Practical Subvector Commitments with Optimal Opening Complexity

Matteo Campanelli

Offchain Labs & University of Tartu

Abstract. We introduce a simple pairing-based vector commitment with subvector opening where, after a one-time preprocessing, the prover can open a subvector of size ℓ in *linear* time. Our focus is on practically relevant solutions compatible with already deployed setups—specifically, the powers-of-τ setup used by KZG and many popular SNARKs. We achieve substantial concrete speedups over aSVC (Tomescu et al., SCN 2020), the relevant state of the art in deployable subvector commitments with $O(\ell \log^2 \ell)$ prover and verifier time: our opening is over $60\times$ faster on subvectors of any size; on large subvectors our opening and verification achieve $\approx 4000\times$ and $170\times$ speedups respectively (and four times as much with parallelism).

Our main result is a construction secure in the AGM and in which:

- A commitment is a single $\mathbb{G}_2$ element and a proof is a single $\mathbb{G}_1$ element;
- Opening requires ℓ point additions in $\mathbb{G}_1$;
- Verification is dominated by 2ℓ $\mathbb{G}_1$ operations.

We also describe two variants of our main design that are directly compatible with deployed applications and where the commitment is a $\mathbb{G}_1$ element; these two schemes show similar speedups over prior work. We additionally support cross-commitment and distributed aggregation, and provide an open-source implementation.

1 Introduction

Vector commitment schemes [LY10, CF13] are a cryptographic primitive that, in contrast to simple hashing, enable a form of fine-grained integrity: a weak client holding a short digest $\mathsf{cm}_{\boldsymbol{v}}$ to a vector $\boldsymbol{v}$, can ensure that position i in $\boldsymbol{v}$ has value y by checking a small proof π. What distinguishes this primitive from other commitment schemes is that both the commitment and the opening are of size independent of m, the length of the vector $\boldsymbol{v}$[1]. Vector commitments—and their extensions with subvector-opening described below—have numerous applications, including "stateless" blockchains[2] [TAB+20], verifiable databases [BGV11], de-

[1] This definition is sometimes extended to include constructions with a mild dependency on m, e.g., $\log m$. In this work we focus on *highly succinct* schemes—i.e., with commitments and proofs of constant-size.

[2] This term usually refers to a chain design where validation is stateless and can be performed by just accessing a short digest of the current state.

centralized storage networks [CFG+20], revocation in group signatures [LPY12], and proofs of replication [Fis19, ABC+24].

Subvector commitments. In some settings (including all of the example applications above) we are interested in opening a committed vector in multiple positions at the same time. That is, the client can request a set of positions $I \subset \{1, \ldots, m\}$ and receive a response $\boldsymbol{y}$ together with a proof π_I guaranteeing that $\boldsymbol{y} = (v_i)_{i \in I}$. Here we strengthen our requirements on the size of the proof π: it should not be only independent of $|\boldsymbol{v}|$, but also independent of ℓ, the size of the requested subvector $\boldsymbol{y}$. We refer to this type of opening as *subvector opening* and to the corresponding primitive as a *subvector commitment* (or SVC).

Fast opening through preprocessing. Because in a vector commitment there exists only a limited number of possible individual openings (as many as the positions in the committed vector $\boldsymbol{v}$), it is possible in principle to *preprocess* the openings π_i for all the possible queries on single positions v_i. With the help of the right algebraic setting and cryptographic assumptions, we can sometimes exploit this fact to design an efficient SVC that works as follows. When the client requests a set of positions $I = \{i_1, \ldots, i_\ell\}$, the prover:

(i) retrieves from memory the relevant individual proofs $\pi_{i_1}, \ldots, \pi_{i_\ell}$;

(ii) uses a special *aggregation* procedure to combine them all into a single and constant-sized proof π^*, which it then sends to the client.

Through this approach, one can build SVCs where the opening time is independent of the vector size m, but instead grows only as a function of ℓ.[3] For this blueprint to be genuinely effective in practice, the resulting SVC should also satisfy two extra properties: aggregation should be more efficient than recomputing the proofs from scratch (else it would defy the point of the approach); both commitments and proofs should be *updatable*, that is we should be able to modify them efficiently whenever the vector $\boldsymbol{v}$ changes.

Limitations of the state of the art. Prior work has shown that the aggregation-based approach is possible from different types of cryptographic assumptions, such as elliptic curves endowed with bilinear pairings [GRWZ20, TAB+20] and groups of unknown-order [CFG+20, BBF19]. Among these two families of designs, pairing-based ones are generally the most attractive from a practical point of view: compared to their counterparts based on unknown-order groups, they tend to feature more compact proofs, smaller constants in running times, homomorphic commitments and/or rely on trusted setups that are already widely deployed (see also discussion in Section 1.2)[4] Yet despite these advantages, even the most performant among deployable pairing-based SVCs—the aggregatable SVC (aSVC) of Tomescu et al. [TAB+20]—requires $O(\ell \log^2 \ell)$ time for both aggregation and verification (due to their reliance on polynomial division techniques). Designing a *deployable* SVC with linear-time openings and verification has remained an open problem (where by "deployable" we mean an SVC with the practically relevant characteristics outlined above).

[3] Notice that ℓ, the size of the requested subvector, can be much smaller than m.

[4] Pairing-based schemes have also proven practical through deployments at massive scale. E.g., KZG vector opening is at the heart of Ethereum's sharding [BFL+22].

1.1 This work: optimal opening time and faster verification from existing setups

Our main contribution is constructing a pairing-based SVC where, after a one-time preprocessing stage, opening runs in *linear* time (the theoretical optimal). Verification in our scheme is dominated by $\approx 2\ell$ group operations (specifically, ℓ $\mathbb{G}_1$ operations plus one multi-scalar multiplication of size ℓ), improving on the $O(\ell \log^2 \ell)$ field operations and size-ℓ $\mathbb{G}_2$ MSM required by Tomescu et al. [TAB+20] (see summary in Table 1). Our improvements are not only asymptotic, but translate to substantial concrete improvements. As we show in Section 7, our main scheme obtains the following speedups:

- *Opening:* 60× on subvectors of *any* size; 4000× on large subvectors ($\ell \approx$64K).
- *Verification:* 170× on large subvectors (over 750× when using parallelism).

In our main construction, a commitment is a single $\mathbb{G}_2$ element, in contrast to the scheme in Tomescu et al. where it lies in $\mathbb{G}_1$. We remove this limitation in two variants of our main scheme without sacrificing efficiency. Notably, both these schemes also show orders of magnitude speedups (see Section 7.2).

All our constructions support: homomorphism, efficient updatability of commitments and proofs, and aggregation of proofs across multiple commitments (cross-commitment aggregation). Being *associative* (same-commitment aggregation is a simple sum of proofs), our proofs are particularly suitable for settings where proof generation is distributed among nodes. We also show that all m individual proofs can be preprocessed in $O(m \log m)$ time. Although we rely on a different verification equation, we establish a connection between the proofs in our schemes and those in KZG commitments. This allows us to leverage techniques for preprocessing KZG openings, specifically Feist-Khovratovich [FK23]. We prove the security of our schemes in the Algebraic Group Model (AGM).

Additional contributions of this work include:

- The two aforementioned variants of our main construction—$\Pi^{\mathsf{fastP}}_{\mathbb{G}_1}$ (optimized for proving time) and $\Pi^{\mathsf{fastV}}_{\mathbb{G}_1}$ (optimized for verification time). These constructions feature a commitment cm consisting of a single $\mathbb{G}_1$ element. If cm is a commitment to vector $\boldsymbol{v}$, it corresponds to a KZG commitment [KZG10] to the (standard) Lagrange-basis encoding of $\boldsymbol{v}$. Thus our variants are fully compatible with already deployed trusted setups and other KZG-based applications, such as popular SNARKs [GWC19] and Ethereum's data availability sampling solutions [BFL+22].
- An open source Rust implementation built on arkworks [ark22], with a comprehensive experimental evaluation comparing our schemes against aSVC.

Application highlight: faster proving in verifiable databases. Our constructions show immediate improvements to recent works adopting vector commitments in the bilinear settings for verifiable databases (or VDBs). At its essence, a VDB is a specialized proof system for database queries where the client holds only a short digest to the whole database. Recently, Botta et al. [BBC+25] proposed qedb, a VDB that supports a representative subset of SQL. At its heart, it combines accumulators and subvector commitments with specific fea-

Scheme	$\lvert\mathbf{cm}\rvert$	$\lvert\boldsymbol{\pi}\rvert$	Preproc.	Open	Verify
aSVC [TAB+20]	$1\,\mathbb{G}_1$	$1\,\mathbb{G}_1$	$O(m \log m)$	$O(\ell \log^2 \ell)\,\mathbb{F}$, $\mathsf{MSM}_1(\ell)$	$O(\ell \log^2 \ell)\,\mathbb{F}$, $2\,\mathbb{P}, \mathsf{MSM}_2(\ell)$
Main constr.*(Fig. 1)	$1\,\mathbb{G}_2$	$1\,\mathbb{G}_1$	$O(m \log m)$	$\ell\,\mathbb{G}_1$	$\ell\,\mathbb{G}_1, 3\,\mathbb{P}$, $\mathsf{MSM}_1(\ell)$
Variant* $\Pi^{\mathsf{fastP}}_{\mathbb{G}_1}$	$1\,\mathbb{G}_1$	$1\,\mathbb{G}_1$	$O(m \log m)$	$\ell\,\mathbb{G}_1$	$\ell\,\mathbb{G}_2, 3\,\mathbb{P}$, $\mathsf{MSM}_1(\ell)$
Variant* $\Pi^{\mathsf{fastV}}_{\mathbb{G}_1}$	$1\,\mathbb{G}_1$	$1\,\mathbb{G}_1, 1\,\mathbb{G}_2$	$O(m \log m)$	$\ell\,\mathbb{G}_1, \ell\,\mathbb{G}_2$	$\ell\,\mathbb{G}_1, 3\,\mathbb{P}$, $\mathsf{MSM}_1(\ell)$

Table 1: Comparison of practical subvector commitments with sublinear opening. The table reports dominating costs for running times. For costs: $\mathbb{P}$ refers to one pairing; $\mathsf{MSM}_i(\ell)$ (for $i \in \{1, 2\}$) refers to one multi-scalar multiplication of size ℓ in group $\mathbb{G}_i$. A star ($\star$) marks constructions from this work. Variants are described in Section 5. The efficiency profile for our verifier-optimized variant ($\Pi^{\mathsf{fastV}}_{\mathbb{G}_1}$) assumes the optimization of Remark 1. In Section 1.2 we compare to other solutions with superlinear opening time that rely on random oracles and that have bespoke setups or larger proof sizes.

tures (including subvectors instantiated with aSVC by Tomescu et al. [TAB+20]) in order to obtain an expressive VDB that is provably secure and features high modularity and performance without relying on general-purpose proof systems.

