreover, it allows for weighted thresholds, *i.e.,* the weight of a party in the overall threshold depends on their respective weights and the threshold is determined by the sum of their weights. We give the first practical construction of DT-VRFs by carefully integrating the aggregate VRF from [56] with our group-scalar IPA GSIPA. One point that is worth mentioning is that due to the dynamic nature regarding parties and thresholds, a DT-VRF satisfies a different notion of uniqueness that we call subset uniqueness, *i.e.,* uniqueness holds with respect to concrete subset of participants. While this is different to global uniqueness in exciting static TVRFs, most VRF applications—such as randomness beacons, lotteries, and leader election—do not require global uniqueness. These protocols either define a single subset per round (e.g., elected committee is chosen to produce randomness or select a leader), accept the first valid output, or resolve conflicts deterministically (e.g., by lexicographic order or aggregator selection). DT-VRFs with this subset uniqueness remain verifiable and pseudorandom for the generating subset. Moreover, related outputs do not compromise correctness or security, as protocols ensure only one output is accepted per round. All in all, by removing the need to re-run a DKG, the relaxation of global uniqueness enables silent setup and scalability, which we believe are critical for many applications where the subset of evaluators is chosen frequently and flexibly e.g., in blockchain consensus protocols and randomness beacons.

Finally, we want to note that DT-VRFs are different from the recently proposed ring VRFs [16]. There VRFs are computed with respect to an ad-hoc group of participants, and like with ring signa-

tures only one of the participants uses its secret key and it is ensured that one participant of the ring evaluated the VRF. Thus the VRF evaluation is more local than in DT-VRFs, *i.e.,* it is unique only with respect to a single participant.

Silent Threshold Signatures Without RO. With the second application we demonstrate the use of GSIPA to construct a new threshold signature scheme with silent setup as explained next.

Our starting point is the blueprint followed by [24] and [33], which considers a large fixed group of signers who compute multi-signature shares of BLS signatures [11]. Concretely, they consider a one-time set up where each signer publishes additional key-dependent information (called hints in [33]). The term *silent* originates in that the required setup is non-interactive (*i.e.,* parties do not need to exchange any information with each other). The set up proceeds by computing one *aggregation key* AK and one short *verification key*. The aggregation key is then used by any aggregator to compute the aggregated public key apk against which the final signature will be verified. To that end, the aggregator first aggregates all signature shares and computes apk using the information of each signer in AK. Subsequently, it proves via a SNARK that apk is the inner product between a vector commitment that contains the group public keys and a bit vector defining the threshold.

We consider the recent accountable multi-signature without random oracles (ROs) from Boneh, Partap and Waters [12], which we adapt to the silent threshold setting following the ideas from [33]. We observe that for silent threshold signatures accountability is not required and we can thus remove the linear dependency on the overall set of potential signers for the signature size. Specifically, we can get rid of the full SNARK frameworks and using our GSIPA, we can commit to the bit vector and auxiliary group elements used to verify the signature to get a silent threshold scheme. Thus, we extend the setup of [12] to include our GSIPA's setup, requiring the use of extended public keys defined over the GSIPA 's message space. Key aggregation can be done as parties join the system, verifying the correctness of each (extended) public key using the crs. Subsequently, the aggregator only needs to fetch the public keys of the threshold signing parties to compute the GSIPA. This allows us to achieve a constant-size silent threshold scheme that, contrary to prior work, does not rely on the random oracle model (ROM) to prove security. In this regard, we note that recent attacks, such as [44], highlight that RO-based proofs can fail in practice, underscoring the need for constructions that avoid this heuristic altogether.

The above approach also aligns with our broader goal of achieving practical and flexible constructions based on well-understood primitives. Simplicity is a central strength of our construction—both conceptually and practically. It requires fewer cryptographic components and assumptions than state-of-the-art alternatives, thereby reducing integration complexity, potential attack surfaces, and the likelihood of subtle implementation bugs. In real-world deployments, solutions that are easier to audit, verify, and maintain are often preferred over theoretically more efficient but structurally complex alternatives. For example, Coinbase's MPC protocol [4] for threshold ECDSA reflects such engineering trade-offs in practice. Our construction aligns with these priorities, making it particularly attractive to a broader audience beyond its theoretical relevance.

Oblivious Proofs for Group Exponentiation. Garg et al. in [32] present a proof system that enables proving statements, *i.e.,* arbitrary circuit relations, about hidden secrets. In particular, these proofs are generated in an oblivious way without access to the secrets themselves. This enables powerful applications such as including a SNARK for proving correct evaluation over encrypted or hidden data or delegating SNARK generation to an untrusted server without revealing the witness.

In [32] Garg et al. introduce a technique called *FRI for hidden values*, which supports efficient proximity testing on encapsulated inputs, referred to as $\mathsf{LHEncap}(m)$ functions. Our polynomial commitments in Sec. 3.2 can achieve similar functionality *without relying on the FRI protocol* and by considering $\mathsf{LHEncap}(m)$ as group exponentiation. This yields a polynomial commitment scheme for group-exponentiated values that supports *constant-size proofs and verification time* and by applying the randomization technique described in [45], we can achieve perfect hiding for group element messages. Similar to [32], our scheme naturally supports integration with polynomial IOP-based SNARKs. Finally assuming $\prod_{j\in[q]} P^{\alpha^j}$ can be precomputed and our IPA serves as a core component in such proofs, one could plausibly achieve constant verification time with appropriate tuning.

[4] https://github.com/coinbase/cb-mpc

1.2 Our Contributions

After discussing the state-of-the-art and our contributions in comparison to it, we want to briefly summarize the contributions below:

- **FC for Group-Scalar Inner Products:** We present the *first* functional commitment (FC) scheme for inner-product relations $\langle \mathbf{A}, \mathbf{b} \rangle$ with constant-size proofs and verification time, while ensuring that the commitment/proof remains in the source group. Our scheme is inspired by the structure-preserving vector commitment [45], particularly in terms of the message space and commitments. However, we significantly adapt the proof generation and verification process to turn it into an FC scheme and support inner-product relations. Our FC scheme, using the framework in [51], also yields a new form of polynomial commitment scheme for values hidden inside group exponentiations. This can be seen as a concrete instantiation of the more general framework of Polynomial Commitments for Hidden Values introduced by [32], tailored to the specific case of linear homomorphic encapsulation via group exponentiation.
- **GSIPA:** We present an inner product argument (IPA) for group-scalar relations, *i.e.*, $(x, w) \in \mathcal{R}_{IP}$, where $x = y$ and $w = (\mathbf{A}, \mathbf{b})$, the prover demonstrates that $\langle \mathbf{A}, \mathbf{b} \rangle = y$. This is the first IPA for group elements that achieves constant-size proofs and constant verification time, wheres existing schemes for group elements, such as Dory [48] and GIPA [15], use folding techniques and incur logarithmic proof size and verification time. Compared to schemes for just scalar inner-product relations, our approach matches the efficiency of LMVC [18] and Counting Vampires [53], and is significantly more efficient than Bulletproofs [14]. Moreover, our scheme operates entirely in the source groups and avoids pairing operations and the use of target group elements $\mathbb{G}_T$, which are significantly larger and contribute to increased proof sizes in existing schemes.
- **Applications:** We present the following applications: First, we introduce the notion of dynamic (weighted) threshold VRFs (DT-VRFs), which unlike existing distributed or threshold VRFs [28,30] avoid the need for costly distributed key generation (DKG) protocol. DT-VRFs enables a non-interactive setup and supports dynamic thresholds, allowing parties to join in an ad-hoc manner without any synchrony assumptions, which makes them a better building block for applications like distributed randomness beacons [22,25,31]. We present a construction based on Malavolta's aggregate VRF [56] and our GSIPA. Second, we present a dynamic threshold signature based on the recent accountable multi-signature without random oracles from Boneh, Partap and Waters [12] and our GSIPA, obtaining the first threshold signature scheme with silent setup without random oracles. Third, we discuss the application of our polynomial commitment scheme for values hidden inside group exponentiations and show an alternative design for oblivious proofs.
- **Implementation:** Finally, we present benchmarks from a prototype implementation and instantiate our dynamic threshold VRF scheme to demonstrate the practicality of our contributions.

2 Preliminaries

We use $\mathsf{BG} = (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, P, \hat{P}) \leftarrow \mathsf{BGSetup}(1^\lambda)$ to denote a bilinear group generator for asymmetric type-3 bilinear groups, where p is a prime of bit length λ. We use $[q]$ to denote the set $\{1, 2, \ldots, q\}$. When drawing multiple values from a set, we may omit notation for products of sets, e.g. $(x, y) \in \mathbb{Z}_p$ is the same as $(x, y) \in \mathbb{Z}_p^2$. For brevity, we will sometimes denote the elements in a vector as $\mathbf{V} = (V_i)_{i \in [q]} = (V_1, \ldots, V_q)$. Given a finite set S, we denote by $x \leftarrow\!\!\$\ S$ the sampling of an element uniformly at random from S. For an algorithm A, let $y \leftarrow A(x)$ be the process of running A on input x with access to uniformly random coins and assigning the result to y. With A^B we denote that A has oracle access to B. We assume all algorithms are polynomial-time (PPT) unless otherwise specified and public parameters are an implicit input to all algorithms in a scheme. Additionally, $|N|$ represents the length of N. We denote the inner product of two vectors $\mathbf{a} = (a_1, \ldots, a_q)$ and $\mathbf{b} = (b_1, \ldots, b_q)$ by $\langle \mathbf{a}, \mathbf{b} \rangle$. This notation extends naturally to vectors whose entries are group elements. Specifically, given $\mathbf{M} \in \mathbb{G}_i^q$, we define $Y = \langle \mathbf{b}, \mathbf{M} \rangle = \prod_{i \in [q]} M_i^{b_i} \in \mathbb{G}_i$. We denote the Hadamard (component-wise) product of two vectors $\mathbf{a} = (a_1, \ldots, a_q)$ and $\mathbf{b} = (b_1, \ldots, b_q)$ by $\mathbf{a} \circ \mathbf{b}$.

2.1 The Algebraic Group Model

The Algebraic Group Model (AGM) [29] is an idealized model, in which adversaries are algebraic. Algebraic algorithms generalize the notion of generic algorithms [62] in the sense that whenever

adversaries compute a group element, they do so by means of generic operations *i.e.*, computing linear combinations of previously known group elements. Moreover, whenever they output a group element $X \in \mathbb{G}$, they also output a representation $(a_i)_{i=1}^{q}$ of $X = \prod_{i=1}^{q} B_i^{a_i}$ as a function of previously observed group elements $(B_1, \ldots, B_q) \in \mathbb{G}^q$. The AGM is a very powerful tool to establish the security of efficient protocols via reductions.

2.2 Assumptions

Definition 1 (Diffie-Hellman Exponent (q-DHE) [17]). *Let $\mathbb{G}$ be a finite cyclic group of order p, P be a generator of $\mathbb{G}$. The q-DHE problem is, given a tuple of elements $(P, B_1, \ldots, B_q, B_{q+2}, \ldots, B_{2q})$ such that $B_i = P^{\alpha^i}$ for $i = 1, \ldots, q, q+2, \ldots, 2q$ for $\alpha \leftarrow\!\$\ \mathbb{Z}_p^*$, to compute the missing group element $B_{q+1} = P^{\alpha^{q+1}}$ in the sequence. As shown in [52,36], q-DHE can be modified so as to work in the asymmetric pairing setting i.e., given $\{P^{\alpha^1}, \ldots, P^{\alpha^q}, P^{\alpha^{q+2}}, \ldots, P^{\alpha^{2q}}; \hat{P}^{\alpha^1}, \ldots, \hat{P}^{\alpha^q}; e(P, \hat{P})^{\alpha^{q+1}}\}$ it is hard to find $P^{\alpha^{q+1}}$.*
Generalized Version. *In our setting, we consider a further generalization in asymmetric bilinear groups, where the adversary is given: $\{P^{\alpha^1}, \ldots, P^{\alpha^q}, P^{\alpha^{q+2}}, \ldots, P^{\alpha^{2q}} \in \mathbb{G}_1;\ \hat{P}^{\alpha^1}, \ldots, \hat{P}^{\alpha^{2q}} \in \mathbb{G}_2\}$ and asked to compute $P^{\alpha^{q+1}} \in \mathbb{G}_1$.*

This assumption can be naturally framed as an instance of the Uber assumption framework for the AGM [4], where both the challenge and target exponents are univariate monomials in a formal variable X of degree at most $2q$. Specifically, the challenge set includes exponents of the form X^i for $i \in \{1, \ldots, q\} \cup \{q+2, \ldots, 2q\}$ for $\mathbb{G}_1$ elements, X^i for $i \in \{1, \ldots, 2q\}$ for $\mathbb{G}_2$ elements, and the target exponent is X^{q+1} (in $\mathbb{G}_1$), which is clearly linearly independent of the challenge polynomials. This satisfies the non-triviality condition of the Uber assumption. As shown in Corollary 3.6 (and related Lemmas 2.5 and 2.6) of [4], such instances in the AGM reduce to the $(2q-1, 2q)$-DLog problem.

Definition 2 ((m,n)-Discrete Logarithm [29,4]). *Let $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T)$ be an asymmetric bilinear group of prime order p. For integers m, n, the (m,n)-Discrete Logarithm ((m,n)-Dlog) problem is, given*

$$(P, B_1 = P^{\alpha^1}, B_2 = P^{\alpha^2}, \ldots, B_m = P^{\alpha^m}, \hat{P}, \hat{B}_1 = \hat{P}^{\alpha^1}, \ldots, \hat{B}_n = \hat{P}^{\alpha^n}),$$

where $\alpha \leftarrow\!\$\ \mathbb{Z}_p^*$, $P \leftarrow\!\$\ \mathbb{G}_1$, $\hat{P} \leftarrow\!\$\ \mathbb{G}_2$, to compute $\alpha \in \mathbb{Z}_p^*$.

This assumption is similar to the n-discrete logarithm assumption considered in, *e.g.*, [29], except that powers α^i are given in the exponents in both groups $\mathbb{G}_1$ and $\mathbb{G}_2$, where this case is considered in Lemma 2.6 in [4].

2.3 Non-Interactive Arguments of Knowledge

Definition 3 (Non-Interactive Argument of Knowledge). *A* (succinct) *non-interactive argument of knowledge for a relation $\mathcal{R}$ is defined by three algorithms* Setup, Prove, *and* Verify *that work as follows:*

$\underline{\mathsf{Setup}(1^\lambda) \to \mathsf{crs}}$: *The setup algorithm takes as input a security parameter λ and outputs a common reference string* crs.
$\underline{\mathsf{Prove}(\mathsf{crs}, \mathsf{aux}, x, w)}$: *The prover algorithm $\mathcal{P}$ takes as input a* crs, *some auxiliary information* aux, *a statement-witness pair $(x, w) \in \mathcal{R}$, and outputs a proof π.*
$\underline{\mathsf{Verify}(\mathsf{crs}, x, \pi)}$: *The verifier algorithm Verify takes as input a* crs, *a statement x together with a proof π, and outputs 1 or 0, indicating acceptance or rejection of the proof.*

These algorithms must satisfy the following properties:

Definition 4 (Perfect Completeness). *We say $\Pi = (\mathsf{Setup}, \mathsf{Prove}, \mathsf{Verify})$ has perfect completeness when for all $(x, w) \in \mathcal{R}$ it holds that*

$$\Pr\left[\begin{array}{l}\mathsf{Verify}(\mathsf{crs}, x, \pi) = 1; \mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda); \\ \pi \leftarrow \mathsf{Prove}(\mathsf{crs}, \mathsf{aux}, x, w)\end{array}\right] = 1.$$

Definition 5 (Knowledge-Soundness). *We say* $\Pi = (\mathsf{Setup}, \mathsf{Prove}, \mathsf{Verify})$ *is knowledge-sound with knowledge-error* $\kappa(\lambda)$ *if for every efficient prover* $\mathcal{P}$ *there exists an efficient extractor* $\mathcal{E}$ *such that for all efficient adversaries* $\mathcal{A}$ *it holds that*

$$\Pr\left[\begin{array}{c|l} \mathsf{Verify}(\mathsf{crs}, x, \pi) = 1 & \mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda) \\ \wedge\ (x, w) \notin \mathcal{R} & (\pi, x) \leftarrow \mathcal{A}(\mathsf{crs}) \\ & w \leftarrow \mathcal{E}(\mathsf{crs}, x, \pi) \end{array}\right] \leq \kappa(\lambda),$$

where $\mathcal{E}$ *has black-box access to each of the next-message functions of* $\mathcal{P}$*. We say* Π *has computational knowledge-soundness when* $\kappa(\lambda)$ *is negligible. When* $\mathcal{P}$ *and* $\mathcal{A}$ *are not required to be efficient, but* $\kappa(\lambda)$ *is still negligible, we say* Π *has statistical knowledge-soundness.*

In our GSIPA application, sometimes we deal with statements that live in an extended message space and thus our statements are of the form (z, aux) where one can use algorithm $\mathsf{VerifyMsg}(\cdot)$ to check whether z and aux yield a true extended statement.

Definition 6 (Extended Knowledge-Soundness). *We say* $\Pi = (\mathsf{Setup}, \mathsf{Prove}, \mathsf{Verify})$ *is extended knowledge-sound with error* $\kappa(\lambda)$ *if for every efficient prover* $\mathcal{P}$ *there exists an efficient extractor* $\mathcal{E}$ *such that for all efficient adversaries* $\mathcal{A}$ *it holds that*

$$\Pr\left[\begin{array}{c|c} \mathsf{Verify}(\mathsf{crs}, x, \pi) = 1 & \mathsf{crs}, \mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda) \\ \wedge\ (x, w) \notin \mathcal{R} & (\mathsf{st}) \leftarrow \mathcal{A}^{\mathcal{O}}(\mathsf{pp}) \\ & (\pi, x) \leftarrow \mathcal{A}(\mathsf{crs}, \mathsf{st}) \\ & w \leftarrow \mathcal{E}(\mathsf{crs}, x, \pi) \end{array}\right] \leq \kappa(\lambda),$$

where $\mathcal{O}$ *is a one-time oracle that takes a statement* z *and returns an extended statement* (z, aux) *and* $\mathcal{E}$ *has black-box access to each of the next-message functions of* $\mathcal{P}$*. We say* Π *has computational extended knowledge-soundness when* $\kappa(\lambda)$ *is negligible. When* $\mathcal{P}$ *and* $\mathcal{A}$ *are not required to be efficient, but* $\kappa(\lambda)$ *is still negligible, we say* Π *has statistical extended knowledge-soundness.*

Succinctness. The proof size should be sublinear in the size of the witness and, ideally, polylogarithmic or even constant-size.

2.4 Hadamard Product Argument (HPA)

We recall an efficient Hadamard product argument for $\mathbf{a} \circ \mathbf{b}$ due to Groth [37]. He introduced a framework for zero-knowledge arguments that reduces Hadamard products to inner product relations. This framework integrates randomization and batch-verification techniques to streamline a variety of linear algebra relations into inner product relations.

Briefly, the randomization technique proceeds as follows: consider a randomly chosen scalar $r \in \mathbb{F}$. Define a binary operator $*$, which maps $\mathbb{F}^n \times \mathbb{F}^n$ to $\mathbb{F}$, as $\mathbf{a} * \mathbf{b} = \mathbf{a}(\mathbf{b} \circ \mathbf{r})^\top$, where $\mathbf{r} = (1, r, r^2, \ldots, r^{n-1})$. An important property of this operator is the following: if $\mathbf{a} * \mathbf{b} = \mathbf{c} * \mathbf{d}$, then it holds that $\mathbf{a} \circ \mathbf{b} = \mathbf{c} \circ \mathbf{d}$ with error probability at most $\frac{n-1}{|\mathbb{F}|}$.

Employing this randomization technique enables the construction of HPA from IPA, with the relation expressed as $\langle \mathbf{a} \circ \mathbf{r}, \mathbf{b} \rangle = \langle \mathbf{c} \circ \mathbf{r}, \mathbf{d} \rangle$. Consequently, by leveraging efficient IPA protocols, one can establish efficient HPA protocols as well.

2.5 Functional Commitments

Definition 7 (Functional Commitment (FC) [51]). *Let* D *be a domain and consider linear functions* $\langle \cdot, \cdot \rangle \colon D^q \times D^q \to D$ *defined by* $\langle \mathbf{x}, \mathbf{m} \rangle = \sum_{i \in [q]} x_i \cdot m_i$ *with* $\mathbf{x} = (x_1, \ldots, x_q) \in D^q$ *and* $\mathbf{m} = (m_1, \ldots, m_q) \in D^q$*. A functional commitment scheme* FC *for* $(D, q, \langle \cdot, \cdot \rangle)$ *is a tuple of four PPT algorithms:*

$\underline{\mathsf{Setup}(1^\lambda, q)}$: The setup algorithm takes as input a security parameter $\lambda \in \mathbb{N}$ and a desired message length $q \in \mathrm{poly}(\lambda)$. It outputs a commitment key CK and, optionally, a trapdoor TK.

$\underline{\mathsf{Commit}(\mathsf{CK}, \mathbf{m})}$: The commit algorithm takes as input the commitment key CK and a message vector $\mathbf{m} \in D^q$. It outputs a commitment com for $\mathbf{m}$ and auxiliary information aux.

