Vector Commitment Design, Analysis, and Applications: A Survey

VIR PATHAK, University of New South Wales, Sydney, Australia

SUSHMITA RUJ, University of New South Wales, Sydney, Australia

RON VAN DER MEYDEN, University of New South Wales, Sydney, Australia

Due to their widespread applications in decentralized and privacy preserving technologies, commitment schemes have become increasingly important cryptographic primitives. With a wide variety of applications, many new constructions have been proposed, each enjoying different features and security guarantees. In this paper, we systematize the designs, features, properties, and applications of vector commitments (VCs). We define vector, polynomial, and functional commitments and we discuss the relationships shared between these types of commitment schemes. We first provide an overview of the definitions of the commitment schemes we will consider, as well as their security notions and various properties they can have. We proceed to compare popular constructions, taking into account the properties each one enjoys, their proof/update information sizes, and their proof/commitment complexities. We also consider their effectiveness in various decentralized and privacy preserving applications. Finally, we conclude by discussing some potential directions for future work.

CCS Concepts: • **Security and privacy** → **Public key (asymmetric) techniques**; **Hash functions and message authentication codes**; **Mathematical foundations of cryptography**.

Additional Key Words and Phrases: Vector Commitment, Functional Commitment, zk-SNARK, Lookup Argument, Range Proof, Stateless Blockchains, Decentralized Storage

ACM Reference Format:

Vir Pathak, Sushmita Ruj, and Ron van der Meyden. 2025. Vector Commitment Design, Analysis, and Applications: A Survey. In *Proceedings of Make sure to enter the correct conference title from your rights confirmation email (CSUR'25).* ACM, New York, NY, USA, 31 pages. https://doi.org/XXXXXXX.XXXXXXX

1 Introduction

The classical notion of a *commitment scheme* has proven to be extremely useful in designing sophisticated cryptographic primitives, such as zero-knowledge protocols. A commitment scheme [17] is a tuple of algorithms which are run by a *committer* A and a *verifier* B. The aim of A is to "commit" to a value v by generating some value C, which is sent to B. Later, A can "open" the commitment C by proving to B that v is the value corresponding to C. Such a protocol should satisfy both *binding* as well as *hiding* properties. Binding refers to the infeasibility for an adversary to produce proofs convincing the verifier that v and v' are the values committed in C, where $v \neq v'$. Informally, the scheme is *hiding* if a commitment reveals nothing about the value it is storing. These protocols have a myriad of applications. Notably, using commitments is a popular method for adding zero-knowledge (zk) into an argument of knowledge protocol (AoK), as

Authors' Contact Information: Vir Pathak, University of New South Wales, Sydney, Sydney, New South Wales, Australia, vir.pathak@unsw.edu.au; Sushmita Ruj, University of New South Wales, Sydney, Sydney, New South Wales, Australia, sushmita.ruj@unsw.edu.au; Ron van der Meyden, University of New South Wales, Sydney, Sydney, New South Wales, Australia, r.vandermeyden@unsw.edu.au.

Manuscript submitted to ACM

well as constructing other privacy preserving technologies such as anonymous credentials, coin flipping, or signature schemes [20] [57] [18] [9]. Recent designs of privacy preserving technologies such as zero-knowledge (zk) SNARK designs, zk elementary databases, stateless blockchains, decentralized storage, or lookup arguments have called for *vector commitments* (VC), a more expressive variant of the basic commitment scheme. Our paper specifically focuses on this primitive, its properties, and some of its most promising applications. We now proceed to informally define VCs, as well as the related notions of polynomial and functional commitments.

Vector Commitments: In a vector commitment [26], we model the message being committed to as a large vector. The vector commitment protocol compresses a large vector into a small commitment which is sent to a verifier. Later the verifier can query the prover for a vector value at any position of its choice, and the prover can provide the corresponding value as well as a proof that this value is correct. Unlike the basic commitment scheme, the prover can now provide openings for singular values in the vector, rather than a proof for the entire vector itself. The primitive comes with a *binding* guarantee, namely that a computationally bounded adversary cannot submit two proofs π and π' proving values v_i and v_i' (with $v_i \neq v_i'$) are in position i of the vector. The standard notion of hiding in commitment schemes also extends naturally to VCs, where commitments hide messages which are vectors.