Our construction $\Pi^{\mathsf{fastP}}_{\mathbb{G}_1}$ can be used as a drop-in replacement for the SVC in [BBC+25]. We now highlight how it can provide substantial speedups on a relevant subset of their supported queries and leave as future work a full-fledged analysis of the impact of this change. We consider queries of the following form (where C1 and C2 are columns in the table T and v is an arbitrary literal):

Q: SELECT C1 FROM T WHERE C2 = v

For simplicity here we focus on the implications for the cost of *proving* in this type of query; however, we remark that proving other queries and verification will also benefit from the speedups of $\Pi^{\mathsf{fastP}}_{\mathbb{G}_1}$ against aSVC (we do not discuss them here as they would be substantially more complex to analyze quantitatively). In qedb, the cost of proving a query like Q consists in simply providing an aSVC opening for a vector commitment encoding C (for this specific type of equality checks, the prover also provides an additional precomputed value—a commitment to an "inverted index"—to show the filtering was done on the right rows; this does not require any computation).

Examples of our improvements: Consider a table T with $N \approx$130K rows and where the result of Q is (three scenarios): *a)* 1% of the table T; *b)* 10% of T; *c)* 25% of T. Then, by using our scheme $\Pi^{\mathsf{fastP}}_{\mathbb{G}_1}$ as a replacement for the SVC in qedb, we can achieve 2–3 orders of magnitude speedups. Specifically, we obtain these ballpark improvements for the proving time for query Q[5]:

a) 0.2s → 1ms; *b)* 5s → 8ms; *c)* half a minute → 16ms.

[5] These timings are derived from the relevant benchmarks in Section 7.

Scenario highlight: broad applicability within Ethereum's ecosystem. Our scheme requires only a standard powers-of-τ setup—which also underlies Ethereum's blob commitments [BFL+22] and PLONK-based rollups—showing potential applicability in this setting. Ethereum uses SVC-like primitives in data availability sampling (DAS) and in the past has considered switching to Verkle Trees for stateless validation via compact state proofs [BFB+24]. The current DAS implementation opens over cosets [WZ24], partly for erasure coding compatibility; our arbitrary-position openings could benefit designs needing flexible access. For stateless validation, our distributed aggregation could let validators with partial witnesses combine proofs incrementally and without centralized coordination.

1.2 Related Work

The literature on vector commitments is vast and we refer the reader to two surveys on the topic, [Nit21] and [PRvdM25]. The notion of *subvector* commitment was introduced in [LM19].

Subvector commitments from bilinear pairings. Tomescu et al. [TAB+20] proposed an aggregatable vector commitment (dubbed aSVC) with a setup compatible with the KZG commitment [KZG10] and other popular SNARK designs [CHM+20, CFF+21, GWC19]. Schemes satisfying this setup are substantially easier to deploy in practice because: they are reusable by different cryptographic constructions, they provide higher trust (they can be "updated" by different parties and their security holds as long as a *single* party performs the update honestly [GKM+18]), and they have already been generated and used at large scale [axi25, ale21, eth23]. Here we point out two differences between aSVC and our work, and refer the reader to the rest of the text for a comparison along different metrics (see specifically: Section 2 for a comparison of the techniques; Table 1 and Section 7 for an efficiency comparison). In contrast to our work, the aggregation in [TAB+20] cannot be performed incrementally or in a distributed manner but is *one-hop* only (in the sense of [CNR+22]); also, aSVC cannot perform aggregation *across commitments* (see comparison with Pointproofs below). A dimension in which aSVC offers a better tradeoff compared to our solution is the requirement for the client to store only a number of items of the setup linear in the size of the opened subvector; our schemes, in contrast, require the client to keep a $O(m)$-sized portion of the parameters. We remark that this limitation can often be inconsequential for the verifier for at least two reasons: this storage overhead is mild (for $m \approx 10^5$ it requires storing $\approx$ 5MB for our main scheme and $\Pi_{\mathbb{G}_1}^{\mathsf{fastV}}$ and $\approx$ 10 MB for $\Pi_{\mathbb{G}_1}^{\mathsf{fastP}}$); in settings where we prove verification inside a SNARK[6] [BCC+17]the client can simply keep a succinct commitment to the setup parameters and obtain the guarantee that the correct subset of parameters were used as part of the SNARK proof.

[6] This practice is not uncommon and can be useful if the SVC is used in the context of proving a more complex computation or to make the verifier extremely efficient.

Pointproofs [GRWZ20, LPR22] are a construction for SVC obtained as a variation of [LY10] and with constant-sized proof and commitments (all the observations in these paragraphs also generally apply to schemes based on [LY10]). Pointproofs satisfy a property called *cross-commitment aggregation* (see Section 4.2), which our constructions also satisfy (but that is not satisfied by aSVC). The parameters for Pointproofs require a larger and idiosyncratic setup that is not compatible with those already deployed or with those used by other SNARKs[7]—this reduces the opportunity for simple deployment of this scheme. Their single-commitment aggregation requires a multi-scalar multiplication with challenges from a random oracle, making it not strictly linear and concretely less efficient than ours[8]. Also, in contrast to Pointproofs, our aggregation is associative making it suitable for distributed settings [CFG+20].

The work in [CNR+22] proposes a construction of linear-map vector commitments (LVC), i.e. commitments to vectors that support proofs of evaluations for linear maps. The equation we obtain in our Lemma 1 is loosely related to the techniques used by the Lagrange-basis scheme in [CNR+22]. This scheme does support some forms of aggregation with preprocessing [CNR+22, §8.2] for *arbitrary* opening subsets—which, importantly, may apply to subvector openings—but these present a few caveats. Both types of aggregation in [CNR+22] have superlinear complexity—one is based on [TAB+20] (whose limitations we discuss throughout this paper), while the other is based on an MSM with random-oracle challenges (discussed in Footnote 8). Additionally, the proofs in [CNR+22] are larger and consist of three group elements. Overcoming these limitations is part of our contributions: we find techniques adjacent to those in [CNR+22] that lead to subvector proofs with a *single* group element (in our main construction and in the variant $\Pi_{\mathbb{G}_1}^{\mathsf{fastP}}$) and provide an efficient and simple preprocessing for our schemes—one that can also reuse existing implementations—by observing a relation between the structure of our proofs and those in KZG-style schemes, despite them relying on different verification equations.

We finally cite other prominent works based on bilinear pairings: [LJGK25], which improves on batch updates in vector commitments based on KZG; [SCP+22], which proposes a tree-based construction on top of the PST commitment [PST13] (with proofs of logarithmic size); [LM19], which originally introduced the notion of subvector opening and proposed a pairing-based scheme with constant-sized proofs (not supporting aggregation and requiring quadratic opening time).

[7] This setup consists of a special powers-of-τ sequence up to degree $2m$ (instead of m as in standard KZG), with a "hole"—the element $[\tau^{m+1}]_1$ must be absent (i.e., no party should know $[\tau^{m+1}]_1$).

[8] Aggregation based on a random oracle $\mathcal{H}$ requires an MSM of size ℓ with high-entropy scalars derived from $\mathcal{H}$. An MSM of this type involves $\omega(\ell)$ (superlinear) cost because Pippenger's algorithm requires these many group operations for such an MSM (see discussion on page 3 in [GLS+21]). In contrast, our scheme requires ℓ plain group additions. This difference results in roughly 20–50× speedups in our favor—these can be inferred directly from our benchmarks by comparing aSVC commitment times (an MSM of comparable structure) against our aggregation times (see Tables 2 and 3 in the full version of this paper [Cam26]).

Other works on subvector commitments. A number of works use the properties of hidden order groups, pairing-free elliptic curves and lattices to design schemes with attractive features, some of which support aggregation. In general, these schemes exhibit worse asymptotics than pairing-based constructions and/or are concretely less efficient (due to larger constants). A partial list of such works includes [BBF19, AR20, CFG+20, CEO22, CHAK23, WW23, TB23].

2 Technical Overview

Let $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T)$ be a bilinear group of prime order p, along with a pairing $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ (see also Section 3.1 for reference on basic notation). A common approach for constructing vector commitments from pairings is to encode a vector $\mathbf{a} = (a_1, \ldots, a_m)$ as the polynomial $f_{\mathbf{a}}(X) = \sum_i a_i \lambda_i(X)$, where $\lambda_i(X)$ are the Lagrange basis polynomials over a multiplicative subgroup $\mathbb{H} = \{h_1, \ldots, h_m\}$ of roots of unity (used for FFT). The commitment is then simply a polynomial commitment to $f_{\mathbf{a}}(X)$, specifically a KZG polynomial commitment. The KZG scheme works as follows. At setup time a secret value τ is sampled (and then discarded—it should be known to no one after the setup phase); the parameters for KZG for polynomials up to degree m contain powers of the secret τ in the groups $\mathbb{G}_1$ and $\mathbb{G}_2$, i.e., $([\tau_i]_1, [\tau_i]_2)_{i=1}^m$. A commitment to $f_{\mathbf{a}}(X)$ is simply $[f_{\mathbf{a}}(\tau)]_1$ ($[f_{\mathbf{a}}(\tau)]_2$ for KZG commitments in $\mathbb{G}_2$) which can be computed as $\sum_{j=0}^{m-1} c_j [\tau^j]_1$ where the values c_j are the coefficients of the polynomial. In order to open a single position i with value a_i, a KZG evaluation proof π_i consists of a commitment to the quotient polynomial $q_i(X)$. The latter is the solution to the following polynomial equation, which is checked by the verifier through pairings[9]:

$$(f_{\mathbf{a}}(X) - a_i) = q_i(X)(X - h_i)$$

The challenge that aSVC [TAB+20] addresses is how to *aggregate* multiple individual proofs $\pi_i = [q_i(\tau)]_1$ into a single constant-sized subvector proof *efficiently*. Their approach leverages partial fraction decomposition: they observe that the quotient polynomial for a batch opening can be written as a linear combination $q(X) = \sum_{i \in I} c_i \cdot q_i(X)$, where the coefficients c_i depend on the derivative A'_I of the accumulator polynomial $A_I(X) = \prod_{j \in I}(X - h_j)$. Computing these coefficients requires evaluating A'_I at all points in I, which takes $O(\ell \log^2 \ell)$ field operations using subproduct trees and DFT-based techniques. The aggregated proof is then $\pi_I = \sum_{i \in I} c_i \cdot \pi_i$.