$\underline{\mathsf{Open}(\mathsf{CK}, \mathsf{com}, \mathsf{aux}, \mathbf{x})}$: The open algorithm takes as input the commitment key CK, a commitment com, auxiliary information (possibly containing $\mathbf{m}$), and a vector $\mathbf{x} \in D^q$. It computes a witness π_y for $y = \langle \mathbf{x}, \mathbf{m} \rangle$ as the linear function defined by $\mathbf{x}$ when evaluated on $\mathbf{m}$ gives y.

$\underline{\mathsf{Verify}(\mathsf{CK}, \mathsf{com}, \pi_y, \mathbf{x}, y)}$: Takes as input the commitment key CK, a commitment com, a witness π_y, a vector $\mathbf{x} \in D^q$, and $y \in D$. It outputs 1 if π_y is a witness for com being a commitment for some $\mathbf{m} \in D^q$ such that $\langle \mathbf{x}, \mathbf{m} \rangle = y$, and 0 otherwise.

One requires that it is impossible to generate a commitment along with valid proofs for two different function values $y \neq y'$ at a single input x, which is formalized as the function binding property.

Definition 8 (Function Binding [51]). *A functional commitment scheme* FC *is said to be function binding if any PPT adversary* $\mathcal{A}$ *the following probability is negligible:*

$$\Pr\left[\begin{array}{l}\mathsf{Verify}(\mathsf{CK}, \mathsf{com}, \pi_y, \mathbf{x}, y) = \\ \mathsf{Verify}(\mathsf{CT}, \mathsf{com}, \pi'_{y'}, \mathbf{x}, y') = 1 \\ \wedge\, y \neq y'\end{array} \middle| \begin{array}{l}(\mathsf{TK}, \mathsf{CK}) \leftarrow \mathsf{Setup}(1^\lambda, \mathcal{F}), \\ (\mathsf{com}, \mathbf{v}, y, y', \pi_y, \pi'_{y'}) \leftarrow \mathcal{A}(\mathsf{CK})\end{array}\right]$$

In this paper, we also consider two types of functional commitments that encompass earlier notions for specific types of functionalities, such as vector commitments [19] and polynomial commitments [41]. We define VC schemes as follows:

Definition 9 (Vector Commitment (VC) [19]). *A VC is a tuple of PPT algorithms defined as follows:*

$\underline{\mathsf{Setup}(1^\lambda, q)}$: The setup algorithm takes a security parameter λ and the size q of the committed vector. It outputs some public parameters pp (which implicitly define the message space $\mathcal{M}$ and are input to all algorithms).

$\underline{\mathsf{Commit}(m_1, \ldots, m_q)}$: The commitment algorithm takes as input a vector of q messages $(m_1, \ldots, m_q) \in \mathcal{M}$ and the public parameters pp. It outputs a commitment com and auxiliary information aux.

$\underline{\mathsf{Open}(m, i, \mathsf{aux})}$: The open algorithm takes as input a message m, index i and auxiliary information aux. It outputs a proof π_i that m is the i-th committed message.

$\underline{\mathsf{Verify}(\mathsf{com}, m, i, \pi_i)}$: The verification algorithm takes as input commitment com, message m, index i and proof π. It outputs 1 if π_i is a valid proof that com commits to a message such that $m = m_i$ and 0 otherwise.

For security one requies that an adversary cannot produce two proofs for the same position in a fixed commitment com that open to different values.

Definition 10 (Position Binding [19]). *A* VC *satisfies position binding if for all* $i \in [q]$ *and for every PPT adversary* $\mathcal{A}$*, the following probability (taken over all honestly generated* pp*) is at most negligible* $\epsilon(\lambda)$ *:*

$$\Pr\left[\begin{array}{c}\mathsf{Verify}(\mathsf{com}, m, i, \pi) = 1 \wedge \\ \mathsf{Verify}(\mathsf{com}, m', i, \pi') = 1 \\ \wedge\, m \neq m'\end{array} \middle| ((\mathsf{com}, \mathsf{aux}), m, m', i, \pi, \pi') \leftarrow \mathcal{A}(\mathsf{pp})\right]$$

Conciseness: This property requires that both the commitment and the opening for a position i be independent of the vector's length, ideally of logarithmic or even constant size.

2.6 Structure Preserving Vector Commitments

We recall the structure preserving vector commitment (SPVC) scheme from [45] over a message space $\mathbf{M} \in \mathbb{G}_1$ (or $\mathbb{G}_2$) based on the q-DHE assumption. They introduce a structured message space built on the q-DHE parameters, a CRS as follows:

Message Structure. Each message consists of vectors with $q+1$ elements over $\mathbb{G}_1$, where each vector includes a main message component $\mathbf{M}$ of length 2, as well as a tag vector $\mathbf{T}$ with $q-1$ elements. The structure of the vector and the definition of the message space $\mathcal{M}^{\mathsf{pp},q}$ are determined by the q-DHE parameters. The message space $\mathcal{M}^{\mathsf{pp},q}$ is defined according to these parameters and we denote by MsgGen the algorithm that extends $M_{i0} = P^{m_i}$ to an element of the message space.

$$\mathcal{M}^{\mathsf{pp},q} = \left\{\begin{array}{l}((\mathbf{M}_1, \mathbf{T}_1), \ldots, (\mathbf{M}_q, \mathbf{T}_q)) \mid \exists \mathbf{m} = (m_1, \ldots, m_q) \in \mathbb{Z}_p^q \text{ s.t.} \forall i \in [q], \\ msg_i = (M_{i0} = P^{m_i}, M_{i1} = B_i^{m_i} = P^{\alpha^i \cdot m_i}, T_{ij} = B_j^{m_i} = P^{\alpha^j \cdot m_i}) \\ \text{where } j \in [i+1, i+q] \setminus \{q+1\}\end{array}\right\} \quad (1)$$

Definition 11 (q-DHE SPVC [45]). *The q-DHE SPVC scheme is defined as follows:*

$\underline{\mathsf{Setup}(1^\lambda, q)}$: Given the security parameter λ and the size q of the committed vector, the key generation picks $\alpha \leftarrow\!\!\$\ \mathbb{Z}_p$ and outputs some public parameters:

$$\mathsf{pp} = \begin{pmatrix} \mathsf{BG}, B_1 = P^{\alpha^1}, \ldots, B_q = P^{\alpha^q}, B_{q+2} = P^{\alpha^{q+2}}, \ldots, B_{2q} = P^{\alpha^{2q}}; \\ \hat{B}_1 = \hat{P}^{\alpha^1}, \ldots, \hat{B}_q = \hat{P}^{\alpha^q}; g_t^{\alpha^{q+1}} \end{pmatrix}$$

where $g_t = e(P, \hat{P})$.

$\underline{\mathsf{Commit}(msg_1, \ldots, msg_q)}$: On input a vector of q messages $(msg_1, \ldots, msg_q) \in \mathcal{M}^{\mathsf{pp},q}$, check if messages are correctly generated via $1 \leftarrow \mathsf{VerifyMsg}(\mathsf{pp}, msg_i)$ for all $i \in [q]$. Pick $r \leftarrow \mathbb{Z}_p^*$ and output a commitment com and auxiliary information aux:

$$\mathsf{com} = \left(P^r \cdot \prod_{i \in [q]} M_{i1} \right) \ \wedge \ \mathsf{aux} = r$$

$\underline{\mathsf{Open}(msg, i, \mathsf{aux})}$: This algorithm is run by the committer to produce a proof π_i that msg is the i-th committed message:

$$\pi_i = \left(B_{q+1-i}^r \cdot \prod_{j \in [q] \setminus \{i\}} T_{j,q+1-i+j} \right)$$

$\underline{\mathsf{Verify}(\mathsf{com}, msg, i, \pi_i)}$: The verification algorithm accepts (*i.e.,* it outputs 1) if and only if the messages are well-formed (*i.e.,* $1 \leftarrow \mathsf{VerifyMsg}(\mathsf{pp}, msg_i)$ for all $i \in [q]$), and also π_i is a valid proof that com was created for msg being the i-th committed message:

$$e\left(\mathsf{com}, \hat{B}_{q+1-i}\right) = e\left(\pi_i, \hat{P}\right) \cdot e\left(M_{i1}, \hat{B}_{q+1-i}\right)$$

$\underline{\mathsf{VerifyMsg}(\mathsf{pp}, msg)}$: Takes a message $msg_i = (\mathbf{M} = (M_{i0}, M_{i1}), \mathbf{T} = (T_{ij}))$ and verifies whether it is well-formed with respect to the pp. It returns 1 if the following equation holds, and 0 otherwise.

$$e\left(M_{i1}, \hat{B}_{j-i}\right) = e\left(T_{ij}, \hat{P}\right) \wedge e\left(M_{i1}, \hat{P}\right) = e\left(M_{i0}, \hat{B}_i\right)$$

where $j \in [i+1, i+q] \setminus \{q+1\}$.

2.7 Accountable Multi-Signatures

Our construction from Section 5.2 is based on the accountable multi-signature from [12]. We adapt the original syntax and unforgeability definitions from [12] in the vein of [33], as used in this work.

Definition 12 (Accountable Multi-Signature (AMS)). *An Accountable Multi-Signature scheme scheme consists of the following polynomial-time algorithms:*

$\underline{\mathsf{Setup}(1^\lambda, q)}$: *On input of the security parameter $\lambda \in \mathbb{N}$ and the parameter q, the setup algorithm outputs public parameters* pp*, which are treated as an implicit input to all algorithms.*

$\underline{\mathsf{KGen}(i)}$: *On input the index i, the key generation algorithm outputs a public/secret key pair* $(\mathsf{pk}_i, \mathsf{sk}_i)$.

$\underline{\mathsf{HintGen}(\mathsf{sk}, q)}$: *On input* pp*, a secret key* sk*, and the number of parties q, the hint generation algorithm outputs a hint* hint.

$\underline{\mathsf{Preprocess}(\{(\mathsf{hint}_i, \mathsf{pk}_i)\}_{i \in [q]})}$: *On input* pp *and all pairs* $\{(\mathsf{hint}_i, \mathsf{pk}_i)\}_{i \in [q]}$*, the preprocessing algorithm computes an aggregation key* AK *and a (succinct) verification key* vk.

$\underline{\mathsf{Sign}(\mathsf{sk}, m)}$: *On input a secret key* sk *and a message m, the signing algorithm outputs a partial signature σ.*

$\underline{\mathsf{SigAggr}(\mathsf{AK}, \mathsf{vk}, J, \{\sigma_j, \mathsf{pk}_j\}_{j \in J})}$: *On input* pp*, an aggregation key* AK*, and a set of partial signatures and public keys $\{\sigma_j, \mathsf{pk}_j\}_{j \in J}$, the signature aggregation algorithm outputs a (succinct) aggregated signature σ.*

$\underline{\mathsf{Verify}(\mathsf{vk}, m, \sigma, J)}$: *On input the verification key* vk*, a message m, a signature σ, and a subset J, it verifies the signature and outputs a bit $b \in \{0, 1\}$ denoting acceptance or rejection.*

$\underline{\mathsf{Trace}(\mathsf{vk}, m, \sigma)}$: *On input the verification key* vk*, a message m and a signature σ, the tracing algorithm outputs the subset J that generated σ.*

Definition 13 (Unforgeability of AMS). *An AMS scheme satisfies unforgeability if, for any adversary $\mathcal{A}$, the output of the game shown below is* 1 *with probability* $\leq \mathsf{negl}(\lambda)$.

- *The adversary picks n to set the maximum bound on the number of parties.*
- *The challenger runs* $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda, n)$ *and gives* pp *to* $\mathcal{A}$.
- *The adversary picks a user to attack* $i^* \leftarrow \mathcal{A}(\mathsf{crs})$.
- *The challenger samples the public key and hint honestly by* $(\mathsf{pk}_i^*, \mathsf{sk}_i^*) \leftarrow \mathsf{KGen}(1^\lambda)$ *and* $\mathsf{hint}^*{}_i = \mathsf{HintGen}(\mathsf{crs}, \mathsf{sk}_i^*, n)$ *and gives* $(\mathsf{pk}_i^*, \mathsf{hint}^*{}_i)$ *to* $\mathcal{A}$.
- *For all other users, the adversary picks a public key* pk_i *and the corresponding hint* hint_i.
- *The adversary may make a partial signature query with a message m for* i^*, *the query is returned by computing* $\sigma_{i^*} = \mathsf{Sign}(\mathsf{sk}^*, m)$. *This is run as many times as desired by* $\mathcal{A}$.
- *Finally, the adversary shall output a challenge message* m^* *and an aggregated signature* σ^* *alongside with* $(\{\mathsf{pk}_i, \mathsf{hint}_i\}_{i\in[q]\setminus i^*})$.
- *The challenger preprocesses* $\{\mathsf{hint}_i, \mathsf{pk}_i\}_{i\in[q]}$ *as* $(\mathsf{AK}, \mathsf{vk}) \leftarrow \mathsf{Preprocess}(\{\mathsf{hint}_i, \mathsf{pk}_i\}_{i\in[q]})$.
- *The adversary wins the forgery game if* $\mathsf{Verify}(m^*, \sigma^*, J^*, \mathsf{vk}) = 1$ *and* $J^* \subset \{1, \ldots, q\}$ *includes* i^*, *in which case, the output is* 1.

3 Novel Functional Commitments

We now introduce a novel FC scheme for group-scalar inner product relations, denoted GSFC. As shown in [51], inner products encompass a range of primitives, including vector and polynomial commitments and serves as a foundation for inner product arguments (IPA), as detailed in Sec. 4.

3.1 Group-Scalar FC for Inner Products

Subsequently, we present the first FC scheme for inner-product relations of the form $\langle \mathbf{A}, \mathbf{b} \rangle$ with constant-size proofs and verification time, while ensuring that the commitment/proof remains in the source group.

Our scheme is inspired by the structure-preserving vector commitment SPVC in [45] (cf. Sec.2.6), particularly in terms of the message space and commitments. However, we significantly adapt the proof generation and verification process to turn it into an FC scheme and support inner-product relations. We combine SPVC with an aggregation technique inspired by [36] to embed the vector $\mathbf{b}$ into a single aggregated proof. However, since this method applies to independent scalar messages under the q-DHE assumption, it is not sufficient on its own. To support a single group element $Y = \langle \mathbf{A}, \mathbf{b} \rangle$, we integrate the approach from [18] to compute Y, enforce a degree bound, and establish security under the q-DL assumption. Conceptually, we shift the sum $\sum b_i \cdot m_i$ from an exponent of an element based in $\mathbb{G}_T$, i.e., $g_t^{\sum b_i \cdot m_i}$, into a group element $Y \in \mathbb{G}_1$.

Strong Function Binding. We present our scheme in the strong function binding model, similar to [18,46], but adapted to our setting to be more suitable for our GSFC schemes. This definition is indeed strictly stronger than the function binding definition from [51] as presented in Definition 8.

Definition 14 (Strong Function Binding). *FC satisfies* strong function binding *if, for any PPT adversary $\mathcal{A}$, for all $\lambda \in \mathbb{N}$, and for any family of functions f, there exists a negligible function $\epsilon(\lambda)$ such that:*

$$\Pr\left[\mathsf{Verify}(\mathsf{CK}, \mathsf{com}, \pi_y, \mathbf{b}, y) = 1 \wedge f(\mathbf{b}) \neq y\right] \leq \epsilon(\lambda),$$

where $(\mathsf{CT}, \mathsf{TK}) \leftarrow \mathsf{Setup}(1^\lambda, q)$ *and* $(\mathsf{com}, f, y, \mathbf{b}, \pi_y) \leftarrow \mathcal{A}(\mathsf{CT})$.

In our construction, we consider linear functions $\mathbb{Z}_p^q \rightarrow \mathbb{G}_1$, which means that $f(\mathbf{b}) = Y$, *i.e.*, $f(\mathbf{b}) = \langle \mathbf{A}, \mathbf{b} \rangle = Y$ for a vector $\mathbf{A} \in \mathbb{G}_1^q$ describing the function space while $\mathbf{b}$ is input to the functions and Y is a group element.

Scheme 1 (Group-Scalar FC for Inner Products (GSFC)) *Lets* SPVC *be the scheme in Def.11, a* GSFC *for inner products scheme is a tuple of four PPT algorithms defined as follows:*

$\underline{\mathsf{Setup}(1^\lambda, q)}$: Takes a security parameter $\lambda \in \mathbb{N}$, a desired message length $q \in \mathrm{poly}(\lambda) < p$, and outputs a commitment key CK and, optionally, a trapdoor TK. It computes CK as

$$\begin{pmatrix} B_1 = P^{\alpha^1}, B_2 = P^{\alpha^2}, \ldots, B_q = P^{\alpha^q}, B_{q+2} = P^{\alpha^{q+2}}, \ldots, B_{2q} = P^{\alpha^{2q}}; \\ \hat{B}_1 = \hat{P}^{\alpha^1}, \ldots, \hat{B}_{2q} = \hat{P}^{\alpha^{2q}};\ \mathcal{M}^{\mathsf{pp},q}; \mathsf{BG} \end{pmatrix}$$

and the trapdoor is set to be $\mathsf{TK} = P^{\alpha^{q+1}}$.

$\underline{\mathsf{Commit}(\mathsf{CK}, \mathbf{msg})}$: Takes as input the commitment key CK, a message vector $\mathbf{msg} = (\mathbf{M}_0, \mathbf{M}_1, \mathbf{T}) \in \mathcal{M}^{\mathsf{pp},q}$, and outputs a commitment com and auxiliary information aux. It runs and outputs $(\mathsf{com}, r) \leftarrow \mathsf{SPVC.Commit}(\mathsf{CK}, \mathbf{msg})$ where

$$\mathsf{com} = \left(C_1 = P^r \cdot \prod_{i \in [q]} M_{i1} \right) \ \wedge\ \mathsf{aux} = (r, \mathbf{msg}).$$

$\underline{\mathsf{Open}(\mathsf{CK}, \mathsf{com}, \mathsf{aux}, \mathbf{b})}$: Takes as input the commitment key CK, a commitment com (to **msg**), auxiliary information aux, and a vector $\mathbf{b} \in \mathbb{F}^q$. It computes a witness π_y for $Y = \langle \mathbf{M}_{i0}, \mathbf{b} \rangle \in \mathbb{G}_1$, *i.e.*, π_y is a witness for the fact that the linear function defined by $\mathbf{b}$ when evaluated on $\mathbf{M}_0$ gives Y. It runs $\pi_i \leftarrow \mathsf{SPVC.Open}(\mathsf{CK}, msg_i, i, \mathsf{aux})$ for all msg_i where $i \in [q]$ such that a witness π_i for msg_i is of the form

$$\pi_i = \left(B_{q+1-i}^r \cdot \prod_{j \in [q] \setminus \{i\}} T_{j,q+1-i+j} \right).$$

Finally it outputs $\pi_y = \prod_{i=1}^q \pi_i^{b_i}$ as witness for Y.

$\underline{\mathsf{Verify}(\mathsf{CK}, \mathsf{com}, \pi_y, \mathbf{b}, Y)}$: Takes as input the commitment key CK, a commitment com, a witness π_y, a vector $\mathbf{b} \in \mathbb{Z}_p^q$, and $Y = \langle \mathbf{M}_{i0}, \mathbf{b} \rangle = \prod_{i=1}^q M_{i0}^{b_i} = P^{\sum b_i \cdot m_i} \in \mathbb{G}_1$. It outputs 1 if

$$e\left(\mathsf{com}, \prod_{i=1}^q \hat{B}_{q+1-i}^{b_i} \right) = e\left(\pi_y, \hat{P} \right) \cdot e\left(Y, \hat{B}_{q+1} \right),$$

i.e., π_y is a witness for com being a commitment such that $\langle \mathbf{b}, \mathbf{M} \rangle = Y$, and 0 otherwise.

Theorem 1. *Scheme 1 is strong function binding (Def. 14) in the AGM assuming the q-DL assumption holds (Def. 2).*

Proof. We provide the proof in Appendix C.1.

3.2 Polynomial Commitments

We show that our GSFC scheme enables a new form of polynomial commitment scheme (PCS) for values hidden inside group exponentiations [32] (also referred to as *encapsulated values*). Specifically, one considers a polynomial $f(X) = m_0 + m_1 \cdot X + \cdots + m_{q-1} \cdot X^{q-1}$ for some $m_0, m_1, \ldots, m_{q-1} \in \mathbb{Z}_p$, where the prover does not know the coefficients m_i in the clear, but only has access to group elements carrying them in their exponents, *i.e.*, $(P^{m_0}, P^{m_1}, \ldots, P^{m_{q-1}})$. For this form, we present a succinct PCS that works directly on these encapsulated values, allowing the prover to commit to the hidden polynomial coefficients implicitly and later generate succinct evaluation proofs for any point $x \in \mathbb{Z}_p$.