Polynomial Commitments: Instead of committing to a vector and opening at positions, we commit to a polynomial and open it at *evaluations* of the polynomial. Polynomial commitments (PCs) are used to construct SNARKs [71] [12]. Namely, they allow a verifier to query parts of a proof string without needing to see the entire proof.

Functional Commitments: Functional commitments (FC) allow us to commit to a vector v and provide openings for any function in some specified function family evaluated at v. Functional commitments have applications in SNAR(G) construction [33], and witness encryption [24]. We discuss in Section 3 various equivalences between these three types of commitments.

We focus on VCs and comment on the relationships VCs have with polynomial and functional commitments. A detailed discussion on polynomial/functional commitments and their applications is out of the scope of this paper. While polynomial commitments are rich in both literature and application, we believe there is a significantly greater need to systematize the theory for VCs, as similar efforts have already been undertaken for polynomial commitments [71]. We also chose to focus on VCs instead of functional commitments as VCs are an established primitive. Therefore, there exists a rich set of applications and literature for VCs, making it desirable to have a systematizing body of work on the topic.

In this paper, we will give a systematic overview of the landscape of vector commitments and their privacy preserving applications. We comment on VC's relationship with polynomial commitments and functional commitments. Our contributions are the following:

(1) To the best of our knowledge, we gather all known vector commitment schemes which suit the privacy preserving applications discussed in the following sections along with stateless blockchain, SNARK, and decentralized storage applications. We compare them based on properties each construction has, their security assumptions, proof/commitment sizes, public parameter sizes, and the complexity of commitment/proof generation. We choose to examine these specific use cases with special attention as we saw that most VC designs were built with them in mind.
(2) We identify properties vector commitments need in order to instantiate the applications listed in the previous point.

(3) We highlight some future problems to consider. We believe that in order to efficiently instantiate the above applications with commitments, solutions to these problems will be critically important.

Similar works exist, such as [60] by the team at CryptoNet Lab. Our survey differs from this work by discussing more recent results and constructions, giving a detailed comparison on properties and complexities, as well as a discussion on which applications are best suited for each commitment scheme considered. We also consider some additional properties, such as homomorphic proofs and openings and our new notion of *dynamism*. Importantly, our work includes a detailed discussion on privacy preserving VC notions and their applications, such as zero-knowledge protocols, zk table lookups, (vector) range proofs, and zk elementary databases. Finally, we also explain general techniques found across many constructions used to realize properties such as aggregation and updatability.

1.0.1 Paper Organization. In order to motivate VCs, we describe applications we will consider in Section 2. In Section 3 we state the relevant definitions for vector commitments, with a view towards polynomial and functional commitments. In Section 4 we describe desirbale proerties in VC design and in Section 5 we focus on the techniques utilized to achieve such properties. In Section 6 we provide an extensive comparison of popular vector commitment schemes. Finally, we conclude in Section 7 with some open problems.

We briefly mention connections between sections in our paper for a reader's convenience. The applications discussed in the Background/Motivation Section (Section 2) are revisited in the Comparison Section (Section 6) when discussing suitable VCs for our selected applications. We discuss desirable VC properties in Section 4, which were previously motivated in Section 2. Methods to endow VCs with such features are outlined in the section on Construction Techniques (Section 5). Finally, all properties discussed in Section 4 are used as points of comparison between our selected VC constructions in Section 6.

2 Background and Motivation

Without defining them formally (see Section 3), a standard vector commitment allows a user to generate a succinct commitment C for a large vector v. Later, a prover can generate a proof π showing that v_i is the element at the ith position of the vector committed in C (for an i of the prover's choice). This proof can be efficiently verified. A *hiding* VC scheme has the same functionality, except commitments *hide* the vector. More formally, an adversary cannot distinguish between a commitment of the vector v and a commitment of the vector v' (where $v \neq v'$). To motivate VCs and demonstrate their power, we first discuss important applications where a hiding VC is not necessary. In the next section, we will proceed to discuss VC applications where the hiding property will be critical. In Section 6.1, we will revisit these applications and discuss suitable existing VC constructions for instantiating them.

2.1 Applications for non-hiding VCs

2.1.1 Stateless Blockchains. An important problem in the blockchain space is increasing efficiency. Currently, blockchain systems like Ethereum [19] can handle around 15 transactions per second [5]. On the other hand, established centralized services like Visa process 1500 transactions per second on average [30]. In order to compete, blockchain systems must drastically increase their scalability.