If one wants to achieve *linear-time* aggregation, a natural approach would be to find a scheme where precomputed proofs could simply be summed, i.e., computing $\pi^* = \sum_{i \in I} \pi_i$. However, this is not possible with standard KZG proofs—the quotient polynomials $q_i(X)$ have different denominators $(X - h_i)$,

[9] For concreteness, the pairing analogue of this equation is:

$$e\left([f_{\mathbf{a}}(\tau)]_1 - [a_i]_1, [1]_2\right) = e\left([q_i(\tau)]_1, [\tau]_2 - [h_i]_2\right).$$

so their commitments are not directly additive in a useful way. This is precisely why aSVC must develop its more complex aggregation machinery. In this work, in order to obtain *linear-time* aggregation, we need to find an altogether different verification equation.

We derive such an equation by working with the product $p(X) := f_{\mathbf{a}}(X)\lambda_j(X)$ rather than the difference $f_{\mathbf{a}}(X) - a_j$. We then consider the division of this polynomial p by $t(X)$, the vanishing polynomial of $\mathbb{H}$. The key insight is that this division yields a verification equation with a particularly nice structure. The remainder of $p(X)/t(X)$ is $a_j\lambda_j(X)$ and, denoting by $q'_j(X)$ the resulting quotient, we: *i)* use as opening proof for position j simply $\pi_j = [q'_j(\tau)]_1$; *(ii)* let the verifier check the following polynomial equation (via pairings):

$$f_{\mathbf{a}}(X)\lambda_j(X) = q'_j(X) \cdot t(X) + a_j\lambda_j(X)$$

To see that this approach enables linear-time aggregation, consider what happens for two positions i and j. We have that:

$$f_{\mathbf{a}}(X)\underbrace{(\lambda_i(X) + \lambda_j(X))}_{\Lambda_{\mathsf{LHS}}} \equiv \underbrace{a_i\lambda_i(X) + a_j\lambda_j(X)}_{\Lambda_{\mathsf{RHS}}} \pmod{t(X)}$$

meaning that the sum of quotients $q'_i(X) + q'_j(X)$ is exactly the quotient for the combined opening. Consequently, the aggregated proof is simply $\pi^* = \pi_i + \pi_j$, a single group addition. This extends immediately to any subset I.

In order to reduce the computational burden for the verifier—who needs to compute Λ_{LHS} and Λ_{RHS} above—a natural choice is to encode these two terms in $\mathbb{G}_1$ (where group operations are cheaper). A simple corollary of this choice, by inspection of the equation above, is that the commitment to the vector (which encodes $f_{\boldsymbol{a}}(X)$) will lie in $\mathbb{G}_2$. We will later describe two variants of our scheme where the commitment lies in the group $\mathbb{G}_1$ offering different efficiency tradeoffs.

We observe several properties satisfied by our design. Besides the obvious homomorphic features of its commitments and the associativity of its proofs, we also show that it enables aggregation of proofs *across different commitments*[10]. Finally, by an observation connecting the structure of our proofs to that of KZG, we leverage known techniques for efficient preprocessing and updates.

Outline of the remainder of the paper. We provide general background in Section 3. Our main scheme and its core correctness and security properties are presented in Section 4; in that same section we discuss cross-commitment aggregation. Variants of our scheme with commitment in $\mathbb{G}_1$ are in Section 5. The observations required to show an efficient preprocessing and updatability are discussed in Section 6. We describe our experimental evaluation in Section 7.

[10] Through which, a client verifying k subvectors for as many commitments and index sets *at the same time*, can simply receive a single proof instead of k separate ones.

3 Preliminaries

3.1 General background and notation

Bilinear groups. A bilinear group is described by a tuple $\mathsf{gk} = (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e)$ such that $\mathbb{G}_1, \mathbb{G}_2$ are cyclic (additive) groups of prime order p. We use the notation $[a]_1, [b]_2, [c]_t$ for elements in $\mathbb{G}_1, \mathbb{G}_2$ and $\mathbb{G}_T$ respectively and use the convention that $[1]_i$ denotes the generator for $\mathbb{G}_i$ and $[v] := v[1]_i$ for $v \in \mathbb{F}, i \in \{1, 2\}$. $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ is a bilinear asymmetric map (pairing), which means that $\forall a, b \in \mathbb{F}$, $e([a]_1, [b]_2) := [ab]_t$. We implicitly have that $[1]_t := e([1]_1, [1]_2)$ generates $\mathbb{G}_T$. We use $[a]_{1,2}$ to refer to the two group elements $[a]_1 \in \mathbb{G}_1, [a]_2 \in \mathbb{G}_2$. We denote by gk the group description $\mathsf{gk} = (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e)$ and by BGen the algorithm generating it on input a security parameter.

Algebraic Group Model. The algebraic group model (AGM) [FKL18] is a security framework where we consider only so-called algebraic adversaries. Such adversaries have direct access to group elements and, in particular, can use their bit representation, like in the standard model. However, these adversaries are assumed to be able to "explain" new group elements, showing how to apply group operations to received group elements. This requirement is formalized as follows: Suppose an adversary $\mathcal{A}$ is given some group elements $[x_1]_1, \ldots, [x_n]_1 \in \mathbb{G}_1$. Then, for every new group element $[z]_1 \in \mathbb{G}_1$ that the adversary outputs, it must also output $z_1, \ldots, z_n \in \mathbb{F}$ such that $[z]_1 = \sum_{i=1}^{n} z_i [x_i]_1$. The analogous requirement holds for elements in $\mathbb{G}_2$.

Cryptographic assumptions. We will use the following assumptions.

Definition 1 (m-DLOG Assumption). *The m-Discrete Logarithm assumption holds with respect to a bilinear group generator* BGen *if for all* PPT *adversaries $\mathcal{A}$, the following probability is negligible in λ:*

$$\Pr\left[\tau \leftarrow \mathcal{A}\left(\mathsf{gk}, \left\{[\tau^i]_1\right\}_{i=0}^{m}, \left\{[\tau^i]_2\right\}_{i=0}^{m}\right) \;\middle|\; \mathsf{gk} \leftarrow \mathsf{BGen}(1^\lambda);\ \tau \leftarrow\!\$\, \mathbb{F}\right]$$

Definition 2 (SDH Assumption [BB04]). *The strong Diffie-Hellman assumption (SDH) holds with respect to a bilinear group generator* BGen *if for all* PPT *adversaries $\mathcal{A}$ and degree bound $m > 0$, we have that the following is negligible for $b \in \{1, 2\}$:*

$$\Pr\left[T = \left[\frac{1}{\tau + c}\right]_b \mid \mathsf{gk} \leftarrow \mathsf{BGen}(1^\lambda); \tau \leftarrow\!\$\, \mathbb{F}; S := \left\{\left[\tau^i\right]_{1,2}\right\}_{i=0}^{m}; (T, c) \leftarrow \mathcal{A}(\mathsf{gk}, S)\right]$$

Algebraic preliminaries. Let $\omega \in \mathbb{F}$ be a primitive m-th root of unity, so $\omega^m = 1$ and $\omega^k \neq 1$ for $0 < k < m$. Define the set of all m-th roots of unity:

$$\mathbb{H} = \{1, \omega, \omega^2, \ldots, \omega^{m-1}\}$$

We write $h_i = \omega^{i-1}$ for $i \in [m]$, so $\mathbb{H} = \{h_1, h_2, \ldots, h_m\}$. The vanishing polynomial of $\mathbb{H}$ is: $t(X) = \prod_{i=1}^{m}(X - h_i) = X^m - 1$. The Lagrange polynomial $\lambda_i(X)$ is the unique polynomial of degree $m-1$ satisfying:

$$\lambda_i(h_j) = \delta_{ij} = \begin{cases} 1 & \text{if } i = j \\ 0 & \text{if } i \neq j \end{cases}$$

Two ways of expressing the standard Lagrange interpolation formula are:

$$\lambda_i(X) = \frac{t(X)}{t'(h_i)(X - h_i)} = \frac{h_i \cdot t(X)}{m(X - h_i)} = \frac{h_i(X^m - 1)}{m(X - h_i)} \qquad (\star)$$

$$\lambda_i(X) = \frac{1}{m}\sum_{k=0}^{m-1} \omega^{-(i-1)k} X^k.$$

It is easy to see that $\lambda_i(0) = \frac{1}{m}$ for all $i \in [m]$. We will also use two simple properties of Lagrange polynomials—that for any $i, j \in [m], i \neq j$:

$$\lambda_i(X)\lambda_j(X) \equiv 0 \pmod{t(X)} \qquad \lambda_i(X)^2 \equiv \lambda_i(X) \pmod{t(X)}.$$

DFT. The Lagrange and monomial bases are related by the Discrete Fourier Transform (DFT). If $A(X) = \sum_{j=1}^{m} a_j \lambda_j(X) = \sum_{k=0}^{m-1} \hat{a}_k X^k$, then the coefficient vectors are related by $\mathbf{a} = F \cdot \hat{\mathbf{a}}$ and $\hat{\mathbf{a}} = \frac{1}{m}\bar{F} \cdot \mathbf{a}$, where F is the DFT matrix with entries $F_{i,k} = \omega^{(i-1)k}$ and $\bar{F}$ has entries $\bar{F}_{k,i} = \omega^{-(i-1)k}$.