Our PCS: The framework from [51] shows that a functional commitment for linear functions (inner products) implies a PCS: To commit to a polynomial $f(X) = m_0 + m_1 \cdot X + \cdots + m_{q-1} \cdot X^{q-1}$ of degree $q-1$, we can simply commit to the vector containing the coefficients $\mathbf{m} = (m_0, m_1, \ldots, m_{q-1}) \in \mathbb{Z}_p^q$. When the sender wants to convince a verifier that $f(x) = y$, for some public (x, y), it is sufficient to generate a witness π_y showing that $\langle \mathbf{m}, \mathbf{x} \rangle = y$, where $\mathbf{x} = (1, x, x^2, \ldots, x^{q-1})$ and y can also be encoded in group exponentiation as $Y = P^y$. This PCS can be seen as a concrete instantiation of the more general framework of Polynomial Commitments for Hidden Values introduced by [32], tailored to the specific case of linear homomorphic encapsulation via group exponentiation.[5]

Our construction is based on the GSFC scheme and uses the framework of [32] to obtain the first PCS from group elements. Specifically, we describe a PCS based on GSFC as follows.

[5] Note that here we only consider computational hiding.

Scheme 2 (PCS for Encapsulated Values) *Let* GSFC *be the scheme in Def. 1, our PCS for encapsulated values is defined as follows:*

$\underline{\mathsf{Setup}(1^\lambda, q)}$: Takes a security parameter $\lambda \in \mathbb{N}$, a desired message length $q \in \text{poly}(\lambda) < p$ representing polynomials of degree $q-1$. It runs $(\mathsf{CK}, \mathsf{TK}) \leftarrow \mathsf{GSFC.Setup}(1^\lambda, q)$ and outputs a commitment key CK and, optionally, a trapdoor TK.

$\underline{\mathsf{Commit}(\mathsf{CK}, \mathbf{msg})}$: Takes as input the commitment key CK, a message vector $\mathbf{msg} = (\mathbf{M}_0, \mathbf{M}_1, \mathbf{T}) \in \mathcal{M}^{\mathsf{pp},q}$ as evaluation points, and outputs a commitment com and auxiliary information aux for a polynomial $f(X) = \sum_{i=0}^{q-1} m_i \cdot X^i$ by running $(\mathsf{com}, \mathsf{aux}) \leftarrow \mathsf{GSFC.Commit}(\mathsf{CK}, \mathbf{msg})$, where coefficients are encoded as $(P^{m_i})_{i=0}^{q-1} \in \mathbb{G}_1^q$. Finally, it outputs $(\mathsf{com}, \mathsf{aux})$.

$\underline{\mathsf{Open}(\mathsf{CK}, \mathsf{com}, \mathsf{aux}, x)}$: Takes as input the commitment key CK, a commitment com (to $\mathbf{M}$ or analogously $f(x)$), auxiliary information aux, and the evaluation point x. It present the point x as a vector of the form $\mathbf{x} = (1, x, x^2, \ldots, x^q)$ and computes a witness π_y for $Y = f(x)$ by running $(\pi_y, Y) \leftarrow \mathsf{GSFC.Open}(\mathsf{CK}, \mathsf{com}, \mathsf{aux}, \mathbf{x})$, where $Y = P^y$.

$\underline{\mathsf{Verify}(\mathsf{CK}, \mathsf{com}, \pi_y, x, Y)}$: Takes as input the commitment key CK, a commitment com, a witness π_y, and evaluations points (x, Y). It converts x to a vector $\mathbf{x} = (1, x, x^2, \ldots, x^q) \in \mathbb{Z}_p^q$ and outputs $\{1, 0\} \leftarrow \mathsf{GSFC.Verify}(\mathsf{CK}, \mathsf{com}, \pi_y, \mathbf{x}, Y))$.

Evaluation Binding. The proof of evaluation binding follows immediately from the function binding of the GSFC scheme.

It is clear, our polynomial commitment construction directly implies a succinct PCS for hidden values in the sense of [32], where we instantiate LHEncap as group exponentiation. However, compared to the scheme in [32], which relies on the FRI protocol [5] to ensure proximity with a *logarithmic size*, our construction avoids such tools and achieves *constant-size* commitments and proofs, while maintaining a similar structure (see Sec. 5.3 for a more detailed discussion).

4 IPA for Group-Scalar Relations

An inner product argument (IPA) is a (non-)interactive argument of knowledge protocol between a PPT prover $\mathcal{P}$ and verifier $\mathcal{V}$ such that the prover demonstrates that for $(x, w) \in \mathcal{R}$, where $x = (y, \mathsf{com}_\mathbf{a}, \mathsf{com}_\mathbf{b}), w = (\mathbf{a}, \mathbf{b})$ we have $\langle \mathbf{a}, \mathbf{b} \rangle = y$. Given two vectors $\mathbf{a}$ and $\mathbf{b}$, an IPA enables $\mathcal{P}$ to convince $\mathcal{V}$ that $\langle \mathbf{a}, \mathbf{b} \rangle = Y$, where the verifier only has access to the commitments $\mathsf{com}_\mathbf{a}$ and $\mathsf{com}_\mathbf{b}$ of $\mathbf{a}$ and $\mathbf{b}$, respectively. We present the first IPA for group-scalar relations (GSIPA), where the inner product is defined as $\langle \mathbf{A}, \mathbf{b} \rangle = \prod_{i=1}^q A_i^{b_i}$, while ensuring that both the proof size and verification time remain constant.

GSFC vs GSIPA. While both GSFC and GSIPA address group-scalar inner product relations, their technical roles and security guarantees differ fundamentally. The GSFC scheme allows a prover to commit to a vector of group elements and later produce a succinct proof that a specific linear function, defined by a public function vector—is correctly evaluated on the committed data. Here, the security goal is *function binding*, and knowledge soundness is not required. That is, GSFC does not constitute an argument of knowledge, and the committed vector cannot, in general, be extracted from the proof. In contrast, GSIPA is a *proof system* designed for settings where the prover commits to both vectors involved in the inner product relation. Consequently, there must exist an extractor that can *extract both* committed vectors from a valid proof, requiring a substantially stronger notion of *knowledge soundness*. To meet this stronger requirement, the GSIPA construction extends GSFC by introducing an additional verification equation that checks the degree structure of the committed polynomials, thereby ensuring soundness and extractability.

Construction. Our construction is based on the GSFC scheme in Sect. 3.1 and the monomial basis construction for encoding vectors by [18], with some simple observations. First, GSFC considers vectors encoded as polynomials in the monomial basis *e.g.*, $\mathbf{m} \in \mathbb{Z}_p^q$ that is, $m(X) = \sum_{i=1}^q m_i X^i$. In other words, considering the verification equation, we can express it as $\left(\sum_{i=1}^q a_i X^i\right)\left(\sum_{i=1}^q b_i X^{q+1-i}\right) = yX^{q+1} + R(X)$, where $R(X)$ is a polynomial of maximum degree $2q$ that must be determined and is related to the proof π_y. Since we the values for $P^{\alpha^1}, \ldots, P^{\alpha^{2q}}$ in the commitment key, we can efficiently compute this.

Another key observation in GSFC is that during verification, the verifier only needs to compute $\prod_{i=1}^{q} \hat{B}_{q+1-i}^{b_i}$, which corresponds to the polynomial $\sum_{i=1}^{q} b_i X^{q+1-i}$. We note that this is essentially a commitment to a monic polynomial with reversed exponent order. Here, instead of sending $\mathbf{b}$ such that the verifier can compute $\hat{C}_{\mathbf{b}} = \prod_{i=1}^{q} \hat{B}_{q+1-i}^{b_i}$, we send it as a commitment to the vector $\mathbf{b}$. We can recognize that this commitment can now be generated by an adversary in the context of soundness.

As the vector $\mathbf{b}$ and commitment $\hat{C}_{\mathbf{b}}$ are now computed by the prover (and not by verifier as $\mathbf{b}$ is not a public vector anymore), to ensure the soundness in the proof, we need to verify both the form and degree of the polynomial related to this commitment $\hat{C}_{\mathbf{b}}$. Therefore, we introduce an additional proof $\hat{\pi}$ and a verification equation that allow us to check this efficiently. Indeed, in the proof of knowledge-soundness, this enables us to extract the witness. We now present our scheme.

Scheme 3 (Inner Product Argument for Group-Scalar Elements (GSIPA)) *The* GSIPA *protocol is a non-interactive proof between a PPT prover $\mathcal{P}$ and verifier $\mathcal{V}$. Given two vectors $\mathbf{A} \in \mathbb{G}_1^q$ and $\mathbf{b} \in \mathbb{Z}_p^q$, the* IPA *allows $\mathcal{P}$ to convince $\mathcal{V}$ that $\langle \mathbf{A}, \mathbf{b} \rangle = Y$ as follows:*

$\underline{\mathsf{Setup}(1^\lambda) \to \mathsf{crs}:}$ Takes as input a security parameter $\lambda \in \mathbb{N}$ and outputs a common reference string crs, which is the same as the commitment key for GSFC.

$\underline{\mathsf{Prove}(\mathsf{crs}, \mathsf{aux}, x, w):}$ Takes as input crs, $\mathsf{aux} = (\mathbf{M}_1, \mathbf{T}) \in \mathcal{M}^{\mathsf{pp},q}$, and an instance consisting of $x = (Y \in \mathbb{G}_1, C_{\mathbf{A}}, \hat{C}_{\mathbf{b}})$ and witness $w = (\mathbf{A}, \mathbf{b})$ for $Y = \langle \mathbf{A}, \mathbf{b} \rangle$ with $\mathbf{A} = \mathbf{M}_0 = (P^{m_i})_{i \in [q]} \in \mathcal{M}^{\mathsf{pp},q}$. Let $\mathbf{msg} = (\mathbf{A}, \mathsf{aux})$, the prover proves the following IP relations with $\langle \mathbf{A}, \mathbf{b} \rangle = \prod A_i^{b_i} = Y$ and Commit being the commitment function implicitly computed in GSFC.Verify (cf. Scheme 1):

$$\mathcal{R}_{\mathsf{IP}} := \{(\mathbf{A}, \mathbf{b}) : C_{\mathbf{A}} = \mathsf{GSFC.Commit}(\mathsf{crs}, \mathbf{msg}) \ \wedge \ \hat{C}_{\mathbf{b}} = \mathsf{Commit}(\mathsf{crs}, \mathbf{b}) \ \wedge \ \langle \mathbf{A}, \mathbf{b} \rangle = Y\}.$$

– First, we recall that the commitments that are part of the statement x for witnesses $(\mathbf{A}, \mathbf{b})$ have been computed as follows:

$$C_{\mathbf{A}} = P^r \cdot \prod_{i \in [q]} M_{i1} \ \wedge \ \hat{C}_{\mathbf{b}} = \prod_{i \in [q]} \hat{B}_{q+1-i}^{b_i}.$$

– Now, it creates a proof $\hat{\pi}$ for a polynomial $\hat{R}(X) = \sum_{j \in [q]} \sum_{i \in [q]} b_i X^{q+1-i+j}$ such that $\deg(R(X)) \leq 2q$ as:

$$\hat{\pi} = \prod_{j \in [q]} \prod_{i \in [q]} \hat{B}_{q+1-i+j}^{b_i} = \hat{P}^{\sum_{j \in [q]} \sum_{i \in [q]} b_i \alpha^{q+1-i+j}} = \hat{P}^{\hat{R}(\alpha)}.$$

Also, it computes a batched opening for $C_{\mathbf{A}}$ similar to the one in the GSFC scheme

$$\pi_y = \prod_{i=1}^{q} \left(B_{q+1-i}^r \cdot \prod_{j \in [q] \setminus \{i\}} T_{j,q+1-i+j} \right)^{b_i}.$$

Finally it outputs $\pi = (\pi_y, \hat{\pi})$ as proof for $Y = \langle \mathbf{A}, \mathbf{b} \rangle$.

$\underline{\mathsf{Verify}(\mathsf{crs}, x, \pi):}$ On input of crs and statement x containing Y (and we also assume that it contains commitments $C_{\mathbf{A}}$ and $\hat{C}_{\mathbf{b}}$) and proof $\pi = (\pi_y, \hat{\pi})$, this algorithm checks the inner product Y to committed $\mathbf{A}$ and $\mathbf{b}$ as follows:

$$e\left(C_{\mathbf{A}}, \hat{C}_{\mathbf{b}}\right) = e\left(\pi_y, \hat{P}\right) \cdot e\left(Y, \hat{B}_{q+1}\right) \ \wedge \ e\left(\prod_{j \in [q]} P^{\alpha^j}, \hat{C}_{\mathbf{b}}\right) = e\left(P, \hat{\pi}\right).$$

If so, it outputs 1 and 0 otherwise.

We note that $\prod_{j \in [q]} P^{\alpha^j}$ can be precomputed in the setup and placed in crs. Moreover, for simplicity, we show the computation of the commitments $C_{\mathbf{A}}$ and $\hat{C}_{\mathbf{b}}$ as part of the Prove algorithm, alongside the generation of the proof. However, these commitments are logically part of the statement, not the proof itself. They can alternatively be computed separately and sent to the verifier in advance, as part of the statement which is input to the verification algorithm.

Theorem 2 (GSIPA **Knowledge Soundness in the AGM).** *If* GSFC *is a computationally binding inner product commitment, then* GSIPA *(Scheme 3) is computationally knowledge-sound (Def. 5) in the AGM (including the variant which satisifes extended knowledge-soundness as defined in Def, 6).*

Proof. We provide the proof in Appendix C.2.

Zero-Knowledge Variant. The above argument can be modified to achieve zero-knowledge. One just needs to add randomness to the commitments and appropriately randomize the proofs with respect to this randomness. We leave the details to the reader, as zero-knowledge is not required for our applications.

Reduce Trust in the CRS. Our scheme requires a structured setup similar to KZG-based schemes, making it compatible with standard trust-reduction techniques for generating the crs such as MPC-based setup [6], updatable reference strings [39], and the widely used powers-of-tau ceremony [59]. In particular, updatable CRS techniques allow any party to securely refresh the CRS without compromising soundness, as long as at least one update is honest and thus significantly reducing trust. We note that after such a CRS update, the message space must be adjusted to align with the new CRS. Since the setup typically needs to be performed only once or infrequently, and efficient trust-reduction mechanisms exist, our focus is primarily on minimizing proof size and verification cost.

4.1 Subvector Openings

We show that both commitments, $C_{\mathbf{A}}$ and $\hat{C}_{\mathbf{b}}$, are indeed vector commitments with (sub)vector openings, *i.e.,* one can open multiple positions of the vector at the same time. In this case, in addition to the inner product relation, one can also open to (sub)vectors. For the commitment $C_{\mathbf{A}}$, since it follows the SPVC scheme, its opening procedure is exactly as defined in SPVC commitments (cf. Sec. 2.6).

Furthermore, we observe that $\hat{C}_b = \prod_{j=1}^{q} \hat{B}_{q+1-j}^{b_j}$ as commitment to $\mathbf{b}$ is a VC similar to mercurial commitments in [52,36] but with a different index for shifting. More precisely, one can encode messages using the monomial basis with inverse order exponentiation, then compute a proof as $\pi_i = \prod_{j\neq i}^{q} P^{b_j \alpha^{q+1+i-j}}$, where the verification would be $e(P^{\alpha^i}, \hat{C}_b) = e(\pi_i, \hat{P}) \cdot e(P, \hat{P})^{\alpha^{q+1} b_i}$. This is a valid vector commitment based on a generalized version of q-DHE (see Sec. 2.2). One can see that we can aggregate and batch proofs like other mercurial-type vector commitments. We note that the proof must be in $\mathbb{G}_1$, as otherwise binding does not hold.

Proof of Binding. Similar to the mercurial commitments in [52,36], we prove the binding of the above scheme under the generalized q-DHE assumption.

Proof. Assume that an adversary $\mathcal{A}$ produces a commitment $(\mathsf{com}, \mathsf{aux})$, an index $i \in \{1, \ldots, q\}$, and two valid openings π_i and π_i' to distinct messages $b_i \neq b_i'$. From the verification equations, it follows that:

$$e(\pi_i, \hat{P}) \cdot e(P, \hat{P}^{\alpha^{q+1}})^{b_i} = e(\pi_i', \hat{P}) \cdot e(P, \hat{P}^{\alpha^{q+1}})^{b_i'}$$

Simply rewriting yields $e(\pi_i / \pi_i', \hat{P}) = e(P, \hat{P}^{\alpha^{q+1}})^{b_i' - b_i}$, or equivalently $e((\pi_i / \pi_i')^{1/(b_i' - b_i)}, \hat{P}) = e(P, \hat{P}^{\alpha^{q+1}})$. Since $b_i \neq b_i'$, the latter relation implies that $P^{\alpha^{q+1}} = (\pi_i / \pi_i')^{1/(b_i' - b_i)}$ is revealed by the collision, which contradicts the generalized q-DHE assumption. □

Aggregation: Similar to [36], we can aggregate proofs and present a subvector opening. Suppose the committer wants to reveal multiple values $\{b_i : i \in S\}$ (where $S \subseteq [q]$) for a single commitment $\hat{C}_{\mathbf{b}}$ via a very short proof π_S. The idea is to take $t_i \leftarrow\!\!\$\ \mathbb{Z}_p^*$: $\pi_S = \prod_{i\in S} \pi_i^{t_i}$ which can, in turn, be verified using the equation:

$$e\left(\prod_{i\in S} P^{\alpha^i \cdot t_i}, \hat{C}_{\mathbf{b}}\right) \stackrel{?}{=} e(\pi_S, g_2) \cdot g_T^{\alpha^{q+1} \sum_{i\in S} b_i \cdot t_i}.$$

The scalars t_i can be computed via a hash function (cf. [36]).

5 Applications

We present several applications. First, we propose the notion of dynamic (weighted) threshold VRFs (DT-VRFs), which unlike existing schemes avoid the need for a costly DKG. DT-VRFs enable a non-interactive setup and support dynamic thresholds, allowing parties to join in an ad-hoc manner without any synchrony assumptions. Second, we show how to construct efficient silent threshold signatures [24,33] that support dynamic and weighted configurations with efficiency comparable to prior work, but without relying on random oracles. Finally, we demonstrate how our GSIPA, and in particular our PCS scheme from Sec. 3.2, can be applied in the context of oblivious proofs [32]—a framework for proving statements about hidden secrets via linearly homomorphic encapsulations. Our PCS achieves similar functionality *without FRI* [5], supporting *constant-size proofs and verification time* for encapsulations such as $\mathsf{LHEncap}(m) = P^m$ (*e.g.*, group exponentiations).

5.1 Dynamic Threshold (Weighted) VRFs

We recall from the introduction that in distributed or threshold VRFs (tVRFs) [30,22,43], parties first run a DKG protocol to compute shares of a secret key. Each party can then compute a partial evaluation using their secret share, and any set of $t+1$ valid partial evaluations can be combined to obtain the global evaluation of the VRF. However, tVRFs remain costly for certain use cases, such as randomness beacons, due to the frequent need to run expensive DKG protocols-typically need $O(n^2)$ and $O(n^3)$ communication cost with a restricted threshold and the lack of flexibility.

Relying on our core results in this paper and particularly GSIPA, we subsequently propose the concept of dynamic threshold (weighted) VRFs (DT-VRFs). They eliminate the need for an expensive DKG as they support a silent setup process. Here the joint public key of the parties is computed as a deterministic function of their locally computed public keys. They support the dynamic selection of a threshold and parties can join the system in an ad-hoc manner, simply publishing their public keys asynchronously. Moreover, we can smoothly adapt them to weighted VRFs as in the case of threshold signatures in Sec. 5.2 and similar to weighted VRFs in [25], where the threshold is determined not just by the number of participants but also by their individual weights, *e.g.*, stake or share in proof-of-stake systems. We, however, note that due to the dynamic threshold setting, the uniqueness property of the DT-VRF is conditioned on the respective subset that satisfies the threshold. As already discussed in the introduction, for many application this does not represent a drawback and is sufficient.

A key-homomorphic and aggregate VRF [56]. Our starting point is the aggregate VRF in [56], where a key pair is defined as: $k \leftarrow\!\$\ \mathbb{Z}_p$ and $\mathsf{vk} = S^k$ with S being a randomly chosen element, *i.e.*, $S = P^s$ for some random $s \in \mathbb{Z}_p$, and the evaluation at point x is computed using a pairing as $y = e(P, H(x))^k$, with H being a hash function (modeled as a RO) that maps bitstrings to $\mathbb{G}_2$ elements. To prove that the evaluation was done correctly, one can simply compute: $\pi = H(x)^k$, which can be efficiently verified by checking

$$y = e(P, \pi) \quad \text{and} \quad e(\mathsf{vk}, H(x)) = e(S, \pi). \tag{2}$$

One interesting property of this scheme is that it is key-homomorphic for linear functions. To see this it suffices to observe that

$$\mathsf{Eval}(k_0, x) \cdot \mathsf{Eval}(k_1, x) = e(P, H(x))^{k_0} \cdot e(P, H(x))^{k_1}$$

$$= e(P, H(x))^{k_0+k_1} = \mathsf{Eval}(k_0 + k_1, x),$$

and similarly for proofs we have $\mathsf{Prove}(k_0, x) \cdot \mathsf{Prove}(k_1, x) =$

$$H(x)^{k_0} \cdot H(x)^{k_1} = H(x)^{k_0+k_1} = \mathsf{Prove}(k_0 + k_1, x).$$

Now using these properties it is clear that we can aggregate public keys as $\mathsf{avk} = \prod_{i\in[0,1]} \mathsf{vk}_i$ and verify aggregated proofs and evaluations as shown above. Actually, to construct a *secure* aggregate VRF [56], for an aggregation over verification keys $\mathsf{vk}_1, \ldots, \mathsf{vk}_\ell$ one requies techniques inspired by signature-aggregation [10]. In particular one commits to all keys and evaluations by computing $R = \tilde{G}(x, \mathsf{vk}_1, \ldots, \mathsf{vk}_\ell, e(P, \pi_1), \ldots, e(P, \pi_\ell))$ and from that unpredicably derives a coefficient vector $(r_1, \ldots, r_q) = \tilde{H}(R)$ used in the aggregated key as $\mathsf{avk} = \prod_{i\in[\ell]} \mathsf{vk}_i^{r_i}$, for the aggregated proof

$\pi = \prod_{i\in[\ell]} \pi_i^{r_i}$ and evaluations $y = \prod_{i\in[\ell]} y_i^{r_i}$. Here, $\tilde{G}$ and $\tilde{H}$ are suitable hash functions treated as random oracles.