One approach to increase scalability is by achieving stateless validation (in account based cryptocurrencies) for transactions through vector commitments. Assume there is a way to map account information to an entry in a large vector $v = [v_1, \ldots, v_n] \in \mathbb{F}_p^n$, where n is the number of accounts. That is, the information for account i is stored as the ith entry in the vector v_i. Miners store a commitment to the current blockchain state vector and every account stores a proof affirming the value v_i in the account's balance. When a transaction between sender i and sender j is posted to the network, sender i submits a proof that the ith position in the vector has enough balance to complete the transaction. If a miner wants to add this transaction to a block he is working on, they will now include this proof as well. To check validity of the transaction, a validator can verify the proof and subsequently update the commitment accordingly. Once a new block is posted, new accounts can update their proofs so that they can verify against a commitment reflecting the updated vector of account information. Note that this requires the VC scheme to be able to efficiently update its commitment to a new vector, as well as users being able to update their proofs without having to recompute these values from scratch (see Section 4.2).

2.1.2 SNARK Construction. A popular way to build SNARK systems [71] for all NP is to combine Merkle commitments and PCPs through the "CS Proofs" Paradigm [73]. PCPs exist for all of NP [34]. In order to achieve the succinctness properties required in a SNARK, a prover submits a Merkle commitment to the PCP string π. When a verifier (simulating the PCP verifier) wants to see some part of the PCP string, the prover can send this portion of the string to the verifier as well as a Merkle proof of these positions. In this way, a prover does not have to send a large PCP string but can send a constant sized Merkle commitment instead. Finally, we can remove interaction using the Fiat-Shamir paradigm [36]. The soundness of this paradigm depends on the collision resistance of the hash function used to generate the Merkle commitment/proofs.

This paradigm can immediately be improved replacing the Merkle commitment with a vector commitment with constant opening and commitment sizes. Lai and Malavolta [51] build on this idea using a subvector commitment with a PCP or a linear map commitment [51] with a *linear* PCP [44]. If a regular Merkle style SNARK requires q queries, then $q \log \ell$ dominates the SNARK's proof size, where ℓ is the length of the PCP string. Using the linear map commitment (see Section 3.7) or a subvector commitment (see Section 4), we can combine all q opening proofs (for each of the verifier's q queries) into a single proof, significantly reducing proof size. While a natural modification of the CS proofs paradigm, concrete security bounds were given recently by Chiesa et al. [29].

2.1.3 Decentralized Storage. With a standard centralized storage service, a client sends all the data it would like to be stored to a single node. In the decentralized storage model [43], the client will send different chunks of his data to different storage nodes. Unlike the centralized solution, the decentralized solution [11] has no single point of failure, making it attractive from a security standpoint. Moreover, decentralized solutions tend to be faster, more reliable, and significantly cheaper compared to centralized alternatives. If a client would like a storage node to store a large amount of data, and periodically retrieve some of this data with a guarantee of its correctness, we can use a VC scheme [26]. Viewing the data to be stored by the network as a large vector v, each participating node stores a subvector of v as well as an opening proof of the subvector the node is storing. A Proof of Storage is simply a subvector proof which a node presents to a client/validator, along with the data corresponding to the subvector. To be assured of the correctness of the data presented, one simply runs the verification algorithm specified by the VC scheme. Campanelli et al. [23] instantiate their decentralized storage scheme in this manner. They optimize it even further by providing arguments of knowledge of arbitrary subvectors for the committed vector. In this way, a proof of storage is simply such an argument, which reduces the Storage proof sizes as these arguments are independent of the subvector size.

2.2 Applications for Hiding VCs

Consider a vector commitment which is not only compact and binding, but also hiding. This means that upon seeing a commitment C, an adversary learns no information about the vector committed in C. Now given a commitment C

hiding vector v, imagine that a prover could demonstrate he knows various properties about the vector committed in C without revealing anything information about v. Such a property could be that v has a small norm, or that inputting v into some public linear map gives a certain public output. The ability to generate such proofs in zero-knowledge opens up a wealth of privacy preserving applications, such as compressed Σ protocols, zk-SNARKs, zk-range proofs, zk-lookup arguments, and zk-elementary databases. We now describe how VCs can play a critical role in such constructions.