3.2 Vector commitments (VC)

Definition 3 (Vector Commitment). *A vector commitment scheme for vectors of length m consists of the following algorithms:*

- $\mathsf{VC.Setup}(1^\lambda, 1^m) \to \mathsf{pp}$*: On input security parameter λ and vector length m, outputs public parameters* pp*.*
- $\mathsf{VC.Commit}(\mathsf{pp}, \mathbf{a}) \to (\mathsf{cm}, \mathsf{aux})$*: On input parameters* pp *and a vector* $\mathbf{a} = (a_1, \ldots, a_m) \in \mathbb{F}^m$*, outputs a commitment* cm *and auxiliary information* aux*.*
- $\mathsf{VC.OpenSingle}(\mathsf{pp}, \mathsf{aux}, i, a_i) \to \pi$*: On input public parameters* pp*, auxiliary information* aux*, position $i \in [m]$, and value a_i, outputs a proof π.*
- $\mathsf{VC.VfySingle}(\mathsf{pp}, \mathsf{cm}, i, v, \pi) \to \{0, 1\}$*: On input parameters* pp*, commitment* cm*, position $i \in [m]$, value v, and proof π, outputs* 1 *(accept) or* 0 *(reject).*

Definition 4 (VC Correctness). *A vector commitment is correct if for any $\lambda, m \in \mathbb{N}$, any vector $\mathbf{a} \in \mathbb{F}^m$, and any $i \in [m]$:*

$$\Pr\left[\mathsf{VC.VfySingle}(\mathsf{pp}, \mathsf{cm}, i, a_i, \pi) = 1 \;\middle|\; \begin{array}{l} \mathsf{pp} \leftarrow \mathsf{VC.Setup}(1^\lambda, 1^m) \\ (\mathsf{cm}, \mathsf{aux}) \leftarrow \mathsf{VC.Commit}(\mathsf{pp}, \mathbf{a}) \\ \pi \leftarrow \mathsf{VC.OpenSingle}(\mathsf{pp}, \mathsf{aux}, i, a_i) \end{array}\right] = 1$$

Definition 5 (VC Position Binding). *A vector commitment is position binding if for any PPT adversary $\mathcal{A}$ and any $\lambda, m \in \mathbb{N}$:*

$$\Pr\left[\begin{array}{c} \mathsf{VfySingle}(\mathsf{pp}, \mathsf{cm}, i, v, \pi) = 1 \wedge \\ \mathsf{VfySingle}(\mathsf{pp}, \mathsf{cm}, i, v', \pi') = 1 \wedge \\ v \neq v' \end{array} \;\middle|\; \begin{array}{l} \mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda, 1^m) \\ (\mathsf{cm}, i, v, \pi, v', \pi') \leftarrow \mathcal{A}(\mathsf{pp}) \end{array}\right] \leq \mathsf{negl}(\lambda)$$

3.3 Subvector commitments (SVC)

Definition 6 (Subvector Commitment). *A subvector commitment scheme extends a vector commitment with the ability to open multiple positions simultaneously. It consists of the following algorithms:*

- $\mathsf{SVC.Setup}(1^\lambda, 1^m) \to \mathsf{pp}$*: On input security parameter* λ *and vector length* m*, outputs public parameters* pp*.*
- $\mathsf{SVC.Commit}(\mathsf{pp}, \mathbf{a}) \to (\mathsf{cm}, \mathsf{aux})$*: On input parameters* pp *and a vector* $\mathbf{a} = (a_1, \ldots, a_m) \in \mathbb{F}^m$*, outputs a commitment* cm *and auxiliary information* aux*.*
- $\mathsf{SVC.Precompute}(\mathsf{pp}, \mathsf{aux}) \to \mathsf{aux}'$*: On input public parameters* pp *and auxiliary information* aux*, outputs extended auxiliary information* aux' *that enables efficient subvector opening.*
- $\mathsf{SVC.OpenMultiple}(\mathsf{pp}, \mathsf{aux}', I, (a_i)_{i\in I}) \to \pi$*: On input public parameters* pp*, extended auxiliary information* aux'*, a set of positions* $I \subseteq [m]$*, and values* $(a_i)_{i\in I}$*, outputs a proof* π*.*
- $\mathsf{SVC.VfyMultiple}(\mathsf{pp}, \mathsf{cm}, I, (v_i)_{i\in I}, \pi) \to \{0, 1\}$*: On input public parameters* pp*, commitment* cm*, a set of positions* $I \subseteq [m]$*, values* $(v_i)_{i\in I}$*, and proof* π*, outputs* 1 *(accept) or* 0 *(reject).*

Definition 7 (SVC Correctness). *A subvector commitment is correct if for any* $\lambda, m \in \mathbb{N}$*, any vector* $\mathbf{a} \in \mathbb{F}^m$*, and any* $I \subseteq [m]$*:*

$$\Pr\left[\begin{array}{l|l} \mathsf{VfyMultiple}(\mathsf{pp}, \mathsf{cm}, I, & \mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda, 1^m) \\ \qquad (a_i)_{i\in I}, \pi) = 1 & (\mathsf{cm}, \mathsf{aux}) \leftarrow \mathsf{Commit}(\mathsf{pp}, \mathbf{a}) \\ & \mathsf{aux}' \leftarrow \mathsf{Precompute}(\mathsf{pp}, \mathsf{aux}) \\ & \pi \leftarrow \mathsf{OpenMultiple}(\mathsf{pp}, \mathsf{aux}', I, (a_i)_{i\in I}) \end{array}\right] = 1$$

Definition 8 (SVC Position Binding). *A subvector commitment is position binding if for any PPT adversary* $\mathcal{A}$ *and any* $\lambda, m \in \mathbb{N}$ *the following is negligible:*

$$\Pr\left[\begin{array}{c|l} \mathsf{VfyMultiple}(\mathsf{pp}, \mathsf{cm}, I, (v_i)_{i\in I}, \pi) = 1 \wedge & \mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda, 1^m) \\ \mathsf{VfyMultiple}(\mathsf{pp}, \mathsf{cm}, I', (v'_i)_{i\in I'}, \pi') = 1 \wedge & (\mathsf{cm}, I, (v_i)_{i\in I}, \pi, \\ \exists i^* \in I \cap I' : v_{i^*} \neq v'_{i^*} & \qquad I', (v'_i)_{i\in I'}, \pi') \leftarrow \mathcal{A}(\mathsf{pp}) \end{array}\right]$$

4 Our Construction

Our construction is presented in Fig. 1. In that figure we proceed by describing our construction "incrementally", providing first a vector commitment with single opening and then augmenting it as an SVC with preprocessing.

4.1 Analysis of the construction

The properties of our scheme will rely on the following fact:

Lemma 1. *Let* $p \in \mathbb{F}[X]$ *with* $p(X) = \sum_{i\in[m]} a_i\lambda_i(X)$ *and let* $\mathbb{H} \subset \mathbb{F}$ *a multiplicative subgroup of* $\mathbb{F}$ *of order* m*. For any* $j \in [m]$

$$p(X)\lambda_j(X) = q'_j(X)t(X) + a_j\lambda_j(X)$$

where $q'_j(X) = \frac{(p(X)-a_j)\lambda_j(X)}{t(X)} = \frac{h_j}{m(X-h_j)}(p(X) - a_j)$*.*

$\mathsf{Setup}(1^\lambda, 1^m)$:

1. Generate bilinear group $\mathsf{gk} = (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e)$.
2. Sample $\tau \leftarrow \mathbb{F}$
3. Compute $[\lambda_i(\tau)]_1$, $[\lambda_i(\tau)]_2$ for $i \in [m]$ (see Section 6). Compute $[t(\tau)]_2$.
4. Output[a] $\mathsf{pp} = \Big(\mathsf{gk}, \big([\tau^i]_{1,2}\big)_{i=1}^m, \big([\lambda_i(\tau)]_{1,2}\big)_{i=1}^m, [t(\tau)]_2\Big)$.

$\mathsf{VC.Commit}(\mathsf{pp}, \boldsymbol{a})$:

1. Compute $\mathsf{cm} = \sum_{j=1}^m a_j [\lambda_j(\tau)]_2$
2. Output $(\mathsf{cm}, \mathsf{aux} = f_a)$ where $f_a(X) := \sum_i a_i \lambda_i(X)$.

$\mathsf{VC.OpenSingle}(\mathsf{pp}, \mathsf{aux}, i, a_i)$:

1. Compute[b] $q_i'(X) := \frac{f_a(X)\lambda_i(X) - a_i\lambda_i(X)}{t(X)}$.
2. Output $\pi_i = \sum_{j\in[m]} q_i'(h_j)[\lambda_j(\tau)]_1$ // i.e., $\pi_i = [q_i'(\tau)]_1$

$\mathsf{VC.VfySingle}(\mathsf{pp}, \mathsf{cm}, i, v, \pi)$: Accept iff this holds:

$$e\Big([\lambda_i(\tau)]_1, \mathsf{cm}\Big) = e\Big(v \cdot [\lambda_i(\tau)]_1, [1]_2\Big)) + e\Big(\pi, [t(\tau)]_2\Big)$$

$\mathsf{SVC.Commit}(\mathsf{pp}, \boldsymbol{a})$: // identical to VC.Commit

1. Compute $\mathsf{cm} = \sum_{j=1}^m a_j [\lambda_j(\tau)]_2$
2. Output $(\mathsf{cm}, \mathsf{aux} = f_a)$ where $f_a(X) := \sum_i a_i \lambda_i(X)$.

$\mathsf{SVC.Precompute}(\mathsf{pp}, \mathsf{aux})$:

1. Compute $\pi_i \leftarrow \mathsf{VC.OpenSingle}(\mathsf{pp}, \mathsf{aux})$ for all i (see also Section 6).
2. Output $\mathsf{aux}' = \mathsf{aux} || (\pi_i)_{i\in[m]}$

$\mathsf{SVC.OpenMultiple}(\mathsf{pp}, \mathsf{aux}, I = (i_1, \ldots, i_\ell), (a_i)_{i\in I})$:

1. Output $\pi^* = \sum_{i\in I} \pi_i$

$\mathsf{SVC.VfyMultiple}(\mathsf{pp}, I = (i_1, \ldots, i_\ell), (v_i)_{i\in I}, \pi^*)$:

1. Let $\Lambda^*_{\mathrm{LHS}} = \sum_{i\in I} [\lambda_i(\tau)]_1$ and $\Lambda^*_{\mathrm{RHS}} = \sum_{i\in I} v_i [\lambda_i(\tau)]_1$.
2. Accept iff this holds:

$$e\Big(\Lambda^*_{\mathrm{LHS}}, \mathsf{cm}\Big) = e\Big(\Lambda^*_{\mathrm{RHS}}, [1]_2\Big) + e\Big(\pi^*, [t(\tau)]_2\Big)$$

[a] Not all the elements in pp are necessary for this construction. Having redundant elements allows us to simply reuse these public parameters for the variants in Section 5. See also discussion in Remark 2.