Remark. Looking ahead, in our setting we have q public keys overall from which a subset I of size $\ell \leq q$ (threshold) of evaluations are aggregated. Consequently, we always derive a vector $(r_1, \ldots, r_q) = \tilde{H}(R)$ from R but discard all the indices that we do not need for aggregation. It is, however, easy to verify that this change does not change anything in the security arguments in [56].

High-level idea of dynamic threshold VRFs: We recall that our GSIPA protocol allows for succinct proofs and compact verification keys with respect to a threshold set I as follows: Each party i samples its key $k_i \leftarrow\!\$\, \mathbb{Z}_p^*$ independently at random and computes the public key as $(\mathsf{vk}_i = S^{k_i}, \mathbf{M}_{1i}, \mathbf{T}_i) \in \mathcal{M}^{q,\mathsf{PP}}$, where we set $m_i = k_i$ and the CRS for the message space is computed with respect to basis S insead of P. Note that $\mathbf{T}$ and $\mathbf{M}_{1i}$ are only used in GSIPA and do not need to be known to verifiers. For q parties we have $\mathbf{k} = (k_1, k_2, \ldots, k_q) \in \mathbb{Z}_p^q$ be the vector of secret keys. Also, let $\mathbf{pk}_0 = (S^{k_1}, S^{k_2}, \ldots, S^{k_q}) \in \mathbb{G}_1^q$ and we then pick the vector of weights $\mathbf{w} = (w_1, w_2, \ldots, w_q) \in \mathbb{Z}_p^q$. Note that when we consider the plain threshold case we will have $\mathbf{w} = \mathbf{1}$, the all-ones vector.

To compute a partial VRF evaluation on a message x, each participating party i uses its private key to compute: $\pi_i = H(x)^{k_i} \in \mathbb{G}_1$ and sends it to the aggregator $\mathcal{P}$. Let $I \subseteq [q]$ be the subset of parties for which the aggregator receives valid partial evaluations, *i.e.,* representing the dynamic threshold, and let us define the bit vector: $\mathbf{b} = (b_1, b_2, \ldots, b_q) \in \{0,1\}^q$ where $b_i = 1$ for each $i \in I$ and 0 otherwise. The aggregate VRF on x is then the tuple: $(\mathbf{b}, \pi)$, where $\pi = \prod_{i\in I}(\pi_i^{r_i})^{b_i}$ and the threshold (weight) of the VRF is $t = \sum_{i\in I} b_i \cdot w_i$. Then, P computes the aggregated public key: $\mathsf{avk} = \prod_{i\in I}(\mathsf{vk}_i^{r_i})^{b_i}$ with the respective r_i taken from a public vector $\mathbf{r}$ (which is defined below). Note that after deriving the randomizer $\mathbf{r}$, we must apply it only to the parties that contributed partial evaluations. This is achieved via the Hadamard product $\mathbf{u} = \mathbf{b} \circ \mathbf{r}$, which retains r_i for active parties (i.e., $b_i = 1$) and zeros out all others, effectively computing $u_i = r_i \cdot b_i$. We also need to prove this relation to ensure that the aggregation key is correctly randomized and corresponds precisely to the subset of active signers. Thus, P has to prove the following constraints:

$$(i) \quad t = \langle \mathbf{w}, \mathbf{b} \rangle \quad \text{and} \quad (ii) \quad \mathsf{avk} = \langle \mathsf{vk}, \mathbf{u} \rangle \quad \text{and} \quad (iii) \quad \mathbf{u} = \mathbf{b} \circ \mathbf{r}$$

For (i) as both vectors are scalars, we can use recent works that use IPA to prove that a committed vector is binary [[18], Sec 5.1] or alternatively use our scalar-based IPA variant of our GSIPA, where the commitment $C_{\mathbf{A}}$ can be directly computed using messages as scalars. This case closely resembles a subvector opening proof from Sec. 4.1 (with an additional $\hat{\pi}$ for the degree check). For (ii) we can use our GSIPA protocol. It remains to argue how we can prove the Hadarmard product in (iii). We want to prove that $\mathbf{u} = \mathbf{b} \circ \mathbf{r}$ where $\mathbf{u}, \mathbf{b} \in \mathbb{Z}_p^q$ are vectors committed by the prover, and $\mathbf{r} \in \mathbb{Z}_p^q$ is a public vector which is derived as $\mathbf{r} = \tilde{H}(R)$ by the verifier from the seed R sent by the prover.

This can be proven using a randomized linear test due to Groth [37] (cf. Sec. 2.4) that reduces the Hadamard product relation to an inner product argument (IPA).

Hadamard Product via IPA Reduction. To prove that $\mathbf{u} = \mathbf{b} \circ \mathbf{r}$, we reduce this to inner product checks using a random linear combination (Sec 2.4). The prover derives $\beta \in \mathbb{F}$ from the RO and defines the monomial vector $\boldsymbol{\beta} = (1, \beta, \ldots, \beta^{n-1})$, and both compute $\boldsymbol{\gamma} := \mathbf{r} \circ \boldsymbol{\beta}$. The prover then proves that $\langle \mathbf{u}, \boldsymbol{\beta} \rangle = \langle \mathbf{b}, \boldsymbol{\gamma} \rangle = z$, by providing inner product arguments for both sides.

We formalize these ideas in the relation $\mathcal{R}_{\mathsf{VRF}}$ below. The relation is for statements of the form $\{c_b, c_w, \mathsf{avk}, \tilde{y}, \tilde{\pi}, x, R\}$ with $\mathbf{r} = \tilde{H}(R)$ and it looks as follows:

$$\mathcal{R}_{\mathsf{VRF}} = \left\{ \begin{array}{l|l} e(\mathsf{avk}, H(x)) = e(S, \tilde{\pi}) \wedge & \mathbf{vk} \in \mathbb{G}^q;\ c_{\mathsf{vk}} = \mathsf{Commit}(\mathsf{vk}) \\ \tilde{y} = e(g_1, \tilde{\pi}) \neq 1_{\mathbb{G}_T}. & \mathbf{w} \in \mathbb{Z}_p^q,\ c_{\mathbf{w}} = \mathsf{Commit}(\mathbf{w}) \\ & \mathbf{b} \in \{0,1\}^q;\ c_b = \mathsf{Commit}(\mathbf{b}) \\ & \langle \mathbf{w}, \mathbf{b} \rangle = t;\ \langle \mathbf{vk}, \mathbf{u} \rangle = \mathsf{avk} \\ & \mathbf{u} = \mathbf{b} \circ \mathbf{r} \end{array} \right\}.$$

Dynamic Threshold VRFs. Subsequently, we define dynamic threshold VRFs which are closely aligned with aggregate VRFs (defined in Appendix B.1), apart from the fact that we do not require to make the key-hommomorphic property from [56] explicit.

Definition 15 (Dynamic Threshold VRF (DT-VRF)). *A DT-VRF is a tuple of PPT algorithms with the following syntax:*

$\underline{\mathsf{Setup}(1^\lambda, q)}$: The setup algorithm on input the security parameter $\lambda \in \mathbb{N}$ and optional an upper bound of users q in the system, returns the common reference string crs.

$\underline{\mathsf{KeyGen}(\mathsf{crs})}$: The generation algorithm on input the crs samples a secret key k_i and returns a public key $\mathsf{pk} = (\mathsf{vk}_i, \tau_i)$ for each party, where $\mathsf{vk}_i = \mathsf{VKey}(k_i)$ for a bijective function VKey and τ_i are additional elements depending on crs (and k_i).

$\underline{\mathsf{Eval}(k_i, x)}$: The evaluation algorithm on input the secret key k_i and some point x returns an image y_i.

$\underline{\mathsf{Prove}(k_i, x)}$: The prover algorithm on input the secret key k_i and some point x returns a proof π_i.

$\underline{\mathsf{Verify}(\mathsf{vk}_i, x, y_i, \pi_i)}$: The verification algorithm on input the verification key vk_i, some point x, an image y_i, and a proof π_i returns a bit $b \in \{0,1\}$ denoting acceptance or rejection.

$\underline{\mathsf{Agg}(x, \{\mathsf{pk}_i, y_i\}_{i=1}^{\ell})}$: The aggregation algorithm on input a point x, ℓ verification keys $(\mathsf{vk}_1, \ldots, \mathsf{vk}_\ell)$ and images $(y_1, \ldots, y_\ell)$ outputs an aggregate key $a\mathsf{vk}$ and an aggregate image $\tilde{y}$.

$\underline{\mathsf{AggProve}(x, \{\mathsf{pk}_i, \pi_i\}_{i=1}^{\ell})}$: The aggregate proof algorithm on input a point x, ℓ public keys with $\mathsf{pk}_i = (\mathsf{vk}_i, \tau_i)$, and proofs $(\pi_1, \ldots, \pi_\ell)$ outputs an aggregate proof $\tilde{\pi}$.

$\underline{\mathsf{AggVerify}(a\mathsf{vk}, t, x, \tilde{y}, \tilde{\pi})}$: The aggregate verification algorithm on input an aggregate verification key $a\mathsf{vk}$, a threshold t, a point x, and an aggregate image-proof pair $(\tilde{y}, \tilde{\pi})$ outputs a bit $b \in \{0,1\}$ denoting acceptance or rejection.

The aggregated verification key avk, the aggregated image and proof should be constant size. We emphasize that we define our security in the malicious setting. That is, we consider the aggregator to be malicious.

Security Properties. As with any VRF we consider pseudorandomness and uniqueness and as with aggregate VRFs we additionally consider aggregate binding.

Aggregate pseudorandomness. It requires that the aggregate output is pseudorandom as long as there is at least one honest participant that provides a partial evaluation. We note that although we are in a threshold setting, we consider *the strong variant* as in aggregate VRFs so that except from one participant all other contributions can come from malicious users.

Definition 16 (Aggregate Pseudorandomness). *A* DT-VRF *satisfies* pseudorandomness *if there exists a negligible function ϵ such that for $\lambda \in \mathbb{N}$, all polynomials $q = q(\lambda)$, and all admissible PPT adversaries $\mathcal{A}$, it holds that*

$$\left| \frac{1}{2} - \Pr\left[1 \leftarrow \mathsf{ExpPseudoRandom}_{\mathcal{A}}(1^\lambda, q)\right] \right| \leq \epsilon(\lambda)$$

where the experiment $\mathsf{ExpPseudoRandom}_{\mathcal{A}}$ *is defined as follows:*

1. *The challenger runs setup* $\mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda, q)$*, generates a key pair* $(k, \mathsf{pk}) \leftarrow \mathsf{KeyGen}(\mathsf{crs})$*, and sends* $(\mathsf{crs}, \mathsf{pk})$ *to* $\mathcal{A}$*.*
2. *$\mathcal{A}$ can query adaptively and at any time an oracle on input x_i and receives back from the challenger a pair (y_i, π_i) where $y_i \leftarrow \mathsf{Eval}(k, x_i)$ and $\pi_i \leftarrow \mathsf{Prove}(k, x_i)$.*
3. *At any point (including between evaluation queries), $\mathcal{A}$ can query a challenge input x^* along with some challenge keys $\{\mathsf{pk}_i\}_{i=1}^{\ell}$ with $\ell \in [q]$. If for any i it holds that $\mathsf{VerifyMsg}(\mathsf{crs}, \mathsf{pk}_i) \neq 1$, the challenger returns $\perp$. Otherwise the challenger (inefficiently) computes $k_i \leftarrow \mathsf{VKey}^{-1}(\mathsf{vk}_i)$ and sets $y_i \leftarrow \mathsf{Eval}(k_i, x^*)$. Then it flips a coin $b \leftarrow \{0,1\}$ and computes:*

$$y^* \leftarrow \begin{cases} \mathsf{Agg}(x^*, \{\mathsf{pk}, \mathsf{Eval}(k, x^*), \{\mathsf{pk}_i, y_i\}_{i=1}^{\ell}\}) & \text{if } b = 0 \\ Y \leftarrow\!\!\$\ D & \text{if } b = 1 \end{cases}$$

4. *At the end of the experiment, $\mathcal{A}$ outputs a guess b^*. The experiment returns 1 if and only if $b^* = b$.*

Furthermore, we say that $\mathcal{A}$ is admissible *if $x^* \notin Q$, where Q denotes the set of points queried to the oracle as defined above.*

Subset Uniqueness. We also require subset uniqueness meaning for any given aggregate image and a subset I, where $I = \{\mathsf{pk}_i\}_{i \in [\ell]}$ represents the subset of partial evaluators' public keys, the output is uniquely determined. There must exist a unique aggregate proof such that the aggregation remains a valid VRF evaluation. This prevents adversaries from producing valid proofs for incorrect sets of keys *i.e.*, the subset I.

Definition 17 (Subset Uniqueness). *A* DT-VRF *satisfies* subset uniqueness *if for $\lambda \in \mathbb{N}$, all polynomials $q = \mathsf{poly}(\lambda)$, all* crs, q *in the support of* $\mathit{Setup}(1^\lambda, q)$*, all subsets $I \subseteq \{\mathsf{pk}\}_{i\in[q]}$ of size $t = \ell = |I|$, and all aggregate keys* avk*, all points $x \in \{0,1\}^\lambda$, all aggregate images $(\hat{y}_0, \hat{y}_1)$ and proofs $(\tilde{\pi}_0, \tilde{\pi}_1)$, it holds that*

$$1 = \mathsf{AggVerify}(a\mathsf{vk}, t, x, \tilde{y}_0, \tilde{\pi}_1) = \mathsf{AggVerify}(a\mathsf{vk}, t, x, \tilde{y}_1, \tilde{\pi}_1)$$

$$\implies \tilde{y}_0 = \tilde{y}_1.$$

Aggregate binding requires that no attacker is able to find two different pre-images for a given aggregate. It is identical to the original one in Def. 23, and does not change with a dynamic threshold as all keys can be sampled maliciously, with the only difference that all pk_i are checked for membership in the message space.

Definition 18 (Aggregate Binding). *A* DT-VRF *satisfies* aggregate binding *if there exists a negligible function ϵ such that for $\lambda \in \mathbb{N}$, all polynomials $q = \mathsf{poly}(\lambda)$, and all PPT adversaries $\mathcal{A}$ it holds that*

$$\Pr\left[1 \leftarrow \mathsf{ExpAggBind}_{\mathcal{A}}(1^\lambda, q)\right] \leq \epsilon(\lambda)$$

where the experiment $\mathsf{ExpAggBind}_{\mathcal{A}}$ *is defined as follows.*

1. *The challenger runs setup* $\mathsf{crs} \leftarrow\!\!\$\ \mathsf{Setup}(1^\lambda, q)$ *and sends the output to $\mathcal{A}$.*
2. *$\mathcal{A}$ returns a point x^*, $t = \ell \in [q]$ verification keys $\mathsf{pk}_1, \ldots, \mathsf{pk}_\ell$, two tuples $(y_1, \ldots, y_\ell)$ and $(y_1^*, \ldots, y_\ell^*)$, and proofs $(\pi_1, \ldots, \pi_\ell)$.*
3. *The challenger computes $(\mathsf{avk}, \tilde{\pi}) \leftarrow \mathsf{AggProve}(x^*, \{\mathsf{pk}_i, \pi_i\}_{i=1}^\ell)$ and $(\mathsf{avk}, \tilde{y}) \leftarrow \mathsf{Agg}(x^*, \{\mathsf{pk}_i, y_i^*\}_{i=1}^\ell)$ and outputs 1 if and only if:*

$$\{\mathsf{Verify}(\mathsf{vk}_i, x^*, y_i, \pi_i) = 1\}_{i=1}^\ell \text{ and } \mathsf{AggVerify}(\mathsf{avk}, t, x^*, \tilde{y}, \tilde{\pi}) = 1$$

$$\text{and } \forall i \in [\ell] : \mathsf{VerifyMsg}(\mathsf{crs}, \mathsf{pk}_i) = 1$$

and furthermore $(y_1, \ldots, y_\ell) \neq (y_1^*, \ldots, y_\ell^*)$.

Our Construction. We now present our construction, which extends the aggregated VRF in [56] as already outlined above.

Scheme 4 (Our DT-VRF**)** *Let* GSFC *be Scheme 1 and* GSIPA *be Scheme 3, our* DT-VRF *works as follows:*

$\underline{\mathsf{Setup}(1^\lambda, q)}$: *Run* $\mathsf{GSFC.Setup}'(1^\lambda, q, S)$*, a modified setup algorithm that generates all the parameters with respect to the basis S rather than P. Additionally define hash functions $H : \{0,1\}^\lambda \to \mathbb{G}_2$, $\tilde{G}_\ell : \{0,1\}^\lambda \times \mathbb{G}_1^\ell \times \mathbb{G}_T^\ell \to \{0,1\}^\lambda$ and $\tilde{H} : \{0,1\}^\lambda \to \mathbb{Z}_p^q$.*

$\underline{\mathsf{KeyGen}(\mathsf{crs})}$: *Sample $k_i \leftarrow_r \mathbb{Z}_p$ and return a public key $\mathsf{pk}_i = (\mathsf{vk}_i, \tau_i) = (S^{k_i}, (\mathbf{M}_{1i}, \mathbf{T}_i)) \in \mathcal{M}^{q,\mathsf{pp}}$.*

$\underline{\mathsf{Eval}(k_i, x)}$: *Return evaluation $y_i = e(P, H(x)^{k_i})$.*

$\underline{\mathsf{Prove}(k_i, x)}$: *Returns proof $\pi_i = H(x)^{k_i}$.*

$\underline{\mathsf{Verify}(\mathsf{vk}_i, x, y_i, \pi_i)}$: *Return 1 if $\mathsf{vk}_i \in \mathbb{G}_1$ and*

$$y = e(P, \pi) \quad \text{and} \quad e(\mathsf{vk}_i, H(x)) = e(S, \pi)$$

and 0 otherwise.

$\underline{\mathsf{Agg}(x, \{\mathsf{pk}_{j_i}, y_{j_i}\}_{i=1}^\ell)}$: *Compute $R \leftarrow \tilde{G}_\ell(x, \mathsf{vk}_{j_1}, \ldots, \mathsf{vk}_{j_\ell}, y_{j_1}, \ldots, y_{j_\ell})$ and $(r_1, \ldots, r_q) \leftarrow \tilde{H}(R)$ and return $\mathsf{avk} = \prod_{i\in[\ell]} \mathsf{vk}_{j_i}^{r_{j_i}}$ and $\tilde{y} = \prod_{i\in[\ell]} y_{j_i}^{r_{j_i}}$.*

$\underline{\mathsf{AggProve}(x, \{\mathsf{pk}_{j_i}, \pi_{j_i}\}_{i=1}^\ell)}$: *Computing the proof does:*

- *Compute $R \leftarrow \tilde{G}_\ell(x, \mathsf{vk}_{j_i}, \ldots, \mathsf{vk}_{j_\ell}, e(P, \pi_{j_1}), \ldots, e(P, \pi_{j_\ell}))$ along with $(r_1, \ldots, r_q) \leftarrow \tilde{H}(R)$*
- *Compute $\mathsf{avk} = \prod_{i\in[\ell]} \mathsf{vk}_{j_i}^{r_{j_i}}$*
- *Compute $\pi' = \prod_{i\in[\ell]} \pi_{j_i}^{r_{j_i}}$*
- *Compute a non-interactive proof Π for relation $\mathcal{R}_{\mathsf{VRF}}$ with threshold $t = \ell$ using* GSIPA*.*
- *Return proof $\tilde{\pi} = (\mathsf{avk}, \Pi, \pi', R)$*

$\underline{\mathsf{AggVerify}(\mathsf{avk}, t, x, \tilde{y}, \tilde{\pi})}$: *Parse* $\tilde{\pi} = (\mathsf{avk}, \Pi, \pi', R)$ *and output* 1 *if the following holds and* 0 *otherwise:*
- $e(\mathsf{avk}, H(x)) = e(S, \pi') \wedge \tilde{y} = e(g_1, \pi') \neq 1_{\mathbb{G}_T}$.
- *Verify* Π *with threshold* t *and* R.