2.2.1 Compressed Σ Protocols. A Σ protocol is a 3 round interactive protocol between a prover P and verifier V satisfying a knowledge soundness property [71]. The prover wants to convince the verifier of knowledge of a witness w for a corresponding statement x to some relation without revealing any information about the witness. A classic example is Schnorr's Σ protocol [66] proving knowledge of a discrete logarithm instance. Recall that any NP relation can be expressed as an arithmetic circuit [71]. Therefore to build a zk-SNARK for any NP relation, it suffices to build one for circuit satisfiability. It turns out that arithmetic circuits can be linearized [8], so Σ protocols for the relation $R_{\text{linear}} = (P, y; x, (r))$ suffice to build zk-SNARKS for arbitrary NP relations. P is a hiding commitment and y is a public output value. P and y make up the public statement. Then the secret witness consists of a vector x which is committed in P using implicit randomness (r). The Σ protocol proves (in zk) knowledge x and r and also proves $L(x) = y$, where L is some publicly known linear map. Attema et al. [7] devise such a sigma protocol and also manage to *compress* it using secret sharing techniques such that the communication complexity goes from linear to logarithmic in the dimension of the vector x.

2.2.2 Zero-Knowledge Lookup Arguments. We now turn our attention to lookup arguments. Lookup arguments [80] [35] [65] involve two vectors $v_1 \in \mathbb{F}_p^N$ and $v_2 \in \mathbb{F}_p^m$, with N typically being much larger than m. The goal of a prover is to demonstrate that v_2 is a subtable of v_1. That is, every entry in v_2 is also an entry in v_1. Lookup arguments require that the prover complexity runs in time independent of the dimension of the large vector v_1. We would also like this proof to not reveal anything about the structure of the two table v_1 and v_2. Such zk lookups have found usage in decentralized and privacy preserving settings. For example, lookups serve as an efficient instantiation for range proofs. To prove a value lies in a certain range, commit to a table of all numbers in the range. Then run a lookup argument proving the value lies in the table. Campanelli et al. leverage lookups to construct zero-knowledge decision trees [22].

To build such a zero-knowledge lookup argument out of vector commmitments, we generate a hiding commitment to our large table and to our subtable. Then, we run a zero-knowledge protocol establishing knowledge of a witness for the following relation $R_{\text{link}} = (t; f)$ where $t \in \mathbb{F}_p^N$ is the public statement and $f \in \mathbb{F}_p^m$ with f a subtable of t. If such an efficient protocol (with efficiency independent of the size of the large table) exists, we say the vector commitment scheme has *position hiding linkability*. That is, zero-knowledge lookup arguments arise naturally out of position hiding linkable VCs. Zapico et al. [80] were the first to put forth this notion, building a position hiding linkable VC secure in the Algebraic Group Model and noninteractive in the random oracle model. However, their construction still has a small (sublinear) dependence on the dimension of the large table.

2.2.3 Zero-Knowledge Range Proofs. In a standard range proof [31], a single value is committed to, and the committer can prove in zero-knowledge that the committed value lies in a certain range. We can do the same with certain hiding *vector* commitments [39] [52]. This allows for range proofs for all (or a subset of) entries in the vector, allowing efficient batch verification in applications like anonymous credentials. Libert's construction [52] also allows for proofs showing the committed vector is *short* (i.e. has small norm). This feature lends itself well in the lattice setting, as they show their construction can be used to verify the validity of ring-LWE ciphertexts (e.g. after homomorphic] computations).

2.2.4 n-Mercurial Commitments and Zero-Knowledge Elementary Databases. Chase first formalized the notion of *mercurial commitment* [27]. Informally, it allows for both *hard* and *soft* commitments (and openings) to messages. Hard commitments have binding requirements akin to standard commitment schemes. On the other hand, soft commitments can be (soft) opened to any message. We require that the hard and soft mechanisms are indistinguishable. Analogously, one can define n-mercurial commitments as the mercurial version of standard vector commitments. Catalano and Fiore show that given a VC scheme and a mercurial commitment scheme, it is possible to construct an n-mercurial commitment scheme [26]. They subsequently realize zero-knowledge elementary databases via a generic construction using n-mercurial commitments and Merkle trees.

2.2.5 All but one VC. We briefly mention all but one Vector Commitments [18] and their applications. All but one VCs allow a user to generate a hiding commitment to a vector and later provide an opening for all but one of the entries (which reveals nothing about the last entry).