[b] See also discussion in Section 6 for an efficient way of computing it.

Fig. 1: Our main construction of VC (single opening) and SVC (subvector opening).

Proof. Consider the division of $p(X)\lambda_j(X)$ by $t(X)$, the vanishing polynomial in $\mathbb{H}$. Its remainder is $a_j\lambda_j(X)$. Indeed:

$$p(X)\lambda_j(X) \equiv a_1\lambda_1(X)\lambda_j(X) + \cdots + a_m\lambda_m(X)\lambda_j(X) \equiv a_j\lambda_j(X) \pmod{t(X)}$$

where above we used the simple properties on Lagrange polynomials described in Section 3. Notice that the degree of $a_j\lambda_j(X)$ is strictly less than m (the degree of t). We can then check, that using $q'(X)$ as defined above, we have:

$$\begin{aligned} & p(X)\lambda_j(X) - q'_j(X)t(X) - a_j\lambda_j(X) \\ = \; & p(X)\lambda_j(X) - \frac{h_j t(X)}{m(X-h_j)}(p(X)-a_j) - a_j\lambda_j(X) \\ = \; & p(X)\lambda_j(X) - \lambda_j(X)(p(X)-a_j) - a_j\lambda_j(X) = 0 \end{aligned}$$

where above we used the equivalent description of λ_j in $(\star)$. □

Theorem 1. *The scheme* VC *(resp.* SVC*) in Fig. 1 is correct.*

Proof. This follows immediately by inspection and by applying Lemma 1. □

Theorem 2. *The scheme* VC *in Fig. 1 satisfies position binding for single opening under the SDH Assumption (Definition 2).*

Proof. Consider an adversary that provides commitment cm, two accepting proofs π, π' for respective claims (i, v), (i, v') with $v \neq v'$. We know that:

$$e([\lambda_i(\tau)]_1, \mathsf{cm}) = v \cdot e([\lambda_i(\tau)]_1, [1]_2) + e(\pi, [t(\tau)]_2) \tag{1}$$
$$e([\lambda_i(\tau)]_1, \mathsf{cm}) = v' \cdot e([\lambda_i(\tau)]_1, [1]_2) + e(\pi', [t(\tau)]_2) \tag{2}$$

Let us look at what the above implies for the discrete logarithms in $\mathbb{G}_1$. By subtracting Eq. (1) from Eq. (2), we can conclude:

$$\begin{aligned} (v-v')\,\lambda_i(\tau) = \mathsf{dlog}_{\mathbb{G}_1}\!\big(\pi'-\pi\big)t(\tau) & \Longrightarrow (v-v')\,\frac{h_i t(\tau)}{m\,(\tau-h_i)} = \mathsf{dlog}_{\mathbb{G}_1}\!\big(\pi'-\pi\big)t(\tau) \\ & \overset{\substack{\text{but with}\\ \text{negligible}\\ \text{probability}}}{\Longrightarrow} (v-v')\,\frac{h_i}{m\,(\tau-h_i)} = \mathsf{dlog}_{\mathbb{G}_1}\!\big(\pi'-\pi\big) \\ & \Longrightarrow \frac{1}{\tau-h_i} = m\frac{\mathsf{dlog}_{\mathbb{G}_1}\!\big(\pi'-\pi\big)}{h_i(v-v')} \\ & \Longrightarrow \left[\frac{1}{\tau-h_i}\right]_1 = \frac{m}{h_i(v-v')}\,(\pi-\pi') \end{aligned}$$

This directly leads to a way to break SDH (Definition 2): the adversary can simply provide $\left(T = \frac{m}{h_i(v-v')}\,(\pi-\pi')\,, c = -h_i\right)$. Notice that the extra elements in pp do not give any extra power to the adversary as they can be computed in polynomial time by the input given in Definition 2. This concludes the proof. □

Theorem 3. *The scheme* SVC *in Fig. 1 satisfies position binding for multiple openings in the AGM under the Discrete Logarithm Assumption.*

A proof of Theorem 3 is in the full version of this paper [Cam26].

4.2 Additional property: cross-commitment aggregation

Our construction extends naturally to support cross-commitment aggregation as defined in [GRWZ20]. This property (informally) states that a prover can prove the opening of multiple commitments at the same time by providing a single proof. We describe the case of two commitments; the general case follows analogously. The blueprint below can be made non-interactive in the random oracle model (we stress that we rely on the random oracle only for the case of cross-commitment aggregation). A security argument is in the full version [Cam26].

Given commitments $\mathsf{cm}, \tilde{\mathsf{cm}}$ to vectors $\boldsymbol{a}, \tilde{\boldsymbol{a}}$ respectively, with corresponding subvector openings $(I, (a_i)_{i\in I})$ and $(\tilde{I}, (\tilde{a}_i)_{i\in\tilde{I}})$ and proofs $\pi^*, \tilde{\pi}^*$, we define the aggregated proof as $\pi := \alpha \cdot \pi^* + \tilde{\alpha} \cdot \tilde{\pi}^*$ where $\alpha, \tilde{\alpha} \leftarrow \mathbb{F}$ are random challenges. Then, the verifier: computes $\Lambda_{\mathsf{LHS}} = \sum_{i\in I}[\lambda_i(\tau)]_1$, $\tilde{\Lambda}_{\mathsf{LHS}} = \sum_{i\in\tilde{I}}[\lambda_i(\tau)]_1$, $\Lambda_{\mathsf{RHS}} = \sum_{i\in I} v_i[\lambda_i(\tau)]_1$, $\tilde{\Lambda}_{\mathsf{RHS}} = \sum_{i\in\tilde{I}} \tilde{v}_i[\lambda_i(\tau)]_1$ ($v_i, \tilde{v}_i$ are claimed values); it accepts iff:

$$e\left(\alpha \cdot \Lambda_{\mathsf{LHS}}, \mathsf{cm}\right) + e\left(\tilde{\alpha} \cdot \tilde{\Lambda}_{\mathsf{LHS}}, \tilde{\mathsf{cm}}\right) = e\left(\alpha \cdot \Lambda_{\mathsf{RHS}} + \tilde{\alpha} \cdot \tilde{\Lambda}_{\mathsf{RHS}}, [1]_2\right) + e\left(\pi, [t(\tau)]_2\right)$$

5 Constructions with Commitment in $\mathbb{G}_1$

It is possible to modify our construction in Section 4 to have a commitment in $\mathbb{G}_1$. However, this change requires the verifier to use terms $[\lambda_i(\tau)]_2$ in $\mathbb{G}_2$. When verifying subvector opening, this implies that either the prover or the verifier must perform such operations. We thus show two variants, both with commitment in $\mathbb{G}_1$, optimized respectively for proving and verification:

- $\Pi^{\mathsf{fastP}}_{\mathbb{G}_1}$, where the prover performs exclusively group operations in $\mathbb{G}_1$;
- $\Pi^{\mathsf{fastV}}_{\mathbb{G}_1}$, where the verifier performs exclusively[11] group operations in $\mathbb{G}_1$.

In both variants the commitment algorithm, on input a vector $\boldsymbol{a}$, computes $\mathsf{cm} = \sum_j a_j[\lambda_j(\tau)]_1$. The preprocessing stage works as in the main scheme.

We describe our constructions as we did in Fig. 1, i.e., proceeding incrementally the subvector case from the single opening setting. In both cases the security argument is essentially the same as the one provided for our main construction.

5.1 The construction $\Pi^{\mathsf{fastP}}_{\mathbb{G}_1}$

Single opening. The single opening prover is exactly as in Fig. 1. The single opening verifier accepts iff the following holds:

$$e\Big(\mathsf{cm}, [\lambda_i(\tau)]_2\Big) = e\Big(v \cdot [\lambda_i(\tau)]_1, [1]_2\Big)) + e\Big(\pi, [t(\tau)]_2\Big)$$

The only change with the original construction above is in switching the terms in the pairing on the left-hand side.

[11] In addition to 3 pairings.

Subvector opening. The prover proceeds exactly as in Fig. 1. The verifier, on input a proof π^* and a subvector $\boldsymbol{v} = \{v_i\}_{i\in I}$ with $I = \{i_1, \ldots, i_\ell\}$, computes:

$$\Lambda^*_{\mathrm{LHS}} = \sum_{i\in I} [\lambda_i(\tau)]_2 \qquad \Lambda^*_{\mathrm{RHS}} = \sum_{i\in I} v_i\,[\lambda_i(\tau)]_1$$

and checks: $e\Big(\mathsf{cm}, \Lambda^*_{\mathrm{LHS}}\Big) = e\Big(\Lambda^*_{\mathrm{RHS}}, [1]_2\Big) + e\Big(\pi^*, [t(\tau)]_2\Big)$.

5.2 The construction $\Pi^{\mathsf{fastV}}_{\mathbb{G}_1}$

This variant essentially works by delegating the verifier's $\mathbb{G}_2$ operations in $\Pi^{\mathsf{fastP}}_{\mathbb{G}_1}$ to the prover. The verifier performs an additional check to ensure that the discrete logarithm of the $\mathbb{G}_2$ element provided by the prover is the expected one.

Single opening. The single opening prover and verifier are the same as in $\Pi^{\mathsf{fastP}}_{\mathbb{G}_1}$.

Subvector opening. On input a set of indices I, the prover outputs $(\pi^*, \Lambda^*_{\mathrm{LHS}})$ with $\pi^* = \sum_{i\in I} \pi_i$ (as in the main scheme) and $\Lambda^*_{\mathrm{LHS}} = \sum_{i\in I}[\lambda_i(\tau)]_2$.