We remark that the dynamic threshold weighted VRF is almost identical to our above construction. The only difference is that within $\mathcal{R}_{\mathsf{IP}}$ (cf. Scheme 3), instead of using $\mathbf{w} = (1, 1, \ldots, 1)$ we use the predefined weights $\mathbf{w} = (w_1, w_2, ..., w_n)$ and for malicious parties who send keys that do not verify, we set $w_i = 0$. The other changes follow straightforwardly.

Remark 1 (On the Cost of Computing $r = \tilde{H}(R)$). Computing r involves hashing a linear number of elements, i.e., $O(n)$ time. This cost is inherent to all constructions that rely on deriving a challenge as a linear combination over a set of inputs. Our approach is consistent with prior work in this case. However, hashing is computationally inexpensive, and the overall proof size remains constant.

Theorem 3. *Let the aggregate VRF in [56] satisfy aggregate pseudorandomness, binding, and uniqueness (see Appendix B.2), and assume* GSIPA *is knowledge sound. Then, the construction in Scheme 4 satisfies aggregate pseudorandomness, aggregate binding and subset uniqueness in the ROM.*

Proof (Sketch). The security essentially directly follows from the underlying aggregate VRF construction in [56] and the knowledge-soundness of GSIPA. One conceptual modification that we have made is that we always derive a vector $(r_1, \ldots, r_q) = \tilde{H}(R)$ from R of length q, but discard all the indices that are not used in an aggregation. It is, however, easy to verify that this change does not change anything in the security arguments in [56] as they just require unpredictability of the values r_i based on an input that cannot be fully controlled by the adversary. So we assume they are used consistently in the DT-VRF and the underlying aggregate VRF. See Appendix C.3 for proofs.

On integration into randomness beacons. Similar to other approaches that employ threshold VRFs in consensus protocols and randomness beacons [22,43,25], our DT-VRF can likewise serve as a cryptographic core for decentralized randomness beacons. In each round, a public input x (e.g., epoch number or previous output) is fixed, and parties compute partial evaluations on x using their secret keys. Once enough evaluations are collected, an aggregator derives the aggregate VRF output and a succinct proof, which together define the beacon value. The DT-VRF ensures unpredictability until aggregation, uniqueness per input, and public verifiability. Its support for dynamic and weighted participation eliminates the need for setup or key distribution across rounds. Combined with simple coordination mechanisms for input selection and output finalization (as commonly done in blockchain-based randomness beacons), our DT-VRF enables efficient and decentralized beacon protocols.

5.2 Silent Threshold Signatures Without RO

Recall that we follow the blueprint from [24] and [33], which considers a large fixed group of signers who produce partial signatures. An aggregator takes the partial signatures and computes a SNARK proof to prove correctness of the aggregated public key and the threshold t. The SNARK is used to compute a short constant-size key, not to hide any information (apk is simply the product of all signers' public keys). This results in a compact signature, which can be verified independently of the number of signers.

While the above constructions are quite efficient, recent work by Lehmann and Özbay [50] found the original construction from [24] to be insecure. This motivates the search for alternative approaches. Towards that end, we propose to instantiate threshold signatures with a silent setup using our GSIPA from Section 4. This approach eliminates the need for customized SNARKs, simplifying the overall construction. Consequently, we achieve a more modular design, making their implementation easier and less error prone.

In the following, we introduce the syntax and unforgeability notion for silent threshold signatures. For ease of exposition, we adapt the definitions from [33] and [12].

Definition 19 (Silent Threshold Signature (STS)). *A Silent Threshold Signature scheme consists of the following polynomial-time algorithms:*

$\underline{\mathsf{Setup}(1^\lambda, q)}$: *On input of the security parameter* $\lambda \in \mathbb{N}$ *and the parameter* q*, the setup algorithm outputs public parameters* pp*, which are treated as an implicit input to all algorithms.*

$\underline{\mathsf{KGen}(i)}$: *On input the index i, the key generation algorithm outputs a public/secret key pair $(\mathsf{pk}_i, \mathsf{sk}_i)$.*

$\underline{\mathsf{HintGen}(\mathsf{sk}, q)}$: *On input pp, a secret key sk, and the number of parties q, the hint generation algorithm outputs a hint hint.*

$\underline{\mathsf{Preprocess}(\{(\mathsf{hint}_i, \mathsf{pk}_i)\}_{i\in[q]})}$: *On input pp and all pairs $\{(\mathsf{hint}_i, \mathsf{pk}_i)\}_{i\in[q]}$, the preprocessing algorithm computes an aggregation key AK and a (succinct) verification key vk.*

$\underline{\mathsf{Sign}(\mathsf{sk}, m)}$: *On input a secret key sk and a message m, the signing algorithm outputs a partial signature σ.*

$\underline{\mathsf{PartialVerify}(m, \sigma, \mathsf{pk})}$: *On input a message m, a partial signature σ, and a public key pk, this algorithm returns 1 if and only if the partial signature verifies correctly, and 0 otherwise.*

$\underline{\mathsf{SigAggr}(\mathsf{AK}, \mathsf{vk}, J, \{\sigma_j, \mathsf{pk}_j\}_{j\in J})}$: *On input pp, an aggregation key AK, and a set of partial signatures and public keys $\{\sigma_j, \mathsf{pk}_j\}_{j\in J}$, the signature aggregation algorithm outputs a (succinct) aggregated signature σ.*

$\underline{\mathsf{Verify}(\mathsf{vk}, m, \sigma, t)}$: *On input the verification key vk, a message m, a signature σ, and threshold t, it verifies the signature and outputs a bit $b \in \{0,1\}$ denoting acceptance or rejection.*

Definition 20 (Unforgeability of STS). *A STS scheme satisfies unforgeability if, for any adversary $\mathcal{A}$, the output of the game shown below is 1 with probability $\leq \mathsf{negl}(\lambda)$.*

- *The adversary picks q to set the maximum bound on the number of parties.*
- *The challenger runs $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda, q)$ and gives pp to $\mathcal{A}$.*
- *The adversary picks a subset of parties to corrupt $A \leftarrow \mathcal{A}(\mathsf{pp})$.*
- *For all honest parties $i \in [q] \setminus A$, the public key and hint are sampled honestly by the challenger $(\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow \mathsf{KGen}(i)$ and $\mathsf{hint}_i = \mathsf{HintGen}(\mathsf{sk}_i, q)$ and are given along with crs to $\mathcal{A}$.*
- *For all $i \in A$, the adversary picks a public key pk_i and the corresponding hint hint_i.*
- *The challenger preprocesses $\{\mathsf{hint}_i, \mathsf{pk}_i\}_{i\in[q]}$ as $(\mathsf{AK}, \mathsf{vk}) \leftarrow \mathsf{Preprocess}(\{\mathsf{hint}_i, \mathsf{pk}_i\}_{i\in[q]})$ which are given to $\mathcal{A}$.*
- *The adversary may make a partial signature query with a message m and an honest party $i \in [q] \setminus A$, the query is returned by computing $\sigma_i = \mathsf{Sign}(\mathsf{sk}, m)$. This is run as many times as desired by $\mathcal{A}$.*
- *Finally, the adversary shall output a challenge message m^* and an aggregated signature σ^*. Let S^* be the subset of honest parties queried by the adversary to sign m^*.*
- *The adversary wins the forgery game if there exists a threshold $t > |S^* \cup A|$ such that $\mathsf{Verify}(m^*, \sigma^*, T, \mathsf{vk}) = 1$, in which case, the output of the game is 1.*

Accountable Multi-Signature without RO. We consider the recent accountable multi-signature without RO from Boneh, Partap and Waters [12], which we adapt to the silent threshold setting following the ideas from [33]. More in detail, Boneh *et al.* consider a setting where n parties, identified by an index $i \in \{1, \ldots, n\}$, compute their individual keys locally and non-interactively. As explained in [12], parties can choose a unique slot number as their index using a counter on a public bulletin board that is initialized to zero. Whenever a party joins the system, it increments the counter by one and uses the current value as its slot number. Once all parties have joined, the counter value n is used as the upper bound on the number of signers, and all participants use it to compute auxiliary information that they include as part of their (extended) public key. The aggregator takes all the extended public keys (or hints) and computes the aggregation key and global verification key. The latter is used to verify signatures from any subset of signers. As an accountable scheme, signatures from [12] include a minimal description of the signer set (*i.e.,* a bit vector of length n), required for verification and accountability.

High-level idea of silent threshold signatures without RO. We observe that for silent threshold signatures accountability is not required and we can thus remove the linear dependency on n for the signature size. Specifically, using our GSIPA , we can commit to the bit vector and auxiliary group elements used to verify the signature to get a silent threshold scheme. Thus, we extend the setup of [12] to include our GSIPA's setup, requiring the use of extended public keys defined over the GSIPA 's message space. Key aggregation can be done as parties join the system, verifying the correctness of each (extended) public key using the crs. Subsequently, the aggregator only needs to fetch the public keys of the threshold signing parties to compute the GSIPA. This allows us to achieve a constant-size silent threshold scheme that, contrary to prior work, does not rely on the random oracle model to prove security. We compare our new scheme with previous work in Table 2.

Our Construction. In the following, we present our silent threshold signatures scheme. It is based on the multi-signature from [12], which achieves existential unforgeability without random oracles

by relying on the Waters algebraic hash [64]. As such, it considers s-bit messages $m = (m_1, \ldots, m_s) \in \{0,1\}^s$ where s only depends on the security parameter. For ease of exposition, we highlight the relevant modifications—compared to the original multi-signature scheme of [12]—in green. We switch the verification key from $\mathbb{G}_2$ to $\mathbb{G}_1$ for compatibility with our GSIPA.

$\underline{\mathsf{Setup}(1^\lambda, q) \to \mathsf{pp}}$:

- $\mathsf{GlobalSetup}(1^\lambda, q)$: Sample a bilinear group along with q fixed generators $\mathsf{par} \leftarrow (\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, P, \hat{P}, \{\hat{U}_i\}_{i\in[q]} \subseteq \mathbb{G}_2, p)$.
- $\mathsf{SystemSetup}(\mathsf{par}) \to \mathsf{ppar}$: Sample $\hat{V}_0, \ldots, \hat{V}_s \leftarrow\!\!\$\ \mathbb{G}_2$ and output $\mathsf{ppar} \leftarrow (\mathsf{par}, \hat{V}_0, \ldots, \hat{V}_s)$.
- $\mathsf{GSIPA.Setup}(\mathsf{ppar}) \to \mathsf{pp}$: Sample $\alpha \leftarrow\!\!\$\ \mathbb{Z}_p$, compute $\mathsf{crs} \leftarrow \{B_1 = P^{\alpha^1}, \ldots, B_q = P^{\alpha^q}, B_{q+2} = P^{\alpha^{q+2}}, \ldots, B_{2q} = P^{\alpha^{2q}}, \hat{B}_1 = \hat{P}^{\alpha^1}, \ldots, \hat{B}_q = \hat{P}^{\alpha^q}, g_t^{\alpha^{q+1}}\}$ with $g_t = e(P, \hat{P})$, and output $\mathsf{pp} \leftarrow (\mathsf{ppar}, \mathsf{crs})$.

$\underline{\mathsf{KGen}(i) \to (\mathsf{sk}_i, \mathsf{pk}_i)}$:

- Sample $\beta_i \leftarrow \mathbb{Z}_p$. Set $\mathsf{sk}_i \leftarrow \beta_i$.
- Set $M_{i,0} \leftarrow P^{\beta_i}, M_{i,1} = B_i^{\beta_i} = P^{\alpha^i \cdot \beta_i}, T_{i,j} = B_j^{\beta_i} = P^{\alpha^j \cdot \beta_i}$, for all $j \in [i+1, i+q] \setminus \{q+1\}$.
  Set $\mathsf{pk}_i \leftarrow (\mathsf{vk}_i = M_{i,0}, M_{i,1}, \{T_{i,j}\}_{j\in[i+1,i+q]\setminus\{q+1\}})$. Output $(\mathsf{sk}_i, \mathsf{pk}_i)$.

$\underline{\mathsf{HintGen}(\mathsf{sk}_i, q) \to \mathsf{hint}_i}$:

- For all $j \in [q] \setminus \{i\}$, compute $\hat{pk}_{i,j} \leftarrow \hat{U}_j^{\beta_i}$.
  Set $\mathsf{hint}_i \leftarrow \{\hat{pk}_{i,j}\}_{j\in[q]\setminus\{i\}}$. Output hint_i.

$\underline{\mathsf{Preprocess}(\{\mathsf{pk}_1, \mathsf{hint}_1\}_1, \ldots, \{\mathsf{pk}_q, \mathsf{hint}_q\}_q) \to (\mathsf{AK})}$: [6]

- For each $i \in [q]$, parse $\{\mathsf{pk}_i, \mathsf{hint}_i\}_i$ as $(\mathsf{vk}_i = M_{i,0}, M_{i,1}$ $\{T_{i,j}\}_{j\in[i+1,i+q]\setminus\{q+1\}}, \{\hat{pk}_{i,j}\}_{j\in[q]\setminus\{i\}})$.
- If $e(M_{i,0}, \hat{U}_j) \neq e(P, \hat{pk}_{i,j})$ or $e(M_{i1}, \hat{B}_{j-i}) = e(T_{i,j}, \hat{P}) \wedge e(M_{i,1}, \hat{P}) = e(M_{i,0}, \hat{B}_i)$ for some $i, j \in [q]$ output $\perp$.
- Output $(\mathsf{AK}_i \leftarrow \{\hat{pk}_{i,j}\}_{j\in[q]\setminus\{i\}})$.

$\underline{\mathsf{Sign}(\mathsf{sk}_i, m) \to \sigma_i}$:

- Parse sk_i as β_i, sample $r_i \leftarrow\!\!\$\ \mathbb{Z}_p$, compute $\sigma_{i,0} \leftarrow P^{r_i}$ and $\sigma_{i,1} \leftarrow \hat{U}_i^{\beta_i} \cdot (\hat{V}_0 \prod_{i\in[s]} \hat{V}_i^{m_i})^{r_i}$. Output $\sigma_i = (\sigma_{i,0}, \sigma_{i,1})$.

$\underline{\mathsf{SigAggr}(\mathsf{AK}, \mathsf{vk}, J, \{\sigma_j, \mathsf{pk}_j\}_{j\in J}) \to \sigma}$:

- For each $j \in J$, parse σ_j as $(\sigma_{j,0}, \sigma_{j,1})$. For each $i \in [q]$ parse AK_i as $\{\hat{pk}_{i,k}\}_{k\in[q]\setminus\{i\}}$.
- Compute $\sigma_0 \leftarrow \prod_{j\in J} \sigma_{j,0}$.
- Compute $\hat{\sigma}_{j,1} \leftarrow \sigma_{j,1} \cdot \prod_{k\in J\setminus\{j\}} \hat{pk}_{k,j}$ for each $j \in J$.
- Compute $\hat{\sigma}_1 \leftarrow \prod_{j\in J} \hat{\sigma}_{j,1}$ and $\hat{\sigma}_2 \leftarrow \prod_{j\in J} \hat{U}_j$.
- Compute a non-interactive proof Π for relation $\mathcal{R}_{\mathsf{STS}}$ with threshold t for subset (threshold) $J \subseteq [q]$ with $t = \langle \mathbf{w} = \mathbf{1}, \mathbf{b} \rangle$ [7] as

$$\mathcal{R}_{\mathsf{STS}} := \left\{ \begin{array}{l|c} e(P, \sigma_1) = & \mathbf{vk} \in \mathbb{G}^q;\ c_{\mathsf{vk}} = \mathsf{Commit}(\mathsf{vk}) \\ e(\sigma_0, \hat{V}_0 \prod_{i\in[s]} \hat{V}_i^{m_i}) \cdot e(\mathsf{avk}, \hat{\sigma}_2) & \mathbf{w} \in \mathbb{Z}_p^q,\ c_{\mathbf{w}} = \mathsf{Commit}(\mathbf{w}) \\ & \mathbf{b} \in \{0,1\}^q;\ c_b = \mathsf{Commit}(\mathbf{b}) \\ & \langle \mathbf{w}, \mathbf{b} \rangle = t;\ \langle \mathbf{vk}, \mathbf{u} \rangle = \mathsf{avk} \end{array} \right\}$$

  where $\mathbf{b}$ represents the subset J, $\mathsf{avk} = \prod_{i\in[J]} \mathsf{vk}_i$, $\mathsf{pk}_j = (M_{i0}, M_{i1}, \mathbf{T})$, the witness is $(\mathbf{M_0}, \mathbf{b})$, and $\mathsf{aux} = (\mathbf{M}_{i1}, \mathbf{T})$.
- Output $\sigma = (\sigma_0, \hat{\sigma}_1, \hat{\sigma}_2, C_{\mathsf{vk}}, C_b, \mathsf{avk}, \pi_{\mathsf{avk}}, \hat{\pi})$.

$\underline{\mathsf{Verify}(\mathsf{vk}, m, \sigma, t) \to \{0,1\}}$:

- Parse σ as $(\sigma_0, \hat{\sigma}_1, \hat{\sigma}_2, C_{\mathsf{vk}}, C_b, \mathsf{avk}, \pi_{\mathsf{avk}}, \hat{\pi})$ and output 1 iff $\hat{\pi}$ verifies for t and $e(P, \sigma_1) = e(\sigma_0, \hat{V}_0 \prod_{i\in[s]} \hat{V}_i^{m_i}) \cdot e(\mathsf{avk}, \hat{\sigma}_2)$.

Remark 2. Similarly to other schemes [12,24,33], to prevent rogue-key attacks, we also implicitly assume that each public key is accompanied by a proof of possession.

[6] We define this algorithm for consistency with the definitions in [33].

[7] We set weights to 1 for simplicity but our IPA can be used to prove this inner product.

Table 2. Comparison of silent threshold signature schemes. We provide concrete numbers considering type-3 pairing groups. To ensure a fair comparison, we have adjusted the reported signature sizes by consistently including both the aggregated public key and the individual signature shares across all schemes—elements that were not always accounted for in previous works. For the aggregated public key size, we exclude individual BLS keys and signer weights, common to all schemes.

Scheme	CRS size	Signature size	Verification key size	Verification cost	Aggregation key size	Aggregation cost	Setup	Security model
zk-SNARK [38]	Circuit specific	$2_{\mathbb{G}_1}, 1_{\mathbb{G}_2}$	$7_{\mathbb{G}_1}$	$3\mathsf{P}, 1\mathsf{Exp}_{\mathbb{G}_1}$	Large	High	MPC	GGM+RO
DCX+24 [24]	$4n_{\mathbb{G}_1} + n\mathbb{G}_2$	$7_{\mathbb{G}_1} + 2_{\mathbb{G}_2} + 1_{\mathbb{F}}$	$5_{\mathbb{G}_1}$	$13\mathsf{P}, 2\mathsf{Exp}_{\mathbb{G}_1}, 3\mathsf{Exp}_{\mathbb{G}_2}$	$4n_{\mathbb{G}_1}$	$O(n)$	PKI	AGM+RO
hinTS [33]	$n_{\mathbb{G}_1} + n\mathbb{G}_2$	$5_{\mathbb{G}_1} + 2_{\mathbb{G}_2} + 8_{\mathbb{F}}$	$2_{\mathbb{G}_1} + 1_{\mathbb{G}_2}$	$9\mathsf{P}, 1\mathsf{Exp}_{\mathbb{G}_1}$	$4n_{\mathbb{G}_1}$	$O(n)$	PKI	AGM+RO
This work + [12]	$(2n-1)_{\mathbb{G}_1} + (n+2)\mathbb{G}_2 + 1\mathbb{G}_T$	$6_{\mathbb{G}_1} + 4_{\mathbb{G}_2}$	$1_{\mathbb{G}_1}$	$14\mathsf{P}, 1\mathsf{Exp}_{\mathbb{G}_2}$	$(n^2)_{\mathbb{G}_1} + (n^2+n)_{\mathbb{G}_2}$	$O(n)$	PKI	AGM

As in Sec. 5.1, to prove that C_b is binary we can either use [18] or a scalar-based IPA variant of our GSIPA, where $C_{\mathbf{A}}$ is computed using scalar messages without relying on structured ones. We go with the latter and report our numbers with this in mind when comparing our scheme with related work in Table 2.

Theorem 4 (Unforgeability of STS). *Let* GSIPA *be extended knowledge sound, and the AMS scheme of [12] be an unforgeable. Then our construction is unforgeable under the definitions of 20.*

Unforgeability is proven in Appendix C.4. We note that, unlike the work of [24], we do not face a simulation barrier in our setting. This is because the extended knowledge soundness of GSIPA allows the simulator to obtain the CRS from the extended soundness oracle, enabling extraction without knowledge of the trapdoor later when it requires it.

STS Comparison. We provide a comparison of our new threshold signature scheme in Sec. 5.2 with the most significant related works in Table2. As shown, our efficiency is comparable to that of [24], whose security has been shown to be flawed [49], and is also close to [33], while our construction does not rely on the random oracle model.