To understand their importance, consider zero knowledge proof systems obtained from the MPC-in-the-head (MPC-ith) [45] paradigm. Let f be a representation for an arbitrary circuit over $\mathbb{F}_p$ and let $x \in \mathbb{F}_p^n$ be a witness for f (so $f(x) = 1$). Use an $n-1$ out of n secret sharing mechanism [67] to derive shares $\{[x]_i\}_{i=1,\ldots,n}$ of x. Then a prover can compute (on his own) arbitrary MPC protocol computing f, taking the shares as input. Here, the prover simulates all n parties in the computation. Since $f(x) = 1$, the result of this MPC computation also returns 1.

The prover commits to the views of all n parties and sends the commitment to the verifier. A verifier checks that the output of the MPC is 1. If this is the case, the verifier requests to see the views of a random subset of $n-1$ parties. The prover provides the views along with opening proofs which verify against the commitment. Once received from the prover, the verifier checks the veracity of the views it received by running the commitment verification algorithm. If this check passes, the verifier next checks if the views are consistent with an honest run of the MPC protocol on the input shares of the corresponding $n-1$ parties. The verifier accepts if this is the case.

We note that an all but one VC is ideal for the functionality required for commitments used in MPC-ith based zero knowledge protocols, as they would generate a succinct commitment to all the parties' views and provide openings for all but one of the views. Furthermore, such an opening reveals nothing about the last view, allowing us to appeal to properties of MPC and secret sharing in order to guarantee security.

3 Preliminaries and Definitions

In this section, we provide the necessary definitions for describing vector commitments. We follow [26] in defining the basic algorithms and their properties and will refer elsewhere when describing more complex properties.

3.1 Notation

Let $\mathbb{F}_p$ be a finite field (p being a prime).

Definition 3.1 (Vector). By *vector*, we mean a tuple of values $v = [v_1, v_2, \ldots, v_n]$ where each entry v_i is an element of $\mathbb{F}_p$. We also denote vectors by $x = [x_1, x_2, \ldots, x_n]$.

Let $I \subseteq \{1, \ldots, n\}$ be an index set. Then we say the vector $x_I := (x_k)_{k \in I}$ is the I–subvector of the vector x.

Definition 3.2 (Linear Map). Let m and n be integers greater than 0. We say a function $f : \mathbb{F}_p^n \to \mathbb{F}_p^m$ is linear if for all $v_1, v_2 \in \mathbb{F}_p^n$ and $\alpha \in \mathbb{F}_p$, we have

$$f(\alpha v_1 + v_2) = \alpha f(v_1) + f(v_2).$$

Definition 3.3 (Negligible Function [15]). A function $f : \mathbb{N} \to \mathbb{R}$ is said to be in the class of *negligible* functions (written as $f \in negl(\lambda)$) if for all $c \geq 0$ there exists $n_0 \in \mathbb{N}$ such that for all integers $n \geq n_0$, we have $|f(n)| < \frac{1}{n^c}$.

That is, a negligible function is a function which decreases faster than the inverse of any polynomial.

Definition 3.4. Let λ be a parameter. The class $poly(\lambda)$ denotes the functions which are bounded from above by a polynomial in λ.

Definition 3.5. We say a probabilistic algorithm is in the class Probabilistic Polynomial Time (PPT) if its expected running time is bounded from above by a polynomial in its input size.

3.2 Pairing Preliminaries

We briefly describe cryptographic pairings, which are an important technique in desigining many vector and polynomial commitment schemes. We will see their importance in the following sections.

Definition 3.6 (Bilinear Pairing [71]). Let $\mathbb{G}_0, \mathbb{G}_1, \mathbb{G}_T$ be three cyclic groups of prime order p and let $g_0 \in \mathbb{G}_0, g_1 \in \mathbb{G}_1$ be generators. A pairing is an efficiently computable function $e_0 : \mathbb{G}_0 \times \mathbb{G}_1 \to \mathbb{G}_T$ satisfying the following:

(1) Bilinearity: For all $u, u' \in \mathbb{G}_0$, $v, v' \in \mathbb{G}_1$, we have

$$e(u \cdot u', v) = e(u, v) \cdot e(e', v)$$

$$e(u, v \cdot v') = e(u, v) \cdot e(u, v').$$

(2) Non-degenerate:

$$g_T = e(g_0, g_1)$$

is a generator of $\mathbb{G}_T$.

When $\mathbb{G}_0 = \mathbb{G}_1$, we say the pairing is *symmetric*.