The verifier, on input a proof π^* and a subvector $\boldsymbol{v} = \{v_i\}_{i\in I}$, computes:

$$\Lambda^{(\mathrm{check})}_{\mathrm{LHS}} = \sum_{i\in I} [\lambda_i(\tau)]_1 \qquad \Lambda^*_{\mathrm{RHS}} = \sum_{i\in I} v_i\,[\lambda_i(\tau)]_1$$

and checks the following two equations:

$$e\Big(\mathsf{cm}, \Lambda^*_{\mathrm{LHS}}\Big) = e\Big(\Lambda^*_{\mathrm{RHS}}, [1]_2\Big) + e\Big(\pi^*, [t(\tau)]_2\Big) \quad e\Big([1]_1, \Lambda^*_{\mathrm{LHS}}\Big) = e\Big(\Lambda^{(\mathrm{check})}_{\mathrm{LHS}}, [1]_2\Big)$$

Remark 1 (Optimizing the verifier in $\Pi^{\mathsf{fastV}}_{\mathbb{G}_1}$). We can exploit the special structure of the two pairing equations above and reduce them to a single one as follows. The verifier samples a random $\rho \leftarrow\!\!\$\ \mathbb{F}$ and then checks the equation:

$$e\Big(\mathsf{cm} + \rho[1]_1, \Lambda^*_{\mathrm{LHS}}\Big) = e\Big(\Lambda^*_{\mathrm{RHS}} + \rho\Lambda^{(\mathrm{check})}_{\mathrm{LHS}}, [1]_2\Big) + e\Big(\pi^*, [t(\tau)]_2\Big)$$

6 Preprocessing in Quasi-Linear Time and Updatability

In this section we observe how the structure of our proofs and those in KZG are related (Section 6.1). This has three implications: computing our individual openings *only requires linear time*; for subvector opening, we can preprocess all individual proofs in $O(m \log m)$ time using techniques from [FK23] (Section 6.2); updating proofs can be performed efficiently following [TAB+20] (Section 6.3).

6.1 A first observation on the structure of our proofs

The proof π_i for position i is defined as $\pi_i = [q'_i(\tau)]_1$ where $q'_i(X) = \frac{(f_a(X)-a_i)\lambda_i(X)}{t(X)}$ and $f_a(X) = \sum_{j=1}^m a_j\lambda_j(X)$ is the polynomial encoding of the committed vector. Using the identity $\lambda_i(X) = \frac{h_i\cdot t(X)}{m(X-h_i)}$ from Eq. ($\star$), we can rewrite:

$$q'_i(X) = \frac{\lambda_i(X)}{t(X)}\cdot(f_a(X) - a_i) = \frac{h_i}{m}\cdot\frac{f_a(X)-a_i}{X-h_i} = \frac{h_i}{m}q_i(X)$$

where above we defined q_i as $q_i(X) := \frac{f_a(X)-a_i}{X-h_i}$. The polynomial $q_i(X)$ also corresponds to the quotient polynomial used in the KZG evaluation proof that $f_a(h_i) = a_i$. We can thus compute our openings by computing KZG proofs and the scaling them appropriately, i.e. we can compute $\pi_i = [q_i'(\tau)]_1 = \frac{h_i}{m} \cdot [q_i(\tau)]_1$. Moreover, as a consequence our individual opening proofs can be computed in linear time since computing the quotients $q_i(X)$ consists mainly of a simple shift of coefficients together with an appropriate rescaling[12]. In the next subsection we discuss how to compute all m quotients $[q_i(\tau)]_1$ efficiently.

6.2 Efficiently computing all individual proofs π_i

Here we provide a self-contained description of the techniques from [FK23], which we can leverage in order to compute our proofs. In a nutshell: all quotients $[q_i(\tau)]_1$ can be expressed as evaluations of a single (formal) polynomial with coefficients in $\mathbb{G}_1$. Let $f_a(X) = \sum_{k=0}^{m-1} \hat{a}_k X^k$ be the *monomial* representation of the committed polynomial. We can compute $(\hat{a}_0, \ldots, \hat{a}_{m-1})$ by an inverse DFT from $(a_1, \ldots, a_m)$. We provide a proof of the following lemma in the full version of this paper [Cam26].

Lemma 2 (FK Lemma [FK23]). *For all $i \in [m]$, we have $[q_i(\tau)]_1 = P(h_i)$ where $P(X) = \sum_{j=1}^{m-1} P_j X^{j-1}$, and we define the coefficients $P_j \in \mathbb{G}_1$ as:*

$$P_j = \sum_{k=j}^{m-1} \hat{a}_k [\tau^{k-j}]_1 = \hat{a}_{m-1}[\tau^{m-1-j}]_1 + \hat{a}_{m-2}[\tau^{m-2-j}]_1 + \cdots + \hat{a}_j[1]_1$$

An implication of the statement above is that computing P at all points $\{h_1, \ldots, h_m\}$ gives us the values we were looking for. Since these points correspond to the m-th roots of unity, computing all $P(h_i)$ simultaneously is exactly an FFT of the coefficient vector $(P_1, \ldots, P_{m-1}, 0)$ over the group $\mathbb{G}_1$.

Computing the coefficients of P via Toeplitz multiplication. It remains to show how to compute the coefficients $(P_1, \ldots, P_{m-1})$ efficiently. Observe that:

$$\begin{pmatrix} P_1 \\ P_2 \\ P_3 \\ \vdots \\ P_{m-1} \end{pmatrix} = \begin{pmatrix} \hat{a}_{m-1} & \hat{a}_{m-2} & \hat{a}_{m-3} & \cdots & \hat{a}_1 \\ 0 & \hat{a}_{m-1} & \hat{a}_{m-2} & \cdots & \hat{a}_2 \\ 0 & 0 & \hat{a}_{m-1} & \cdots & \hat{a}_3 \\ \vdots & & & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & \hat{a}_{m-1} \end{pmatrix} \begin{pmatrix} [\tau^{m-2}]_1 \\ [\tau^{m-3}]_1 \\ [\tau^{m-4}]_1 \\ \vdots \\ [1]_1 \end{pmatrix}$$

This is a multiplication of an upper-triangular Toeplitz matrix by a vector of group elements. A description of how to compute it in $O(m \log m)$ time by embedding into a circulant matrix and using FFT, together with the complete preprocessing algorithm, is in the full version of this paper [Cam26].

[12] This is fairly immediate to show and well known. See, e.g., [FK23, §1.2].

Remark 2 (Efficiently deriving the $[\lambda(\tau)]_{1,2}$ parameters without τ). Recall from Fig. 1 that the verifier in our scheme(s) requires some terms encoding the Lagrange polynomials in the group. These can be computed in time $O(m \log m)$ from a standard KZG-style setup even without having the secret point τ and can be carried out using techniques similar to those described above (we stress, however, that deriving these parameters is a once-and-for-all and vector-independent step). Recall that the Lagrange polynomials satisfy $\lambda_i(X) = \frac{1}{m} \sum_{k=0}^{m-1} \omega^{-(i-1)k} X^k$, so evaluating at τ and encoding in $\mathbb{G}_1$ gives $[\lambda_i(\tau)]_1 = \frac{1}{m} \sum_{k=0}^{m-1} \omega^{-(i-1)k} [\tau^k]_1$. This is precisely an inverse DFT over the group $\mathbb{G}_1$ of the vector $\left([\tau^j]_1\right)_{j=0}^{m-1}$. We can compute all $[\lambda_i(\tau)]_{1,2}$ via two inverse FFTs over $\mathbb{G}_1$ and $\mathbb{G}_2$ respectively. The overall running time for this parameter derivation step is in the same ballpark as that of precomputing all individual proofs.

6.3 Updatability of our construction.

Our commitments are immediately updatable using standard techniques: when any entry a_k of the committed vector changes by δ we compute the updated commitment as $\mathsf{cm}' \leftarrow \mathsf{cm} + \delta \cdot [\lambda_k(\tau)]_i$ in the appropriate group $\mathbb{G}_i$. Since the proofs in our construction are scalar multiples of standard KZG evaluation proofs, we can directly apply the mechanism developed for aSVC [TAB+20] based on *update keys*. Update keys are small, position-dependent auxiliary data that enable efficient local proof maintenance: when a vector entry changes, any proof-holder can update their proof in constant time without access to the full vector. Each update key $\mathsf{upk}_j = (\alpha_j, u_j)$ consists of two group elements that can be precomputed during setup, and all m update keys can be computed in $O(m \log m)$ time using DFT-based techniques. Given these keys, when any entry a_k of the committed vector changes by δ, a proof π_j for any j can be updated to a valid proof with respect to the new commitment in $O(1)$ time, using only upk_j and upk_k. A self-contained description of this mechanism is in the full version of this paper [Cam26].

7 Experimental Evaluation

Here we describe our implementation and our experimental results comparing our constructions to aSVC [TAB+20].

Experimental setup. We implement aSVC [TAB+20], our main scheme and our two variants in Rust on top of the `arkworks` library [ark22]. The code is available at:

`https://github.com/matteocam/linear-time-svc`

As an instantiation for the bilinear setting we adopt the curve BLS12-381, which provides ≈128 bits of security. Unless stated otherwise, our benchmarks are run *single-threaded*. The machine we use for our experiments is a common laptop, specifically a Mac Book Air with Apple M2 CPU (8 cores) and 24GB of RAM. We summarize the results for our main construction in Fig. 2.

7.1 Evaluating our main construction

Preprocessing time. The time to compute all proofs is essentially the same in both aSVC and our scheme (our preprocessing requires exactly m additional group operations but otherwise it is the same). It takes ≈20 minutes for $m = 2^{17}$.

Commitment size and commitment time. Our main construction requires committing to the vector in $\mathbb{G}_2$. This implies that our commitment is twice as large in comparison to aSVC and our variants (96B instead of 48B). Also, this implies an additional overhead in our main scheme for computing the commitment (due to $\mathbb{G}_2$ operations being more expensive than their $\mathbb{G}_1$ counterparts). Our experiments show that this consistently yields a 3× slowdown compared to aSVC and our two variants. We believe that, in many settings, the additional costs for commitment times may not be significant because: *i)* committing is a one-off operation; *ii)* the actual commitment times are relatively low (our scheme can commit to a vector of size $\approx 1M$ in less than half a minute without using any special hardware). In settings where commitment size is crucial (or it is crucial to have compatibility with other schemes using the same type of $\mathbb{G}_1$ commitment to the vector), our two variants offer a viable alternative. As we show later in this section, they still provide substantial speedups compared to aSVC.

Opening time. We measure opening time and verification (see below) on relevant opening sizes (up to $\ell \approx 64$K) for a vector of size $m \approx 130K$. For the largest subvectors our construction achieves massive speedups (≈4000×) and can provide an opening within 30ms, in contrast to aSVC which requires approximately two minutes. We also observe large improvements at small vectors, in particular a 62× speedup for small subvectors ($\ell = 4$) and over 100× speedups from $\ell \approx 32$.

Verification time. For the large subvectors our construction achieves significant speedups (≈170×) and can verify an opening in ≈ 0.7s, in contrast to aSVC which requires more than two minutes. The speedups for the smallest subvector sizes ($\ell \leq 8$) are moderate (13%–57% faster than aSVC), but we start observing noticeable speedups starting from relatively small subvectors—over 2× for $\ell \approx 32$, with improvements growing consistently as subvector size increases. As we discuss later in this section, for verification, we can obtain even larger speedups when comparing the two constructions if we exploit parallelism.