5.3 Constant-Size Oblivious Proofs

As briefly mentioned in Sec. 3.2, Garg et al. [32] present a proof system for proving statements about *hidden secrets* that satisfy arbitrary circuit relations. Notably, their proofs are generated *obliviously*—that is, without access to the underlying secrets. To achieve this, they introduce the notion of *FRI for hidden values*, enabling efficient proximity testing over *encapsulated inputs* denoted by functions $\mathsf{LHEncap}(m)$. Their approach yields polynomial commitment schemes (PCS) for values hidden under linearly homomorphic primitives such as homomorphic encryption (LHE), homomorphic commitments, fully homomorphic encryption (FHE), or group exponentiation. Leveraging this, they build succinct non-interactive arguments of knowledge (SNARKs) for proving correct evaluations on encrypted or hidden data (e.g., FHE ciphertexts), and they enable secure delegation of SNARK generation to untrusted servers without leaking the witness. More concretely, the PCS in [32] operates as follows: Given a polynomial

$$f(X) = m_0 + m_1 X + \cdots + m_{q-1} X^{q-1}$$

for coefficients $m_i \in \mathbb{F}$, let $[\![m_0]\!], \ldots, [\![m_{q-1}]\!]$ be their encapsulations under a linearly homomorphic mapping. For example, $[\![m]\!]$ might represent an LHE ciphertext, a homomorphic commitment, or a group exponentiation such as g^m. These encapsulations may offer perfect (*e.g.,*encryption, commitments) or computational hiding (*e.g.,*group exponentiation). The committer can then compute $[\![f(x^*)]\!]$ along with a succinct proof that it corresponds to the evaluation of the committed polynomial at a public input x^*. The general concept of Linearly Homomorphic Encapsulation (LHEncap), as formalized in [32], refers to any homomorphic mapping from $\mathbb{F}$ to an output space $\mathcal{S}$, preserving linear structure.

Our PCS construction in Sec. 3.2 achieves similar functionality, but *without relying on the FRI protocol*. Instead, we instantiate $\mathsf{LHEncap}(m)$ using group exponentiation, i.e., $\mathsf{LHEncap}(m) = P^m$. This yields a polynomial commitment scheme for group-exponentiated values with *constant-size proofs and verification*, where hiding is computational. To strengthen this to *perfect hiding*, we apply the

randomization technique from [45]. Specifically, we randomize the message vector $\mathbf{M} \in \mathbb{G}_1^q$ using a blinding factor $\gamma \in \mathbb{F}$ to obtain $\mathbf{M}' = \mathbf{M}^\gamma$. The associated tag vectors are randomized accordingly to ensure verification correctness. The original messages can later be recovered via de-randomization using γ.

Integration with Polynomial IOP-based SNARKs. Similar to [32], our scheme naturally supports integration with *polynomial IOP-based SNARKs*. A *polynomial IOP* is an interactive oracle proof in which the prover provides oracle access to polynomials, and the verifier queries them at chosen points.

These IOPs can be transformed into succinct arguments using any PCS (see [21,32] for background and framework). Our polynomial commitment scheme then allows the prover to commit to the encoded polynomial in the exponent and answer verifier queries with consistent, succinct evaluation proofs. We leave the full construction and integration into a complete SNARK system as an exciting direction for future work.

6 Implementation and Evaluation

We have implemented our main primitives and protocols and have instantiated our dynamic threshold VRF scheme to demonstrate the practicality of our contributions. We provide a Rust prototype based upon the `ark-bls12-381` implementation of type-3 bilinear groups with the BLS12-381 curve.

GSFC and GSIPA. We first benchmarked the GSFC and GSIPA schemes (see Table 3) a Hetzner CPX31 (4 virtual CPU cores, 8 GB RAM) instance, running Debian 13 and Rust 1.89. We show the mean execution time of each algorithm, following a clean separation between components such as setup, commitment, proof generation, verification, and the MsgGen algorithm used to create messages (cf. Section 2.6 in Section 2.6), in order to facilitate benchmarking precisely. These were tested for vector sizes of $q \in \{128, 512, 1024\}$ to observe performance scaling.

As detailed in Table 3 that reports our implementation, Setup and MsgGen phases are the most computationally expensive ones. However, both are *one-time preprocessing steps* and do not impact the efficiency of proof generation or verification during actual use. Additionally, in GSFC, verification is constant-time with respect to q. In our implementation, we *precompute* the term $\prod_{i\in[q]} \hat{B}_{q+1-i}^{b_i}$ and pass it to the verifier. This aligns with the construction, where the verifier only needs to perform a *fixed number of pairing checks*, independent of q. Since the vector $\mathbf{b}$ is public, this term can be precomputed outside the verification phase.

For GSIPA, the setup, message generation, and commitment phases (we compute $\hat{C}_b$ during the proof to align the commitment structure with GSFC—though it could also be computed during the commitment phase) exhibit similar runtimes to those of the GSFC scheme. The commitment time remains nearly constant across different values of q, as the costly exponentiations are already performed during message generation, leaving only lightweight group additions in the commitment step. Therefore, only the proof generation (Proof) and verification (Verify) times are reported here.

Table 3. Benchmark Results for GSFC and GSIPA.

q	GSFC					GSIPA	
	Setup	MsgGen	Commit	Open	Verify	Proof	Verify
128	236.05 ms	6.86 ms	304.95 μs	14.73 ms	7.57 ms	29.55 ms	4.01 ms
512	941.26 ms	25.83 ms	643.68 μs	107.02 ms	16.75 ms	264.38 ms	3.94 ms
1024	1.88 s	50.63 ms	1.10 ms	351.12 ms	25.83 ms	946.20 ms	3.90 ms

Dynamic Threshold VRF. We also benchmarked our dynamic threshold VRF from Section 5.1. We provide two sets of benchmarks to understand the complexity of all algorithms by running all algorithms on the same machine as well as the costs of running nodes distributed on machines that are distributed in different areas of the world.

To compute the IPA between two scalar vectors, we employed our scalar-based IPA variant of our

Table 4. Benchmark results for the DT-VRF with $t = \frac{q}{2} + 1$ running on a single Hetzner CPX31 instance.

q	t	Eval + Proof	Verify	AggKey + AggProof	AggVerify
128	65	3.33 ms	4.38 ms	1.68 s	20.70 ms
512	257	3.33 ms	4.41 ms	26.11 s	26.63 ms
1024	513	3.25 ms	4.43 ms	104.50 s	32.91 ms

Table 5. Benchmark results (in seconds) for the DT-VRF with $t = \frac{q}{2} + 1$ running on five distributed VMs.

q	t	distributed Eval, AggKey, AggProof
128	65	2.51 s
512	257	26.53 s
1024	513	104.33 s

GSIPA, where the commitment $C_{\mathbf{A}}$ can be directly computed using messages in scalars without relying on structured messages. For instance, the commitment in $\mathbb{G}_1$ for $\mathbf{w} \in \mathbb{Z}_p$ can be commuted $C_{\mathbf{w}} = \prod_{i\in[q]} B_i^{w_i}$. This case resembles a subvector opening from Section 4.1 (with an additional $\hat{\pi}$ for the degree check). Note that alternative IPA constructions for scalar vectors, such as those in [53,18], could also be used here.

In Table 4, we present the numbers for evaluating the VRF and providing the associated proofs (Eval + Proof), the verification (Verify), the aggregation and producing the associated proofs (AggKey + AggProof), and their verification (AggVerify). We also provide the average execution time of the aggregation steps of DT-VRF running on five virtual machines (VM) from Hetzner distributed over Northern America, Europe and Asia in Table 5. The benchmarks were conducted with four instances of CPX31 (4 virtual CPU cores, 8 GB RAM) running in Ashburn, VA, Falkenstein and Nürnberg, Germany, and Helsinki, Finland and one instance of CPX21 (3 virtual CPU cores, 4 GB RAM) running in Singapore. Each VM represented multiple nodes and one the nodes also executed the aggregation. The times presented in Table 5 represent the overall time it takes from a user requesting the evaluation of the DT-VRF until receiving the aggregated key and evaluation together with the proof. As such these numbers also include overheads of serialization, deserialization and network latency. We however want to note, that the timings are mostly dominated by the aggregation which is consistent with the benchmarks performed on a single machine.

Sizes. We briefly discuss the size of the main cryptographic objects in each scheme. In GSFC, both the commitment and the proof consist of a single group element in $\mathbb{G}_1$ (about 48 bytes). In GSIPA, the commitment consists of one element in $\mathbb{G}_1$ and one in $\mathbb{G}_2$ (about 144 bytes), while the proof includes two group elements—one in $\mathbb{G}_1$ and one in $\mathbb{G}_2$. In the dynamic threshold VRF construction, the output includes one element in $\mathbb{G}_T$ (576 bytes) for the evaluation and one in $\mathbb{G}_1$ as the proof of evaluation. Additionally, each of the four IPA proofs contributes one element in $\mathbb{G}_1$ and one in $\mathbb{G}_2$. In total, the scheme requires $5\mathbb{G}_1 + 4\mathbb{G}_2 + \mathbb{G}_T$ elements (about 1260 bytes).

6.1 Python Implementation

Along with the Rust prototype, we provide a companion Python implementation in the appendix A. This version focuses on correctness, usability, and repeatability, and readers can readily review and rerun the code. In contrast, the Rust version uses the mature rust libraries (BLS12-381) and prioritizes performance, scalability, deployment suitability, and benefits from Rust's modern security guarantees. Together, two implementations demonstrate both the practical feasibility and conceptual clarity of our approaches.

7 Conclusion

In this paper, we investigated group-scalar relations and introduced a new class of functional commitments supporting inner products for such relations that further enable polynomial commitments for values hidden in exponentiations. Based on these new building blocks, we presented an inner

product argument tailored to group-scalar relations that achieves constant-size proofs and verification time entirely within the source group. These foundational tools allowed us to construct dynamic threshold verifiable random functions (DT-VRFs) and silent threshold signatures without relying on random oracles, improving flexibility and efficiency over prior work. Our constructions avoid expensive cryptographic machinery while allowing for new cryptographic applications with strong security guarantees, high efficiency, and scalability. Prototype benchmarks show the practical viability of our techniques.

Acknowledgments

Omid Mir and Sebastian Ramacher were supported by the European Union's Horizon Europe project SUNRISE (grant agreement no. 101073821), PREPARED, a project funded by the Austrian security research programme KIRAS of the Federal Ministry of Finance (BMF), and the CHIST-ERA project REMINDER through the Austrian Science Fund (FWF) project number I 6650-N. Views and opinions expressed are however those of the authors only and do not necessarily reflect those of the funding agencies. Neither European Union nor the granting authorities can be held responsible for them. We want to thank anonymous reviewers for their helpful comments.

References

1. Abe, M., Fehr, S.: Adaptively secure feldman VSS and applications to universally-composable threshold cryptography. In: Franklin, M. (ed.) CRYPTO 2004. LNCS, vol. 3152, pp. 317–334 (Aug 2004). https://doi.org/10.1007/978-3-540-28628-8_20
2. Arun, A., Ganesh, C., Lokam, S.V., Mopuri, T., Sridhar, S.: Dew: A transparent constant-sized polynomial commitment scheme. In: Boldyreva, A., Kolesnikov, V. (eds.) PKC 2023, Part II. LNCS, vol. 13941, pp. 542–571 (May 2023). https://doi.org/10.1007/978-3-031-31371-4_19
3. Baird, L., Garg, S., Jain, A., Mukherjee, P., Sinha, R., Wang, M., Zhang, Y.: Threshold signatures in the multiverse. In: 2023 IEEE Symposium on Security and Privacy. pp. 1454–1470. IEEE Computer Society Press (May 2023). https://doi.org/10.1109/SP46215.2023.10179436
4. Bauer, B., Fuchsbauer, G., Loss, J.: A classification of computational assumptions in the algebraic group model. In: Micciancio, D., Ristenpart, T. (eds.) CRYPTO 2020, Part II. LNCS, vol. 12171, pp. 121–151 (Aug 2020). https://doi.org/10.1007/978-3-030-56880-1_5
5. Ben-Sasson, E., Bentov, I., Horesh, Y., Riabzev, M.: Fast reed-solomon interactive oracle proofs of proximity. In: Chatzigiannakis, I., Kaklamanis, C., Marx, D., Sannella, D. (eds.) ICALP 2018. LIPIcs, vol. 107, pp. 14:1–14:17. Schloss Dagstuhl (Jul 2018). https://doi.org/10.4230/LIPIcs.ICALP.2018.14
6. Ben-Sasson, E., Chiesa, A., Green, M., Tromer, E., Virza, M.: Secure sampling of public parameters for succinct zero knowledge proofs. In: 2015 IEEE Symposium on Security and Privacy. pp. 287–304. IEEE Computer Society Press (May 2015). https://doi.org/10.1109/SP.2015.25
7. Benabbas, S., Gennaro, R., Vahlis, Y.: Verifiable delegation of computation over large datasets. In: Rogaway, P. (ed.) CRYPTO 2011. LNCS, vol. 6841, pp. 111–131 (Aug 2011). https://doi.org/10.1007/978-3-642-22792-9_7
8. Benaloh, J.C., de Mare, M.: One-way accumulators: A decentralized alternative to digital sinatures (extended abstract). In: Helleseth, T. (ed.) EUROCRYPT'93. LNCS, vol. 765, pp. 274–285 (May 1994). https://doi.org/10.1007/3-540-48285-7_24
9. Boneh, D., Drake, J., Fisch, B., Gabizon, A.: Halo infinite: Proof-carrying data from additive polynomial commitments. In: Malkin, T., Peikert, C. (eds.) CRYPTO 2021, Part I. LNCS, vol. 12825, pp. 649–680. Virtual Event (Aug 2021). https://doi.org/10.1007/978-3-030-84242-0_23
10. Boneh, D., Drijvers, M., Neven, G.: Compact multi-signatures for smaller blockchains. In: Peyrin, T., Galbraith, S. (eds.) ASIACRYPT 2018, Part II. LNCS, vol. 11273, pp. 435–464 (Dec 2018). https://doi.org/10.1007/978-3-030-03329-3_15
11. Boneh, D., Lynn, B., Shacham, H.: Short signatures from the Weil pairing. In: Boyd, C. (ed.) ASIACRYPT 2001. LNCS, vol. 2248, pp. 514–532 (Dec 2001). https://doi.org/10.1007/3-540-45682-1_30
12. Boneh, D., Partap, A., Waters, B.: Accountable multi-signatures with constant size public keys. To appear at PKC 2025 (2023), https://eprint.iacr.org/2023/1793
13. Bootle, J., Cerulli, A., Chaidos, P., Groth, J., Petit, C.: Efficient zero-knowledge arguments for arithmetic circuits in the discrete log setting. In: Fischlin, M., Coron, J.S. (eds.) EUROCRYPT 2016, Part II. LNCS, vol. 9666, pp. 327–357 (May 2016). https://doi.org/10.1007/978-3-662-49896-5_12
14. Bünz, B., Bootle, J., Boneh, D., Poelstra, A., Wuille, P., Maxwell, G.: Bulletproofs: Short proofs for confidential transactions and more. In: 2018 IEEE Symposium on Security and Privacy. pp. 315–334. IEEE Computer Society Press (May 2018). https://doi.org/10.1109/SP.2018.00020

15. Bünz, B., Maller, M., Mishra, P., Tyagi, N., Vesely, P.: Proofs for inner pairing products and applications. In: Tibouchi, M., Wang, H. (eds.) ASIACRYPT 2021, Part III. LNCS, vol. 13092, pp. 65–97 (Dec 2021). https://doi.org/10.1007/978-3-030-92078-4_3
16. Burdges, J., Ciobotaru, O., Alper, H.K., Stewart, A., Vasilyev, S.: Ring verifiable random functions and zero-knowledge continuations. Cryptology ePrint Archive, Paper 2023/002 (2023), https://eprint.iacr.org/2023/002
17. Camenisch, J., Kohlweiss, M., Soriente, C.: An accumulator based on bilinear maps and efficient revocation for anonymous credentials. In: Jarecki, S., Tsudik, G. (eds.) PKC 2009. LNCS, vol. 5443, pp. 481–500 (Mar 2009). https://doi.org/10.1007/978-3-642-00468-1_27
18. Campanelli, M., Nitulescu, A., Ràfols, C., Zacharakis, A., Zapico, A.: Linear-map vector commitments and their practical applications. In: Agrawal, S., Lin, D. (eds.) ASIACRYPT 2022, Part IV. LNCS, vol. 13794, pp. 189–219 (Dec 2022). https://doi.org/10.1007/978-3-031-22972-5_7
19. Catalano, D., Fiore, D.: Vector commitments and their applications. In: Kurosawa, K., Hanaoka, G. (eds.) PKC 2013. LNCS, vol. 7778, pp. 55–72 (Feb / Mar 2013). https://doi.org/10.1007/978-3-642-36362-7_5
20. Chaidos, P., Kiayias, A.: Mithril: Stake-based threshold multisignatures. pp. 239–263. LNCS (Oct 2024). https://doi.org/10.1007/978-981-97-8013-6_11
21. Chiesa, A., Hu, Y., Maller, M., Mishra, P., Vesely, P., Ward, N.P.: Marlin: Preprocessing zkSNARKs with universal and updatable SRS. In: Canteaut, A., Ishai, Y. (eds.) EUROCRYPT 2020, Part I. LNCS, vol. 12105, pp. 738–768 (May 2020). https://doi.org/10.1007/978-3-030-45721-1_26
22. Choi, K., Manoj, A., Bonneau, J.: SoK: Distributed randomness beacons. In: 2023 IEEE Symposium on Security and Privacy. pp. 75–92. IEEE Computer Society Press (May 2023). https://doi.org/10.1109/SP46215.2023.10179419
23. Chu, H., Fiore, D., Kolonelos, D., Schröder, D.: Inner product functional commitments with constant-size public parameters and openings. pp. 639–662. LNCS (2022). https://doi.org/10.1007/978-3-031-14791-3_28
24. Das, S., Camacho, P., Xiang, Z., Nieto, J., Bünz, B., Ren, L.: Threshold signatures from inner product argument: Succinct, weighted, and multi-threshold. In: ACM CCS 2023. pp. 356–370. ACM Press (Nov 2023). https://doi.org/10.1145/3576915.3623096
25. Das, S., Pinkas, B., Tomescu, A., Xiang, Z.: Distributed randomness using weighted VUFs. pp. 314–344. LNCS (Jun 2025). https://doi.org/10.1007/978-3-031-91098-2_12
26. Das, S., Yurek, T., Xiang, Z., Miller, A.K., Kokoris-Kogias, L., Ren, L.: Practical asynchronous distributed key generation. In: 2022 IEEE Symposium on Security and Privacy. pp. 2518–2534. IEEE Computer Society Press (May 2022). https://doi.org/10.1109/SP46214.2022.9833584
27. Daza, V., Ràfols, C., Zacharakis, A.: Updateable inner product argument with logarithmic verifier and applications. In: Kiayias, A., Kohlweiss, M., Wallden, P., Zikas, V. (eds.) PKC 2020, Part I. LNCS, vol. 12110, pp. 527–557 (May 2020). https://doi.org/10.1007/978-3-030-45374-9_18
28. Dodis, Y.: Efficient construction of (distributed) verifiable random functions. In: Desmedt, Y. (ed.) PKC 2003. LNCS, vol. 2567, pp. 1–17 (Jan 2003). https://doi.org/10.1007/3-540-36288-6_1
29. Fuchsbauer, G., Kiltz, E., Loss, J.: The algebraic group model and its applications. In: Shacham, H., Boldyreva, A. (eds.) CRYPTO 2018, Part II. LNCS, vol. 10992, pp. 33–62 (Aug 2018). https://doi.org/10.1007/978-3-319-96881-0_2
30. Galindo, D., Liu, J., Ordean, M., Wong, J.: Fully distributed verifiable random functions and their application to decentralised random beacons. In: IEEE European Symposium on Security and Privacy, EuroS&P 2021, Vienna, Austria, September 6-10, 2021. pp. 88–102. IEEE (2021). https://doi.org/10.1109/EUROSP51992.2021.00017, https://doi.org/10.1109/EuroSP51992.2021.00017
31. Galindo, D., Liu, J., Ordean, M., Wong, J.M.: Fully distributed verifiable random functions and their application to decentralised random beacons. pp. 88–102 (2021). https://doi.org/10.1109/EuroSP51992.2021.00017
32. Garg, S., Goel, A., Wang, M.: How to prove statements obliviously? pp. 449–487. LNCS (Aug 2024). https://doi.org/10.1007/978-3-031-68403-6_14
33. Garg, S., Jain, A., Mukherjee, P., Sinha, R., Wang, M., Zhang, Y.: hinTS: Threshold signatures with silent setup. In: 2024 IEEE Symposium on Security and Privacy. pp. 3034–3052. IEEE Computer Society Press (May 2024). https://doi.org/10.1109/SP54263.2024.00057
34. Gennaro, R., Jarecki, S., Krawczyk, H., Rabin, T.: Secure distributed key generation for discrete-log based cryptosystems. Journal of Cryptology **20**(1), 51–83 (Jan 2007). https://doi.org/10.1007/s00145-006-0347-3
35. Gentry, C., Halevi, S., Lyubashevsky, V.: Practical non-interactive publicly verifiable secret sharing with thousands of parties. In: Dunkelman, O., Dziembowski, S. (eds.) EUROCRYPT 2022, Part I. LNCS, vol. 13275, pp. 458–487 (May / Jun 2022). https://doi.org/10.1007/978-3-031-06944-4_16
36. Gorbunov, S., Reyzin, L., Wee, H., Zhang, Z.: Pointproofs: Aggregating proofs for multiple vector commitments. In: Ligatti, J., Ou, X., Katz, J., Vigna, G. (eds.) ACM CCS 2020. pp. 2007–2023. ACM Press (Nov 2020). https://doi.org/10.1145/3372297.3417244