Note that bilinearity implies the following: for all $\alpha, \beta \in \mathbb{Z}_p$, we have

$$e(g_0^{\alpha}, g_1^{\beta}) = e(g_0, g_1)^{\alpha\beta} = e(g_0^{\beta}, g_1^{\alpha}).$$

3.3 Vector, Polynomial, and Functional Commitment

In this subsection, we define vector commitments, polynomial commitments (PCs), and functional commitments (FCs). We comment on the similarities between each type of commitment scheme, and how one can be instantiated with another. We consider finite fields of the form $\mathbb{F}_p$, where p is a prime.

3.4 Vector Commitment

Definition 3.7 (Vector Commitment). We say a *vector commitment* is a tuple of the following algorithms:

Setup$(1^{\lambda}, n, (r))$: Given the security parameter λ and the length $n = poly(\lambda)$ of the vector being committed, output public parameters pp. From here on, we assume all other algorithms in the vector commitment definition take pp as an implicit input. The algorithm also implicitly takes a randomness parameter r, hence written in parentheses in the algorithm signature. If this randomness is public, we say the vector commitment has *trustless* setup. Otherwise, we say it has *trusted* setup.

Com$(x, (r))$: Outputs a commitment C for a vector x of length $n = poly(\lambda)$ previously specified during the *Setup* phase. Some auxiliary information *aux* may also be generated. Some VC schemes may also take an implicit randomness input (r). The *aux* data may be information such as proving (opening) or updating keys which make these procedures more efficient. From here on, we generally do not write the randomness (r) as an explicit input unless it is a necessary point in the discussion.

Open(x_i, i, aux): Possibly using some auxiliary information *aux*, the opening algorithm generates a proof π_i convincing a verifier that x_i is the value at the ith position in vector x.

Verify(C, y, i, π_i): Given a commitment C, a value y, a position i, and a proof π_i, a verifier outputs 1 or 0, respectively indicating an acceptance or rejection of the proof that y is the ith value in the committed vector (i.e., $y = x_i$).

VC's should be *correct*. That is, the verifier should be convinced when presented with the correct value x_i for a position i in the vector x as well as the corresponding proof.

Definition 3.8 (Correctness). Let p be a prime. We say a VC scheme is *correct* if the following holds for all $\lambda \in \mathbb{N}$, for all $n \in poly(\lambda)$, and for all $x = (x_1, ..., x_n) \in \mathbb{F}_p^n$: if $pp \leftarrow \texttt{Setup}(1^\lambda, n, (r))$, $C \leftarrow \texttt{Com}(x)$, and $\pi_i \leftarrow \texttt{Open}(x_i, i, aux)$, then for all $i \in [n]$, **Verify**(C, x_i, i, π_i) outputs 1 with overwhelming probability.

We would also like our public parameters, commitments, and proofs to be *compact* (succinct). That is, their length should be independent of the length of the vector being committed. Finally, vector commitments should satisfy some security guarantees. Namely, they should satisfy a certain *binding* property, which requires that it is computationally infeasible for an adversary to generate convincing proofs π_i and π_i' for values x_i and x_i' (with $x_i \neq x_i'$). We reproduce the formal definition from [26].

Definition 3.9 (Binding). We say a vector commitment satisfies position binding [26] if for all $i = 1, \ldots, n$ and all PPT adversaries $\mathcal{A}$ the following probability is negligible in the security parameter:

$$Pr\left[\texttt{Verify}(C, x_i, i, \pi_i) = 1 \wedge \texttt{Verify}(C, x_i', i, \pi_i') = 1\right]$$

where

$$(C, x_i, x_i', i, \pi_i, \pi_i') \leftarrow \mathcal{A}(pp)$$

and

$$pp \leftarrow \texttt{Setup}(1^\lambda, n, (r)).$$

We give an example VC construction in what follows. We conclude this section by noting that some readers may have come across the related notion of a *cryptographic accumulator* [13]. Accumulators are similar to vector commitments, but accumulator proofs only demonstrate set membership. On the other hand, VC proofs establish the *position* of a certain element in a committed vector. Moreover, universal accumulators provide proofs showing set *non-membership*. Acharya et al. recently put forth a similar notion for vector commitments [1].

3.4.1 A Simple VC Construction. We present a vector commitment scheme due to Catalano and Fiore [26]. Binding reduces to the Squared Computational Diffie Hellman Assumption, which has been shown to be equivalent to CDH.