7.2 Additional discussion

The effects of parallelism on verification time. We also evaluate our main scheme against aSVC when running both of them using parallel features of the library arkworks. In general, parallelism has minor effects with the exception of our verifier. Our aggregation algorithm performs ℓ additions among group elements and its running times stay the same. For aSVC, the running times of both opening and verification see relatively minor improvements[13], plausibly due

[13] This is especially true for opening, where the improvements are always below 10% compared to the times in Fig. 2. For verification, aSVC obtains a 10% (or less) speedup at most values of ℓ but we observe a more noticeable improvement (≈ 30%) for ℓ in the range 2^9–2^{12}.

to the reliance on FFTs in both algorithms. Our verification algorithm consists of ℓ $\mathbb{G}_1$ operations and one multi-scalar multiplication in $\mathbb{G}_1$. The latter gets substantially optimized through parallelism and as a consequence we obtain a $10\times$ speedup against aSVC already at $\ell = 256$ and over $750\times$ speedup for $\ell \approx 64K$ (with a verification time of ≈ 170ms, $4\times$ faster than single-threaded execution).

Evaluating our variants. Because of their efficiency profile (see Table 1), our variants show that one can obtain roughly the same large speedups of our main scheme while also having a commitment in $\mathbb{G}_1$. For our prover-optimized variant ($\Pi^{\mathsf{fastP}}_{\mathbb{G}_1}$), the improvements for opening times over aSVC are exactly the same as those of our main scheme. The verifier in $\Pi^{\mathsf{fastP}}_{\mathbb{G}_1}$ shows milder speedups over aSVC at the same opening sizes compared to our main construction. For example, it is 40% faster at $\ell = 64$ and $\approx 3\times$ faster at $\ell \approx 1500$ than aSVC. Opening times for our verifier-optimized variant ($\Pi^{\mathsf{fastV}}_{\mathbb{G}_1}$) are generally 3–$4\times$ higher than in our main scheme; despite this, they still show large speedups over aSVC, e.g. $\approx 30\times$ at $\ell = 16$ and $\approx 54\times$ at $\ell = 2048$ (improving further with parallelism).

The role of $\mathbb{G}_2$ operations in the aSVC verifier. The dominant costs in the aSVC verifier stem from the $O(\ell \log^2 \ell)$ field operations and the multi-scalar multiplication (MSM) in $\mathbb{G}_2$. There exist, however, standard techniques that allow delegating this MSM to the prover by replacing it with $\approx \ell$ field operations and one additional pairing. This modification—which also adds one extra $\mathbb{G}_2$ element to the final aSVC proof—is described, e.g., in Appendix I of [BBC+25].

As a sanity check around the *robustness* of our claimed speedups, we have also investigated whether they would hold against this optimized variant of the aSVC verifier. We find, however, that such a change in aSVC would have marginal consequences: from our measurements, this MSM contributes to approximately 30–50% of the time in the aSVC verifier for $\ell \leq 2048$, after which point an overwhelming fraction of the running time consists of field operations (e.g., the MSM takes less than a 4% fraction at $\ell = 32768$). This implies that, on any subvector size, *even if* we considered the MSM *completely free* in the aSVC verifier, we would obtain at most a $2\times$ speedup in aSVC; yet, our construction would still be $10\times$ faster on most subvector sizes. On small subvectors, e.g. $\ell \leq 16$—where the impact of the MSM in aSVC is relatively low—the benefits would not be sufficient to see any substantial change.

Acknowledgments

I am grateful to Ed Felten, Mahak Pancholi and Victor Shoup for the encouraging and insightful discussions, and to the anonymous reviewers, whose comments improved this work. Huge thanks to Dario Fiore for the knowledgeable and generous feedback. Finally, thanks to Domenico Campanelli and Rita Fumarola for (somewhat) tolerating academic writing at their place during the holiday season (and for all the food fueling it, of course).
This work was supported by the Estonian Research Council grant PRG2531.

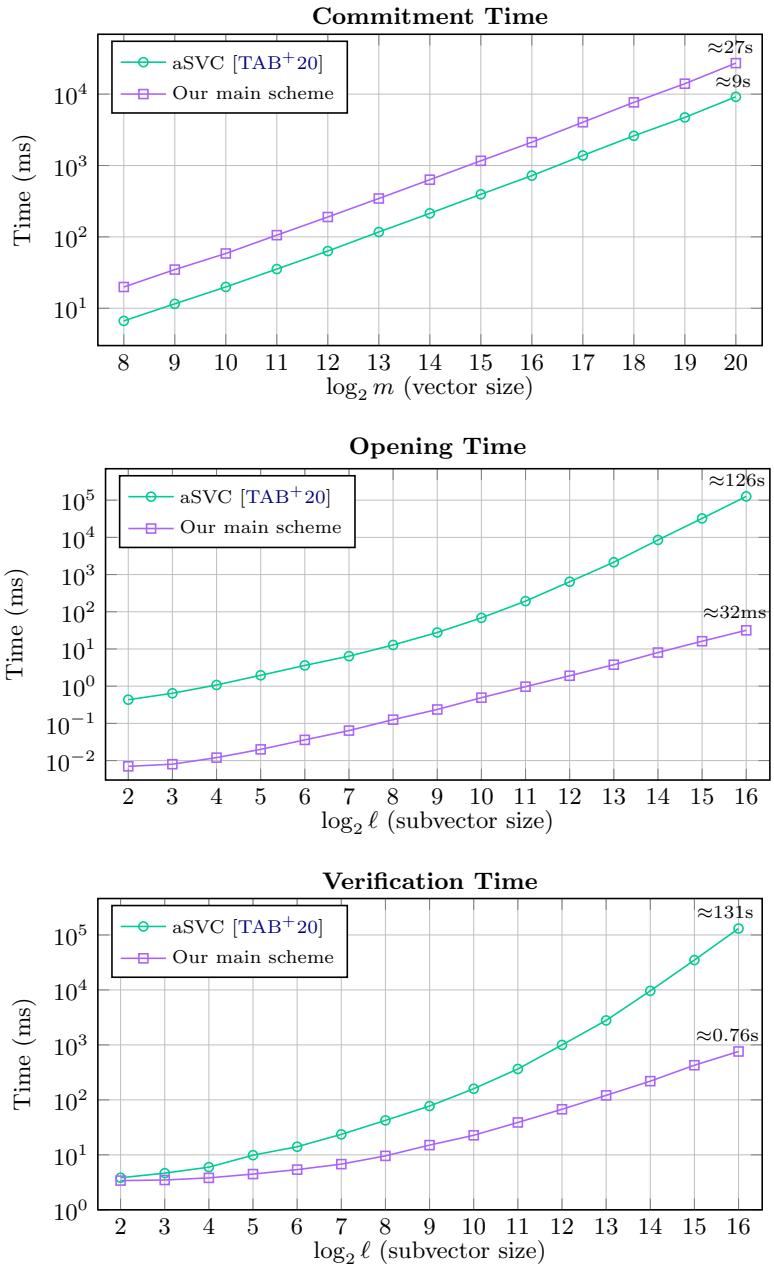

Fig. 2: Performance comparison of aSVC and our main scheme (single-threaded). Top: Commitment time as a function of vector size m (from 2^8 to 2^{20}). Middle and bottom: Opening and verification times for $m = 2^{17}$ as a function of subvector size ℓ (log-log scale).

References

ABC+24. Giuseppe Ateniese, Foteini Baldimtsi, Matteo Campanelli, Danilo Francati, and Ioanna Karantaidou. Advancing scalability in decentralized storage: A novel approach to proof-of-replication via polynomial evaluation. In Leonid Reyzin and Douglas Stebila, editors, *CRYPTO 2024, Part II*, volume 14921 of *LNCS*, pages 3–39, Santa Barbara, CA, USA, August 18–22, 2024. Springer, Cham, Switzerland.

ale21. Aleo trusted setup. https://setup.aleo.org/, 2021.

AR20. Shashank Agrawal and Srinivasan Raghuraman. KVaC: Key-Value Commitments for blockchains and beyond. In Shiho Moriai and Huaxiong Wang, editors, *ASIACRYPT 2020, Part III*, volume 12493 of *LNCS*, pages 839–869, Daejeon, South Korea, December 7–11, 2020. Springer, Cham, Switzerland.

ark22. arkworks contributors. `arkworks` zksnark ecosystem. https://arkworks.rs, 2022.

axi25. Axiom trusted ceremony. https://web.archive.org/web/20250120135649/https://docs.axiom.xyz/docs/transparency-and-security/kzg-trusted-setup, 2025.

BB04. Dan Boneh and Xavier Boyen. Efficient selective-ID secure identity based encryption without random oracles. In Christian Cachin and Jan Camenisch, editors, *EUROCRYPT 2004*, volume 3027 of *LNCS*, pages 223–238, Interlaken, Switzerland, May 2–6, 2004. Springer Berlin Heidelberg, Germany.

BBC+25. Vincenzo Botta, Simone Bottoni, Matteo Campanelli, Emanuele Ragnoli, and Alberto Trombetta. qedb: Expressive and modular verifiable databases (without SNARKs). Cryptology ePrint Archive, Report 2025/1408, 2025.

BBF19. Dan Boneh, Benedikt Bünz, and Ben Fisch. Batching techniques for accumulators with applications to IOPs and stateless blockchains. In Alexandra Boldyreva and Daniele Micciancio, editors, *CRYPTO 2019, Part I*, volume 11692 of *LNCS*, pages 561–586, Santa Barbara, CA, USA, August 18–22, 2019. Springer, Cham, Switzerland.

BCC+17. Nir Bitansky, Ran Canetti, Alessandro Chiesa, Shafi Goldwasser, Huijia Lin, Aviad Rubinstein, and Eran Tromer. The hunting of the SNARK. *Journal of Cryptology*, 30(4):989–1066, October 2017.

BFB+24. Vitalik Buterin, Dankrad Feist, Guillaume Ballet, Piper Merriam, and Gottfried Herold. EIP-6800: Ethereum state using verkle trees. https://eips.ethereum.org/EIPS/eip-6800, 2024.