37. Groth, J.: Linear algebra with sub-linear zero-knowledge arguments. In: Halevi, S. (ed.) CRYPTO 2009. LNCS, vol. 5677, pp. 192–208 (Aug 2009). https://doi.org/10.1007/978-3-642-03356-8_12
38. Groth, J.: On the size of pairing-based non-interactive arguments. In: Fischlin, M., Coron, J.S. (eds.) EUROCRYPT 2016, Part II. LNCS, vol. 9666, pp. 305–326 (May 2016). https://doi.org/10.1007/978-3-662-49896-5_11
39. Groth, J., Kohlweiss, M., Maller, M., Meiklejohn, S., Miers, I.: Updatable and universal common reference strings with applications to zk-SNARKs. In: Shacham, H., Boldyreva, A. (eds.) CRYPTO 2018, Part III. LNCS, vol. 10993, pp. 698–728 (Aug 2018). https://doi.org/10.1007/978-3-319-96878-0_24
40. Kate, A., Mukherjee, P., Samanta, S., Sarkar, P.: Dyna-hinTS: Silent threshold signatures for dynamic committees. Cryptology ePrint Archive, Paper 2025/631 (2025), https://eprint.iacr.org/2025/631
41. Kate, A., Zaverucha, G.M., Goldberg, I.: Constant-size commitments to polynomials and their applications. In: Abe, M. (ed.) ASIACRYPT 2010. LNCS, vol. 6477, pp. 177–194 (Dec 2010). https://doi.org/10.1007/978-3-642-17373-8_11
42. Katz, J.: Round-optimal, fully secure distributed key generation. pp. 285–316. LNCS (Aug 2024). https://doi.org/10.1007/978-3-031-68394-7_10
43. Kavousi, A., Wang, Z., Jovanovic, P.: SoK: Public randomness. pp. 216–234 (2024). https://doi.org/10.1109/EuroSP60621.2024.00020
44. Khovratovich, D., Rothblum, R.D., Soukhanov, L.: How to prove false statements: Practical attacks on fiat-shamir. Cryptology ePrint Archive, Paper 2025/118 (2025), https://eprint.iacr.org/2025/118
45. Krenn, S., Mir, O., Slamanig, D.: Structure-preserving compressing primitives: Vector commitments, accumulators and applications. Cryptology ePrint Archive (2024)
46. Lai, R.W.F., Malavolta, G.: Subvector commitments with application to succinct arguments. In: Boldyreva, A., Micciancio, D. (eds.) CRYPTO 2019, Part I. LNCS, vol. 11692, pp. 530–560 (Aug 2019). https://doi.org/10.1007/978-3-030-26948-7_19
47. Lai, R.W.F., Malavolta, G., Ronge, V.: Succinct arguments for bilinear group arithmetic: Practical structure-preserving cryptography. In: Cavallaro, L., Kinder, J., Wang, X., Katz, J. (eds.) ACM CCS 2019. pp. 2057–2074. ACM Press (Nov 2019). https://doi.org/10.1145/3319535.3354262
48. Lee, J.: Dory: Efficient, transparent arguments for generalised inner products and polynomial commitments. In: Nissim, K., Waters, B. (eds.) TCC 2021, Part II. LNCS, vol. 13043, pp. 1–34 (Nov 2021). https://doi.org/10.1007/978-3-030-90453-1_1
49. Lehmann, A., Özbay, C.: Commit-and-prove system for vectors and applications to threshold signing. Cryptology ePrint Archive (2025)
50. Lehmann, A., Özbay, C.: Commit-and-prove system for vectors and applications to threshold signing. Cryptology ePrint Archive, Paper 2025/355 (2025), https://eprint.iacr.org/2025/355
51. Libert, B., Ramanna, S.C., Yung, M.: Functional commitment schemes: From polynomial commitments to pairing-based accumulators from simple assumptions. In: Chatzigiannakis, I., Mitzenmacher, M., Rabani, Y., Sangiorgi, D. (eds.) ICALP 2016. LIPIcs, vol. 55, pp. 30:1–30:14. Schloss Dagstuhl (Jul 2016). https://doi.org/10.4230/LIPIcs.ICALP.2016.30
52. Libert, B., Yung, M.: Concise mercurial vector commitments and independent zero-knowledge sets with short proofs. In: Micciancio, D. (ed.) TCC 2010. LNCS, vol. 5978, pp. 499–517 (Feb 2010). https://doi.org/10.1007/978-3-642-11799-2_30
53. Lipmaa, H., Siim, J., Zajac, M.: Counting vampires: From univariate sumcheck to updatable ZK-SNARK. In: Agrawal, S., Lin, D. (eds.) ASIACRYPT 2022, Part II. LNCS, vol. 13792, pp. 249–278 (Dec 2022). https://doi.org/10.1007/978-3-031-22966-4_9
54. Liskov, M.: Updatable zero-knowledge databases. In: Roy, B.K. (ed.) ASIACRYPT 2005. LNCS, vol. 3788, pp. 174–198 (Dec 2005). https://doi.org/10.1007/11593447_10
55. Liu, J., Manulis, M.: Fast SNARK-based non-interactive distributed verifiable random function with ethereum compatibility. Cryptology ePrint Archive, Report 2024/968 (2024), https://eprint.iacr.org/2024/968
56. Malavolta, G.: Key-homomorphic and aggregate verifiable random functions. pp. 98–129. LNCS (Nov 2024). https://doi.org/10.1007/978-3-031-78023-3_4
57. Micali, S., Rabin, M.O., Vadhan, S.P.: Verifiable random functions. In: 40th FOCS. pp. 120–130. IEEE Computer Society Press (Oct 1999). https://doi.org/10.1109/SFFCS.1999.814584
58. Micali, S., Reyzin, L., Vlachos, G., Wahby, R.S., Zeldovich, N.: Compact certificates of collective knowledge. In: 2021 IEEE Symposium on Security and Privacy. pp. 626–641. IEEE Computer Society Press (May 2021). https://doi.org/10.1109/SP40001.2021.00096
59. Nikolaenko, V., Ragsdale, S., Bonneau, J., Boneh, D.: Powers-of-tau to the people: Decentralizing setup ceremonies. pp. 105–134. LNCS (Jun 2024). https://doi.org/10.1007/978-3-031-54776-8_5
60. Nitulescu, A.: Sok: Vector commitments (2021), https://www.di.ens.fr/~nitulesc/files/vc-sok.pdf
61. Papamanthou, C., Shi, E., Tamassia, R.: Signatures of correct computation. In: Sahai, A. (ed.) TCC 2013. LNCS, vol. 7785, pp. 222–242 (Mar 2013). https://doi.org/10.1007/978-3-642-36594-2_13
62. Shoup, V.: Lower bounds for discrete logarithms and related problems. In: Fumy, W. (ed.) EUROCRYPT'97. LNCS, vol. 1233, pp. 256–266 (May 1997). https://doi.org/10.1007/3-540-69053-0_18

63. Towa, P., Vergnaud, D.: Succinct diophantine-satisfiability arguments. In: Moriai, S., Wang, H. (eds.) ASIACRYPT 2020, Part III. LNCS, vol. 12493, pp. 774–804 (Dec 2020). https://doi.org/10.1007/978-3-030-64840-4_26
64. Waters, B.R.: Efficient identity-based encryption without random oracles. In: Cramer, R. (ed.) EUROCRYPT 2005. LNCS, vol. 3494, pp. 114–127 (May 2005). https://doi.org/10.1007/11426639_7

A Python Implementation Details

We have implemented our main primitives, protocols, and the dynamic threshold VRF scheme in Python. The implementation focuses on correctness and reproducibility rather than optimization or deployment-level readiness. We provide a Python prototype based upon the `bplib` library[8] and petlib[9] with `OpenSSL` bindings[10] for pairing-based cryptography. They use the popular pairing friendly curve BN256 which provides efficient type-3 bilinear groups at a security level of around 100 bits. Our measurements have been performed on an Intel i7 CPU @ 3.20GHz and 16 GB RAM, running Ubuntu 22.04 and Python 3.11. Each test was averaged over 50 runs for accuracy.

GSFC and GSIPA. We first benchmarked the GSFC and GSIPA schemes (see Table 6). We show the mean execution time of each algorithm, following a clean separation between components such as setup, commitment, proof generation, verification, and the MsgGen algorithm used to create messages (cf. Eq. 1 in Sec. 2.6), in order to facilitate benchmarking precisely. These were tested for vector sizes $q \in [10, 100]$ to observe performance scaling.

As detailed in Table 6 that reports our implementation, Setup and MsgGen phases are the most computationally expensive ones. However, both are *one-time preprocessing steps* and do not impact the efficiency of proof generation or verification during actual use. Additionally, in GSFC, verification is constant-time with respect to q. In our implementation, we *precompute* the term $\prod_{i\in[q]} \hat{B}_{q+1-i}^{b_i}$ and pass it to the verifier. This aligns with the construction, where the verifier can perform only a *fixed number of pairing checks*, independent of q. Since the vector $\mathbf{b}$ is public, this term can be precomputed outside the verification phase. We note that practical runtimes may slightly increase due to implementation-level factors such as memory access patterns, interpreter overhead, and cryptographic library internals.

For GSIPA, the setup, message generation, and commitment phases (we compute $\hat{C}_b$ during the proof to align the commitment structure with GSFC—though it could also be computed during the commitment phase) exhibit similar runtimes to those of the GSFC scheme. The commitment time remains nearly constant across different values of q, as the costly exponentiations are already performed during message generation, leaving only lightweight group additions in the commitment step. Therefore, only the proof generation (Proof) and verification (Verify) times are reported here.

Table 6. Benchmark Results for GSFC and GSIPA (in seconds)

q	GSFC					GSIPA	
	Setup	MsgGen	Commit	Open	Verify	Proof	Verify
10	0.1426	0.0194	0.0002	0.0076	0.0043	0.0642	0.0063
20	0.4051	0.0738	0.0003	0.0160	0.0044	0.2333	0.0065
30	0.9052	0.1672	0.0003	0.0252	0.0043	0.5273	0.0065
40	1.6823	0.3050	0.0004	0.0326	0.0041	0.9229	0.0063
50	2.3733	0.5198	0.0004	0.0477	0.0044	1.3708	0.0067
70	4.6297	0.9416	0.0005	0.0750	0.0046	2.6913	0.0068
100	9.5852	1.9971	0.0007	0.1192	0.0045	5.2782	0.0065

Dynamic Threshold VRF. Additionally, we also benchmarked our dynamic threshold VRF from Sec. 5.1. We provide the mean execution time of each algorithm of our DT-VRF in Table 7. The results

[8] https://github.com/gdanezis/bplib
[9] https://github.com/gdanezis/petlib
[10] https://github.com/dfaranha/OpenPairing

Table 7. Benchmark Results for the DT-VRF with IPA-based Proofs (in seconds)

$q = \mathsf{thr}$	Eval	EvalVerify	`Commit + Proof avk`	`Commit + Proof Threshold`	Verify
10	0.05	0.05	0.07	0.31	0.04
20	0.10	0.10	0.35	1.00	0.03
30	0.14	0.16	0.59	2.16	0.08
40	0.23	0.21	1.05	3.87	0.04
50	0.30	0.25	1.45	5.65	0.03
70	0.34	0.35	2.94	10.53	0.04
100	0.55	0.65	5.64	19.06	0.03

already demonstrate that the schemes are very efficient. However, we note that all measurements were taken without any optimizations, and we believe that with thorough code-level optimizations, the performance could be significantly improved.

As shown in Table 7 we implemented our VRF, IPA for public keys (labeled `Commit + Proof avk`) i.e., $\langle \mathsf{avk}, \mathbf{b} \rangle$, an IPA for threshold evaluation (`Commit + Proof Threshold`) i.e., $\langle \mathbf{w}, \mathbf{b} \rangle$, and an HPA i.e., $\langle \mathbf{u}, \boldsymbol{\alpha} \rangle = z$ and $\langle \mathbf{b}, \boldsymbol{\gamma} \rangle = z$. To compute the IPA between two scalar vectors, we employed our scalar-based IPA variant of our GSIPA, where the commitment $C_{\mathbf{A}}$ can be directly computed using messages in scalars without relying on structured messages. For instance, the commitment in $\mathbb{G}_1$ for $\mathbf{w} \in \mathbb{Z}_p$ can be commuted $C_{\mathbf{w}} = \prod_{i \in [q]} B_i^{w_i}$. This case resembles a subvector opening from Sec. 4.1 (with an additional $\hat{\pi}$ for the degree check). Note that alternative IPA constructions for scalar vectors, such as those in [53,18], could also be used here.

We omit the reports for KeyGen, AggKey, AggProof, and AggVerify, as their runtime is negligible (close to zero) compared to the overall protocol execution. These operations involve simple group additions or multiplications and do not significantly impact the performance, even as the parameter q increases. We also exclude the MsgGen time, since it is identical to the message generation cost reported in the GSFC and GSIPA benchmarks. The columns labeled `Commit + Proof avk` and `Commit + Proof Threshold` indicate the time required to compute the commitment to the respective vectors and generate the corresponding proofs. Again, the total verification time remains constant at around 0.04s. For simplicity, we consider the threshold thr as the worst-case scenario, where it is equal to q, however, in many real-world applications, it is typically set to a lower bound such as $q/2 + 1$.

B Aggregate VRFs

B.1 Defining Aggregate VRFs

Definition 21 (Aggregate VRF [56]). *An aggregate verifiable random function (VRF) is a tuple of polynomial-time algorithms with the following syntax.*

$\mathsf{Setup}(1^\lambda)$: *On input the security parameter $\lambda \in \mathbb{N}$, the setup algorithm returns the common reference string* crs.

$\mathsf{KeyGen}(\mathsf{crs})$: *On input the common reference string* crs*, the generation algorithm samples a secret key k and returns $k \in \mathcal{K}$ and the verification key $\mathsf{vk} = \mathsf{VKey}(k)$, where* VKey *is a bijective function.*

$\mathsf{Eval}(k, x)$: *On input the secret key $k \in \mathcal{K}$ and some point $x \in \{0,1\}^\lambda$, the evaluation algorithm returns an image $y \in \mathcal{Y}$.*

$\mathsf{Prove}(k, x)$: *On input the secret key $k \in \mathcal{K}$ and some point $x \in \{0,1\}^\lambda$, the prover algorithm returns a proof $\pi \in \mathcal{P}$.*

$\mathsf{Verify}(\mathsf{vk}, x, y, \pi)$: *On input the verification key* vk*, some point $x \in \{0,1\}^\lambda$, an image $y \in \mathcal{Y}$, and a proof $\pi \in \mathcal{P}$, the verification algorithm returns a bit $b \in \{0,1\}$ denoting acceptance or rejection.*

$\mathsf{Agg}(x, \{\mathsf{vk}_i, y_i\}_{i=1}^n)$: *On input a point $x \in \{0,1\}^\lambda$, n verification keys $(\mathsf{vk}_1, \ldots, \mathsf{vk}_n)$ and images $(y_1, \ldots, y_n) \in \mathcal{Y}^n$, the aggregation algorithm outputs an aggregate key $\tilde{\mathsf{vk}}$ and an aggregate image $\tilde{y}$.*

$\mathsf{AggProve}(x, \{\mathsf{vk}_i, \pi_i\}_{i=1}^n)$: *On input a point $x \in \{0,1\}^\lambda$, n verification keys $(\mathsf{vk}_1, \ldots, \mathsf{vk}_n)$, and proofs $(\pi_1, \ldots, \pi_n) \in \mathcal{P}^n$, the aggregate proof algorithm outputs an aggregate proof $\tilde{\pi}$.*

$\mathsf{AggVerify}(\{\mathsf{vk}_i\}_{i=1}^n, x, \tilde{y}, \tilde{\pi})$: *On input n verification keys $(\mathsf{vk}_1, \ldots, \mathsf{vk}_n)$, a point $x \in \{0,1\}^\lambda$, and an aggregate image-proof pair $(\tilde{y}, \tilde{\pi})$, the aggregate verification algorithm outputs a bit $b \in \{0,1\}$ denoting acceptance or rejection.*

B.2 Security of Aggregate VRFs

We provide the formal definitions of aggregate pseudorandomness, aggregate binding and uniqueness from [56] as follows:

Aggregate Pseudorandomness. Aggregate Pseudorandomness ensures that any output of the aggregate VRF is still pseudorandom, so long as at least one of the parties involved is honest. This means that the output of the aggregation algorithm should still satisfy pseudorandomness, even if some of the keys are controlled by the $\mathcal{A}$.

Definition 22 (Aggregate Pseudorandomness). *A* VRF *satisfies* aggregate pseudorandomness *if there exists a negligible function ϵ such that for $\lambda \in \mathbb{N}$ and all admissible PPT adversaries $\mathcal{A}$, it holds that*

$$\left| \frac{1}{2} - \Pr\left[1 \leftarrow \mathsf{ExpAggRandom}_{\mathcal{A}}(1^\lambda)\right] \right| \leq \epsilon(\lambda)$$

where the experiment $\mathsf{ExpAggRandom}_{\mathcal{A}}$ *is defined as follows:*

1. *The challenger samples a reference string* $\mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda)$ *and a key pair* $(k, vk) \leftarrow \mathsf{KeyGen}(\mathsf{crs})$ *and sends* (crs, vk) *to* $\mathcal{A}$.
2. $\mathcal{A}$ *can query adaptively and at any time an oracle on input* x_i *and receives back from the challenger a pair* (y_i, π_i) *where* $y_i \leftarrow \mathsf{Eval}(k, x_i)$ *and* $\pi_i \leftarrow \mathsf{Prove}(k, x_i)$.
3. *At any point (including between evaluation queries),* $\mathcal{A}$ *can query a challenge input* x^* *along with some challenge keys* $\{vk_i\}_{i=1}^n$. *The challenger (inefficiently) computes* $k_i \leftarrow \mathsf{VKey}^{-1}(vk_i)$ *and sets* $y_i \leftarrow \mathsf{Eval}(k_i, x^*)$. *Then it flips a coin* $b \leftarrow \{0,1\}$ *and computes:*

$$y^* \leftarrow \begin{cases} \mathsf{Agg}(x^*, vk, \{\mathsf{Eval}(k, x^*), \{vk_i, y_i\}_{i=1}^n\}) & \text{if } b = 0 \\ Y \leftarrow\!\!\$\ D & \text{if } b = 1 \end{cases}$$

4. *At the end of the experiment,* $\mathcal{A}$ *outputs a guess* b^*. *The experiment returns 1 if and only if* $b^* = b$.

Furthermore, we say that $\mathcal{A}$ *is* admissible *if* $x^* \notin Q$, *where* Q *denotes the set of points queried to the oracle as defined above.*

Aggregate Binding. Aggregate binding requires that no attacker is able to find two different preimages for a given aggregate. This property is useful in settings where one stores an aggregate proof, and wants to prevent changes the corresponding images after the fact.

Definition 23 (Aggregate Binding). *A VRF satisfies* aggregate binding *if there exists a negligible function ϵ such that for $\lambda \in \mathbb{N}$ and all PPT adversaries $\mathcal{A}$ it holds that*

$$\Pr\left[1 \leftarrow \mathsf{ExpAggBind}_{\mathcal{A}}(1^\lambda)\right] \leq \epsilon(\lambda)$$

where the experiment $\mathsf{ExpAggBind}_{\mathcal{A}}$ *is defined as follows.*

1. *The challenger samples a reference string* $\mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda)$ *and sends it to* $\mathcal{A}$.
2. $\mathcal{A}$ *returns a point* x^*, n *verification keys* $\mathsf{vk}_1, \ldots, \mathsf{vk}_n$, *two tuples* $(y_1, \ldots, y_n)$ *and* $(y_1^*, \ldots, y_n^*)$, *and proofs* $(\pi_1, \ldots, \pi_n)$.
3. *The challenger computes* $\tilde{\pi} \leftarrow \mathsf{AggProve}(x^*, \{\mathsf{vk}_i, \pi_i\}_{i=1}^n)$ *and* $\tilde{y} \leftarrow \mathsf{Agg}(x^*, \{\mathsf{vk}_i, y_i^*\}_{i=1}^n)$ *and outputs* 1 *if and only if:*

$$\{\mathsf{Verify}(\mathsf{vk}_i, x^*, y_i, \pi_i) = 1\}_{i=1}^n$$

and

$$\mathsf{AggVerify}(\{\mathsf{vk}_i\}_{i=1}^n, x^*, \tilde{y}, \tilde{\pi}) = 1$$

and furthermore $(y_1, \ldots, y_n) \neq (y_1^*, \ldots, y_n^*)$.

Aggregate Uniqueness. We also require that Aggregate Uniqueness. In particular, it must satisfy uniqueness, which states that for a given aggregate image, there must exist a unique aggregate proof—i.e., the output of the aggregation algorithm must still constitute a valid VRF. We consider this property below, starting with the notion of Uniqueness, and then giving Aggregate Uniqueness.

Definition 24 (Uniqueness). *A VRF satisfies* uniqueness *if for $\lambda \in \mathbb{N}$, all* crs *in the support of* $\mathsf{Setup}(1^\lambda)$*, all verification keys vk, all points $x \in \{0,1\}^\lambda$, and all images $(y_0, y_1) \in Y^2$ and proofs $(\pi_0, \pi_1) \in P^2$, it holds that*

$$1 = \mathsf{Verify}(vk, x, y_0, \pi_0) = \mathsf{Verify}(vk, x, y_1, \pi_1) \implies y_0 = y_1.$$

Definition 25 (Aggregate Uniqueness). *A VRF satisfies* aggregate uniqueness *if for $\lambda \in \mathbb{N}$, all polynomials $n = n(\lambda)$, all* crs *in the support of Setup(1^λ), all verification keys $\{vk_i\}$, and all points $x \in \{0,1\}^\lambda$, it holds that $\tilde{vk}$ satisfies uniqueness (Def. 24), where*

$$(\tilde{vk}, \tilde{y}) \leftarrow \mathsf{Agg}\left(x, \{vk_i, \mathsf{Eval}(k_i, x)\}_{i=1}^n\right).$$

C Security Proofs

We provide the security proofs of our schemes here.

C.1 Proof of GSFC Functional Binding Theorem 1

Proof. Let $\mathcal{A}$ be an algebraic adversary that constructs a commitment $(C, \mathbf{a})$ for a public vector $\mathbf{b}$, an index $i \in \{1, \ldots, q\}$, and a valid opening $(\pi, \mathbf{r})$ to a message $(Y, \mathbf{y})$. Here, $\mathbf{y}'$, $\mathbf{a}$, and $\mathbf{r}$ represent the group elements Y, C, and π, respectively.

Assume that $R(X)$ and $C(X)$ are the polynomial representations of the commitment C and proof π such that

$$R(X) = \sum_{j \in [2q] \setminus \{q+1\}} r_j \cdot X^j \quad \text{and} \quad C(X) = \sum_{j \in [2q] \setminus \{q+1\}} a_j \cdot X^j.$$

Similarly, for the messages,

$$Y(X) = \sum_{j \in [2q] \setminus \{q+1\}} y'_j \cdot X^j, \quad \text{setting } y = \sum_{j \in [2q] \setminus \{q+1\}} y'_j.$$

From the verification equations, it follows that:

$$e\left(C, \prod_{i=1}^{q} \hat{B}_{q+1-i}^{b_i}\right) = e\left(\pi_y, \hat{P}\right) \cdot e\left(Y, \hat{B}_{q+1}\right).$$

Since the vector $\mathbf{b}$ is publicly known and the verifier checks this condition. We can compute the related polynomial as:

$$B(X) = \sum_{i \in [q]} b_i \cdot X^{q+1-i}.$$

By substituting the polynomial representations and using exponentiation, we obtain: $P(X) = C(X)B(X) - y \cdot X^{q+1} - R(X)$. Comparing these polynomials:

$$\left(\sum_{j \in [2q] \setminus \{q+1\}} a_j \cdot X^j\right)\left(\sum_{i \in [q]} b_i \cdot X^{q+1-i}\right) - y \cdot X^{q+1} - R(X) = 0.$$

From the verification eq. either α is a root of $P(X)$ or $P(X) = 0$.

Case 1: $P(X) = 0$ In this case, we have:

$$\left(\sum_{j \in [2q] \setminus \{q+1\}} a_j \cdot X^j\right)\left(\sum_{i \in [q]} b_i \cdot X^{q+1-i}\right) = y \cdot X^{q+1} + R(X).$$

Expanding the left-hand side:

$$\sum_{j \in [2q] \setminus \{q+1\}} \sum_{i \in [q]} a_j \cdot b_i \cdot X^{j+(q+1-i)} = y \cdot X^{q+1} + R(X).$$

Extracting X^{q+1} from the left-hand side:

$$\sum_{j\in[q]} a_j \cdot b_j X^{q+1} + \left(\sum_{j\in[2q]\setminus\{q+1\}} \sum_{i\in[q],j\neq i} a_j \cdot b_i X^{j-i} \right) = y \cdot X^{q+1} + R(X).$$

The terms contributing to X^{q+1} are those where $j = i$. So to find the coefficient of X^{q+1} in the expansion, we substitute $j = i$ in the summation $\sum_{j\in[2q]\setminus\{q+1\},j=i\in[q]} a_j \cdot b_i$. Since $j = i$, we sum over j in $[q]$: $\sum_{j\in[q]} a_j \cdot b_j$. Now let assume

$H(X) = \left(\sum_{j\in[2q]\setminus\{q+1\}} \sum_{i\in[q],i\neq j} a_j \cdot b_i X^{j-i}\right)$ as a polynomial with the degree at most $2q$ (i.e., the maximum exponent occurs when j is at its largest ($j = 2q$) and i is at its smallest $i = 0$ so we have $\max(j - i) = 2q$). Rearrange:

$$\sum_{j\in[q]} a_j \cdot b_j X^{q+1} = y \cdot X^{q+1} + R(X) - H(X)$$

Since both sides contain X^{q+1}, their coefficients must be equal:

$$\sum_{j\in[q]} a_j \cdot b_j = y.$$

Thus, there exists vectors $\mathbf{a}$ and $\mathbf{b}$ such that $\langle \mathbf{a}, \mathbf{b} \rangle = y$. This implies that $\mathcal{A}$ loses. Thus we consider the case 2 as follows:

Case 2: $P(X) \neq 0$ and $P(\alpha) = 0$. In this case, we can construct an adversary $\mathcal{B}$ against the q-discrete logarithm (q-DL) assumption.

Given P^{α} in $\mathbb{G}_1$, $\mathcal{B}$ computes all roots of $P(X)$ and efficiently identifies the one corresponding to P^{α}. Since $P(X)$ has at most $2q$ roots in $\mathbb{Z}_p$ and $2q < p$, root-finding algorithms run in polynomial time. As $P(\alpha) = 0$, then α must be one of these roots, allowing $\mathcal{B}$ to recover α, contradicting the q-DL assumption. □

C.2 Proof of GSIPA Knowledge-Soundness Theorem 2

Proof. Completeness follows from the correctness of functional commitment. We aim to show that for any efficient prover $\mathcal{P}^*$ that produces an accepting proof with non-negligible probability, there exists an efficient extractor $E_{\mathcal{P}^*}$ that can extract the witness vector $\mathbf{A}, \mathbf{b}$ such that $Y = \langle \mathbf{A}, \mathbf{b} \rangle$ or we can break the q-DL. We prove its Knowledge-Soundness which follows the binding property of the commitments. Let $\mathcal{A}$ be an algebraic adversary that constructs a commitment $(C_a, C_b, \mathbf{a}, \mathbf{b}')$, a valid opening $(\pi, \hat{\pi}, \mathbf{r}, \mathbf{c})$ to a message $(Y', \mathbf{y}')$. Here, $\mathbf{y}'$, $\mathbf{a}$, $\mathbf{b}'$, $\mathbf{r}$ and $\mathbf{c}$ represent the group elements $Y' \in \mathbb{G}_1$, $C_a \in \mathbb{G}_1$, $C_b \in \mathbb{G}_2$ and $\pi \in \mathbb{G}_1$, $\hat{\pi} \in \mathbb{G}_2$, respectively. Assume that $C(X)$, $B(X)$, $R(X)$ and $\hat{R}(X)$ are the polynomial representations of the commitments C_a, C_b and proof π and $\hat{\pi}$ such that

$$R(X) = \sum_{j\in[2q]\setminus\{q+1\}} r_j \cdot X^j, \quad C(X) = \sum_{j\in[2q]\setminus\{q+1\}} a_j \cdot X^j,$$

$$B(X) = \sum_{j\in[2q]} b'_j \cdot X^j, \quad \hat{R}(X) = \sum_{j\in[2q]} c_j X^j.$$

Similarly, for the messages,

$$Y(X) = \sum_{j\in[2q]\setminus\{q+1\}} y'_j \cdot X^j, \quad \text{assuming } y = \sum_{j\in[2q]} y'_j.$$

Our proof follows three steps:

Step 1: Check the degree of $\hat{C}_b$. We start with the first verification equation:

$$e\left(\prod_{i\in[q]} P^{\alpha^i}, \hat{C}_b \right) = e\left(P, \hat{\pi}\right).$$

We have the polynomial equation representation of verification equation:

$$\left(\sum_{i\in[q]} X^i\right)\left(\sum_{j\in[2q]} b'_j X^j\right) = \sum_{j\in[2q]} c_j X^j.$$

Expanding the left-hand side:

$$\sum_{i\in[q]}\sum_{j\in[2q]} b'_j X^{i+j} = \sum_{j\in[2q]} c_j X^j.$$

Extracting out X^{2q+1}, we obtain:

$$\sum_{j\in[q+1,2q]} b'_j X^{2q+1} + \sum_{i\in[q]}\sum_{j\in[2q],i+j\neq 2q+1} b'_j X^{i+j} = \sum_{j\in[2q]} c_j X^j.$$

We observe that the left-hand side contains a polynomial of degree $3q$, whereas the right-hand side has degree at most $2q$. Thus, we set the coefficients of all terms where $i+j > 2q$ to zero.

We start with X^{2q+1}, we have the coefficients

$$\sum_{j\in[q+1,2q]} b'_j = 0$$

With the help of other terms $\sum_{i\in[q]}\sum_{j\in[2q],i+j\neq 2q+1} b'_j X^{i+j}$ such that $i+j > 2q+1$ we can prove that each $b'_j = 0$ for all $j \in [q+1, 2q]$, we proceed by induction.

Base Case: Consider the highest index $j = 2q$. The equation above states that:

$$b'_{2q} + \sum_{j\in[q+1,2q-1]} b'_j = 0.$$

If $b'_{2q} \neq 0$, there must exist some other $b'_j \neq 0$ to cancel it, but since there are no terms of degree $> 2q$ in the RHS equation and b'_{2q} is the coefficient of the maximum degree $3q$ as $b'_{2q} = X^{3q}$, we must have:

$$b'_{2q} = 0.$$

Inductive Step: Assume that $b'_j = 0$ for all $j > k$, where $k \in [q+1, 2q]$. Consider the next term:

$$b'_k + \sum_{j\in[k+1,2q]} b'_j = 0.$$

By the induction hypothesis, all terms in the summation vanish, leaving:

$$b'_k = 0.$$

By applying this argument recursively for all $j \in [q+1, 2q]$, we conclude that:

$$b'_j = 0 \quad \forall j \in [q+1, 2q].$$

Thus, the polynomial $B(X)$ simplifies to:

$$B(X) = \sum_{j\in[q]} b'_j X^j,$$

Step 2: Extracting witness by considering $\hat{C}_b$ has degree $\leq q$ and the proof is valid: The second verification equation is:

$$e\left(C_A, \hat{C}_b\right) = e\left(\pi_y, \hat{P}\right) \cdot e\left(Y, \hat{B}_{q+1}\right)$$

We start with the given polynomial product:

$$\left(\sum_{\substack{1\leq j\leq 2q \\ j\neq q+1}} a_j X^j\right)\left(\sum_{1\leq i\leq q} b'_i X^i\right) = y \cdot X^{q+1} - R(X).$$

Expanding the left side product, we have:

$$\sum_{\substack{1\le j\le 2q \\ j\neq q+1}} \sum_{1\le i\le q} a_j b'_i X^{j+i}.$$

To find the coefficient corresponding to the term X^{q+1}, we set the exponent equal to $q+1$: $j+i = q+1$. Simplifying this equation gives us: $j = q+1-i$. This means:

$$\sum_{j\in[q]} a_j b'_{q+1-j} X^{q+1} = y X^{q+1}.$$

Lets $b_j = b'_{q+1-j}$ for all $j \in [q]$, hence, the general expression for the coefficient of the term X^{q+1} is:

$$\sum_{j\in[q]} a_j b_j X^{q+1} = y X^{q+1}.$$

Finally, the extractor outputs $\mathbf{a}, \mathbf{b} \in \mathbb{Z}_p^q$ as valid witness such that $\langle \mathbf{a}, \mathbf{b} \rangle = y$. Note that we can trivially generate $\mathbf{A} = (P^{a_i})_{i\in[q]}$.

Step 3: If an adversary could create a proof for an incorrect $(\mathbf{b}, \mathbf{a})$, then the polynomial representation would lead to a contradiction with the *q-DL* assumption, similar to Case 2 in Theorem 1. □

Proof of Extended Knowledge-Soundness. In the extended knowledge soundness model, an adversary is allowed to make oracle queries *before* seeing the crs. Let the crs includes elements of the form $(P^{\alpha^i})_{i\in[2q]}$, where $\alpha \in \mathbb{Z}_p$ is sampled uniformly at random. Suppose the adversary makes an oracle query on an element $X \in \mathbb{G}_1$ before seeing the crs.

Since the adversary has not yet received any α-dependent values, the query X cannot be a function of α. Therefore, X can be expressed using a basis independent of α, and in particular, its representation is independent of the hidden value α. This implies that evaluating the adversary's query does not increase the degree of any polynomial in α that may later appear in the protocol transcript.

Thus, when the verifier checks a relation involving the crs and the adversary's response, any equation derived from this interaction still involves polynomials in α of bounded degree—unchanged from the original knowledge soundness setting.

The only way the adversary could craft a successful forgery that breaks soundness would be by correctly guessing α, which occurs with probability at most $1/p$, i.e., negligible in the security parameter λ.

Hence, oracle queries made prior to CRS generation do not affect soundness, and the extended knowledge soundness property holds. □

C.3 Security Proof of DT-VRF

For *aggregate binding* and *aggregate pseudorandomness*, the argument remains identical and follows directly from the proofs of Theorems 4.6 and 4.4 in [56]. Note that AggProve is computed honestly by the challenger after the adversary's output, so GSIPA has no impact on these arguments. We note that here we only consider the unweighted case, i.e., the weight vector $\mathbf{w} = \mathbf{1}$. One can adjust the proof to arbitrary weights but beyond the scope of our current focus.

For *subset uniqueness.* We prove that for any fixed subset I with $|T| = \sum_i b_i$ (represented by a binary vector b), if two aggregate proofs $(\tilde{\pi}, \tilde{\pi}')$ and outputs $(\tilde{y}, \tilde{y}')$ both verify correctly under the same input x, such that the aggregation remains a valid VRF, then the outputs must be equal. Intuitively, our PRF function is deterministic with respect to avk and ℓ and since the target group $\mathbb{G}_T$ is cyclic and the base element $g_u = e(S, H(x))$ is fixed, the exponentiation map $z \mapsto g_u^z$ is injective. Therefore, any change in the exponent $z = \sum_i k_i r_i$ results in a different output $\tilde{y} = g_u^z$. Since the exponent depends on the set I (via the subset of indices $\mathbf{b}$ where $b_i = 1$) and the associated challenge vector $\mathbf{r}$, it follows that fixed aggregation sets yield fixed aggregate outputs.

Proof (Subset Uniqueness). Let I be a fixed subset of participating parties, represented by a binary indicator vector $\mathbf{b} \in \{0,1\}^n$, where $b_i = 1$ if $\mathsf{vk}_i \in I$ and $b_i = 0$ otherwise. We extract the set I and the corresponding participating verification keys vk_i for which $b_i = 1$ from the GSIPA proof. The

GSIPA proof proves that the aggregate verification key avk is correctly computed as $\mathsf{avk} = \prod(\mathsf{vk}_i^{r_i})^{b_i}$ for the committed vector b and the revealed keys and threshold are correctly formed and bound to a unique set I.

Each party $\mathsf{vk}_i \in I$ is represented by a public key $\mathsf{vk}_i = g^{k_i}$ and a VRF output $y_i = g_u^{k_i}$, where $g_u = e(P, H(x)) \in \mathbb{G}_T$, and $k_i \in \mathbb{Z}_p$ is the secret key. The challenge vector $\mathbf{r} = (r_1, \ldots, r_n) \in \mathbb{Z}_p^n$ is deterministically derived as $\mathbf{r} \leftarrow \tilde{H}(x, I, \{\mathsf{vk}_i\}_{i\in T}, \{y_i\}_{i\in T})$, where $\tilde{H}$ is modeled as a random oracle. The aggregate proof and output are defined as

$$\tilde{\pi} = H(x)^{\sum_{i\in[n]} b_i r_i k_i}, \qquad \tilde{y} = e(P, \tilde{\pi}) = e(P, H(x))^{\sum_{i\in[n]} b_i r_i k_i} = g_u^z,$$

where $z := \sum_{i\in[n]} b_i r_i k_i \in \mathbb{Z}_p$. We now argue that for any fixed set I and corresponding $\mathbf{b}$, the aggregate output $\tilde{y}$ is uniquely determined.

Assume, for contradiction, that there exist two distinct aggregate outputs $\tilde{y}_0 \neq \tilde{y}_1 \in \mathbb{G}_T$ and valid corresponding proofs $\tilde{\pi}_0, \tilde{\pi}_1$ that both verify correctly for the same set I, challenge vector $\mathbf{r}$, and aggregate key avk, under input x. Then both outputs must be of the form

$$\tilde{y}_j = g_u^{z_j}, \quad \text{with } z_j = \sum_{i\in[n]} b_i r_i k_i^{(j)} \text{ for } j = 0, 1.$$

Since $g_u \in \mathbb{G}_T$ is a fixed generator of a prime-order group, the map $z \mapsto g_u^z$ is injective. Hence, $\tilde{y}_0 = \tilde{y}_1$ if and only if $z_0 = z_1$. If $\tilde{y}_0 \neq \tilde{y}_1$, then necessarily $z_0 \neq z_1$, implying that at least one of the values $k_i^{(j)}$ differs for some $i \in I$.

However, the GSIPA proof binds the committed indicator vector $\mathbf{b}$, the challenge vector $\mathbf{r}$, and the set of participating public keys $\{\mathsf{vk}_i\}_{i\in T}$ to the aggregate key avk. This binding guarantees that the exponent $z = \sum_{i\in[n]} b_i r_i k_i$ is uniquely determined by the verified public keys and input x. Thus, if both proofs verify and use the same set I, then the exponent z, and hence the output $\tilde{y}$, must be identical.

Therefore, the assumption $\tilde{y}_0 \neq \tilde{y}_1$ leads to a contradiction, and we conclude that for any fixed set I and input x, the aggregate output $\tilde{y}$ is uniquely determined, concluding the proof of subset uniqueness.

C.4 Proof of Unforgeability of STS Theorem 4

Proof (Sketch). We reduce the security of our threshold scheme to the AMS unforgeability (cf. Def. 13) of the underlying multi-signature from [12]. Thus, we consider a reduction $\mathcal{B}$ playing the role of the AMS adversary against [12]. At the same time, the reduction simulates a challenger for an internally run an STS adversary $\mathcal{A}$, who tries to break unforgeability of our scheme (cf. Def. 20).

Now let adversary $\mathcal{B}$ receives the upper bound on the number of parties q from $\mathcal{A}$ and forwards it to the real challenger. It gets the public parameters pp back and forwards them to $\mathcal{A}$.

Now $\mathcal{A}$ picks a subset A of parties to corrupt and communicates it to $\mathcal{B}$. At this point, $\mathcal{B}$ makes a guess on the user to attack $i^* \in \{1, \ldots, q\} \setminus A$. For the target signer i^*, the $\mathcal{B}$ gets vk_{i^*} and sends the vk_{i^*} to the oracle $\mathcal{O}$ of the extended knowledge soundness definition and gets aux. $\mathcal{B}$ can gets crs running the GSIPA oracle and forwards them to $\mathcal{A}$ alongside with $(\mathsf{pk}_i, \mathsf{hint}_i)_{i\in[q]\setminus A}$. The adversary outputs $(\mathsf{pk}_i, \mathsf{hint}_i)_{i\in A}$.

$\mathcal{B}$ has all $(\mathsf{pk}_i, \mathsf{hint}_i)_{i\in\{1,\ldots,q\}}$, allowing it to compute $(\mathsf{AK}, \mathsf{vk})$ from $\mathsf{Preprocess}(\{\mathsf{pk}_1, \mathsf{hint}_1\}_1, \ldots, \{\mathsf{pk}_q, \mathsf{hint}_q\}_q)$ and sends it to $\mathcal{A}$. The remaining components of the aggregate verification key and proof can be computed from the GSIPA crs after it is revealed.

Whenever $\mathcal{B}$ receives a signing query k from $\mathcal{A}$, if $k \in \{1, \ldots, q\} \setminus \{A \cup \{i^*\}\}$, it answers honestly. Otherwise, if $k = i^*$, it forwards the signing query to the AMS challenger. Upon reception of the query response, $\mathcal{B}$ forwards it back to $\mathcal{A}$. This process runs as many times as desired by $\mathcal{A}$.

Eventually, $\mathcal{B}$ receives a message msg^*, signature σ^* from $\mathcal{A}$. From σ^*, $\mathcal{B}$ can extract the subset of signers J^* due to the extended knowledge soundness of GSIPA and checks if $i^* \in J^*$. If so, $\mathcal{B}$ wins whenever $\mathcal{A}$ wins. Otherwise, $\mathcal{B}$ aborts. This reduction incurs a polynomial loss in the number of parties.