BFL+22. Vitalik Buterin, Dankrad Feist, Diederik Loerakker, George Kadianakis, Matt Garnett, Mofi Taiwo, and Ansgar Dietrichs. EIP-4844: Shard blob transactions. Ethereum Improvement Proposals, February 2022. Deployed on Ethereum mainnet March 13, 2024 (Dencun upgrade).

BGV11. Siavosh Benabbas, Rosario Gennaro, and Yevgeniy Vahlis. Verifiable delegation of computation over large datasets. In Phillip Rogaway, editor, *CRYPTO 2011*, volume 6841 of *LNCS*, pages 111–131, Santa Barbara, CA, USA, August 14–18, 2011. Springer Berlin Heidelberg, Germany.

Cam26. Matteo Campanelli. Practical subvector commitments with optimal opening complexity. Cryptology ePrint Archive, Paper 2026/118, 2026.

CEO22. Matteo Campanelli, Felix Engelmann, and Claudio Orlandi. Zero-knowledge for homomorphic key-value commitments with applications to

privacy-preserving ledgers. In Clemente Galdi and Stanislaw Jarecki, editors, *SCN 22*, volume 13409 of *LNCS*, pages 761–784, Amalfi, Italy, September 12–14, 2022. Springer, Cham, Switzerland.

CF13. Dario Catalano and Dario Fiore. Vector commitments and their applications. In Kaoru Kurosawa and Goichiro Hanaoka, editors, *PKC 2013*, volume 7778 of *LNCS*, pages 55–72, Nara, Japan, February 26 – March 1, 2013. Springer Berlin Heidelberg, Germany.

CFF+21. Matteo Campanelli, Antonio Faonio, Dario Fiore, Anaïs Querol, and Hadrián Rodríguez. Lunar: A toolbox for more efficient universal and updatable zkSNARKs and commit-and-prove extensions. In Mehdi Tibouchi and Huaxiong Wang, editors, *ASIACRYPT 2021, Part III*, volume 13092 of *LNCS*, pages 3–33, Singapore, December 6–10, 2021. Springer, Cham, Switzerland.

CFG+20. Matteo Campanelli, Dario Fiore, Nicola Greco, Dimitris Kolonelos, and Luca Nizzardo. Incrementally aggregatable vector commitments and applications to verifiable decentralized storage. In Shiho Moriai and Huaxiong Wang, editors, *ASIACRYPT 2020, Part II*, volume 12492 of *LNCS*, pages 3–35, Daejeon, South Korea, December 7–11, 2020. Springer, Cham, Switzerland.

CHAK23. Matteo Campanelli, Mathias Hall-Andersen, and Simon Holmgaard Kamp. Curve trees: Practical and transparent zero-knowledge accumulators. In Joseph A. Calandrino and Carmela Troncoso, editors, *USENIX Security 2023*, pages 4391–4408, Anaheim, CA, USA, August 9–11, 2023. USENIX Association.

CHM+20. Alessandro Chiesa, Yuncong Hu, Mary Maller, Pratyush Mishra, Psi Vesely, and Nicholas P. Ward. Marlin: Preprocessing zkSNARKs with universal and updatable SRS. In Anne Canteaut and Yuval Ishai, editors, *EUROCRYPT 2020, Part I*, volume 12105 of *LNCS*, pages 738–768, Zagreb, Croatia, May 10–14, 2020. Springer, Cham, Switzerland.

CNR+22. Matteo Campanelli, Anca Nitulescu, Carla Ràfols, Alexandros Zacharakis, and Arantxa Zapico. Linear-map vector commitments and their practical applications. In Shweta Agrawal and Dongdai Lin, editors, *ASIACRYPT 2022, Part IV*, volume 13794 of *LNCS*, pages 189–219, Taipei, Taiwan, December 5–9, 2022. Springer, Cham, Switzerland.

eth23. Ethereum KZG ceremony. https://github.com/ethereum/kzg-ceremony, 2023.

Fis19. Ben Fisch. Tight proofs of space and replication. In Yuval Ishai and Vincent Rijmen, editors, *EUROCRYPT 2019, Part II*, volume 11477 of *LNCS*, pages 324–348, Darmstadt, Germany, May 19–23, 2019. Springer, Cham, Switzerland.

FK23. Dankrad Feist and Dmitry Khovratovich. Fast amortized KZG proofs. Cryptology ePrint Archive, Report 2023/033, 2023.

FKL18. Georg Fuchsbauer, Eike Kiltz, and Julian Loss. The algebraic group model and its applications. In Hovav Shacham and Alexandra Boldyreva, editors, *CRYPTO 2018, Part II*, volume 10992 of *LNCS*, pages 33–62, Santa Barbara, CA, USA, August 19–23, 2018. Springer, Cham, Switzerland.

GKM+18. Jens Groth, Markulf Kohlweiss, Mary Maller, Sarah Meiklejohn, and Ian Miers. Updatable and universal common reference strings with applications to zk-SNARKs. In Hovav Shacham and Alexandra Boldyreva, editors, *CRYPTO 2018, Part III*, volume 10993 of *LNCS*, pages 698–728, Santa Barbara, CA, USA, August 19–23, 2018. Springer, Cham, Switzerland.

GLS+21. Alexander Golovnev, Jonathan Lee, Srinath Setty, Justin Thaler, and Riad S. Wahby. Brakedown: Linear-time and post-quantum SNARKs for R1CS. Cryptology ePrint Archive, Report 2021/1043, 2021.

GRWZ20. Sergey Gorbunov, Leonid Reyzin, Hoeteck Wee, and Zhenfei Zhang. Pointproofs: Aggregating proofs for multiple vector commitments. In Jay Ligatti, Xinming Ou, Jonathan Katz, and Giovanni Vigna, editors, *ACM CCS 2020*, pages 2007–2023, Virtual Event, USA, November 9–13, 2020. ACM Press.

GWC19. Ariel Gabizon, Zachary J. Williamson, and Oana Ciobotaru. PLONK: Permutations over Lagrange-bases for oecumenical noninteractive arguments of knowledge. Cryptology ePrint Archive, Report 2019/953, 2019.

KZG10. Aniket Kate, Gregory M. Zaverucha, and Ian Goldberg. Constant-size commitments to polynomials and their applications. In Masayuki Abe, editor, *ASIACRYPT 2010*, volume 6477 of *LNCS*, pages 177–194, Singapore, December 5–9, 2010. Springer Berlin Heidelberg, Germany.

LJGK25. Zhongtang Luo, Yanxue Jia, Alejandra Victoria Ospina Gracia, and Aniket Kate. Cauchyproofs: Batch-updatable vector commitment with easy aggregation and application to stateless blockchains. In Marina Blanton, William Enck, and Cristina Nita-Rotaru, editors, *2025 IEEE Symposium on Security and Privacy*, pages 1947–1963, San Francisco, CA, USA, May 12–15, 2025. IEEE Computer Society Press.

LM19. Russell W. F. Lai and Giulio Malavolta. Subvector commitments with application to succinct arguments. In Alexandra Boldyreva and Daniele Micciancio, editors, *CRYPTO 2019, Part I*, volume 11692 of *LNCS*, pages 530–560, Santa Barbara, CA, USA, August 18–22, 2019. Springer, Cham, Switzerland.

LPR22. Benoît Libert, Alain Passelègue, and Mahshid Riahinia. PointProofs, revisited. In Shweta Agrawal and Dongdai Lin, editors, *ASIACRYPT 2022, Part IV*, volume 13794 of *LNCS*, pages 220–246, Taipei, Taiwan, December 5–9, 2022. Springer, Cham, Switzerland.

LPY12. Benoît Libert, Thomas Peters, and Moti Yung. Group signatures with almost-for-free revocation. In Reihaneh Safavi-Naini and Ran Canetti, editors, *CRYPTO 2012*, volume 7417 of *LNCS*, pages 571–589, Santa Barbara, CA, USA, August 19–23, 2012. Springer Berlin Heidelberg, Germany.

LY10. Benoît Libert and Moti Yung. Concise mercurial vector commitments and independent zero-knowledge sets with short proofs. In Daniele Micciancio, editor, *TCC 2010*, volume 5978 of *LNCS*, pages 499–517, Zurich, Switzerland, February 9–11, 2010. Springer Berlin Heidelberg, Germany.

Nit21. Anca Nitulescu. Sok: Vector commitments. *URL: https://www. di. ens. fr/˜ nitulesc/files/vc-sok. pdf*, 2021.

PRvdM25. Vir Pathak, Sushmita Ruj, and Ron van der Meyden. Vector commitment design, analysis, and applications: A survey. Cryptology ePrint Archive, Report 2025/667, 2025.

PST13. Charalampos Papamanthou, Elaine Shi, and Roberto Tamassia. Signatures of correct computation. In Amit Sahai, editor, *TCC 2013*, volume 7785 of *LNCS*, pages 222–242, Tokyo, Japan, March 3–6, 2013. Springer Berlin Heidelberg, Germany.

SCP+22. Shravan Srinivasan, Alexander Chepurnoy, Charalampos Papamanthou, Alin Tomescu, and Yupeng Zhang. Hyperproofs: Aggregating and maintaining proofs in vector commitments. In Kevin R. B. Butler and Kurt

Thomas, editors, *USENIX Security 2022*, pages 3001–3018, Boston, MA, USA, August 10–12, 2022. USENIX Association.

TAB+20. Alin Tomescu, Ittai Abraham, Vitalik Buterin, Justin Drake, Dankrad Feist, and Dmitry Khovratovich. Aggregatable subvector commitments for stateless cryptocurrencies. In Clemente Galdi and Vladimir Kolesnikov, editors, *SCN 20*, volume 12238 of *LNCS*, pages 45–64, Amalfi, Italy, September 14–16, 2020. Springer, Cham, Switzerland.

TB23. Ertem Nusret Tas and Dan Boneh. Vector commitments with efficient updates. In *5th Conference on Advances in Financial Technologies*, 2023.

WW23. Hoeteck Wee and David J. Wu. Succinct vector, polynomial, and functional commitments from lattices. In Carmit Hazay and Martijn Stam, editors, *EUROCRYPT 2023, Part III*, volume 14006 of *LNCS*, pages 385–416, Lyon, France, April 23–27, 2023. Springer, Cham, Switzerland.

WZ24. Benedikt Wagner and Arantxa Zapico. A documentation of ethereum's PeerDAS. Cryptology ePrint Archive, Paper 2024/1362, 2024.