Setup$(1^\lambda, n)$: Generate two bilinear groups $\mathbb{G}, \mathbb{G}_T$ of prime order p and a pairing $e : \mathbb{G} \times \mathbb{G} \rightarrow \mathbb{G}_T$. Take a random generator $g \in \mathbb{G}$. Randomly sample $z_1, \ldots, z_n \in \mathbb{Z}_p$ and set $h_i = g^{z_i}$, $h_{i,j} = g^{z_i z_j}$ for $i = 1, \ldots, n$ and $i \neq j$. Output

$$pp = (g, \{h_i\}_{i=1,\ldots,n}, \{h_{i,j}\}_{i \neq j, i,j=1,\ldots,n})$$

Com$(pp, x = [x_1, \ldots, x_n])$: Using the public parameters, compute

$$C = h_1^{x_1} \cdots h_n^{x_n}.$$

Output C and auxiliary information $aux = [x_1, \ldots, x_n]$.

Open(pp, i, x_i, aux): Compute

$$\pi_i = \prod_{j=1, j\neq i}^{n} h_{i,j}^{x_j} = \prod_{j=1, j\neq i}^{n} \left(h_j^{x_j}\right)^{z_j}.$$

Output π_i.

Verify(C, x, i, π_i) : Check if

$$e(C/h_i^x, h_i) = e(\Lambda_i, g).$$

Accept if the equation holds and reject if it does not.

Correctness: To see correctness, note that

$$\begin{aligned} e(C/h_i^{x_i}, h_i) &= e(h_1^{m_1} \cdots h_{i-1}^{x_{i-1}} h_{i+1}^{x^{i+1}} \cdots h_n^{x_n}, h_i) \\ &= e(g^{z_1 x_1} \cdots g^{z_{i-1} x_{i-1}} g^{z_{i+1} x_{i+1}} \cdots g^{z_n x_n}, g^{z_i}). \end{aligned}$$

On the other hand,

$$e(\Lambda_i, g) = e(g^{z_i z_1 x_1 + \cdots + z_i z_{i-1} x_{i-1} + z_i z_{i+1} x_{i+1} + \cdots + z_i z_n x_n}, g).$$

By the bilinearity of the pairing map, these two quantities are equal and correctness follows.

The main drawbacks of this scheme is that the public parameters grow quadratically in the length of the vector being committed to. That is, this scheme lacks *compactness*. The other problem is that this VC requires a trusted setup, as it generates discrete logarithms z_i which could break binding if they were known. We also remark that this scheme also supports efficient proof and commitment updates. A nice property of this scheme is that its verification procedure is fast: the algorithm just requires a single pairing computation.

3.5 Polynomial Commitment

Definition 3.10 (Polynomial Commitment [71]). Let $\mathbb{F}$ be a field. A *polynomial commitment* over $\mathbb{F}$ is a tuple of four algorithms:

Setup$(1^\lambda, D)$: This algorithm takes in a security parameter and an upper bound on the degree of the polynomial we would like to commit to. It generates public parameters pp.

Commit$(pp, p(X))$: This algorithm takes in public parameters and polynomial over $\mathbb{F}$. It outputs a commitment C to that polynomial.

Open$(pp, p(X), z, v)$: This algorithm takes in public parameters pp, a polynomial $p(X)$, an input z and a value v. It outputs a proof π asserting $p(z) = v$.

Verify(C, pp, π, z, v) : This algorithm takes in a commitment C, a public parameter string pp, a proof π, and values z, v. It runs a check to determine if the proof asserting $p(z) = v$ is correct, where p is the polynomial committed to in C.

For an example of a polynomial commitment construction, we refer to the popular KZG [47] polynomial commitment scheme. Due to its fast verification and succinct proof/commitment sizes, KZG is a popular choice for instantiating zk-SNARKs in applications like zkRollups [5] , data availability sampling protocols, and Verkle trees [49]. Binding follows from a Diffie Hellman type of assumption.

Definition 3.11 (Polynomial Evaluation Binding [53]). A polynomial commitment scheme PC is said to be computationally binding if any PPT adversary $\mathcal{A}$ has negligible advantage in winning the following game:

(1) The challenger generates public parameters pp by running **Setup**$(1^\lambda, D)$ and gives pp to $\mathcal{A}$.

(2) The adversary $\mathcal{A}$ outputs a commitment C, an input x, and two values y, y' and two proofs π, π'. $\mathcal{A}$ wins the game if (i): $y \neq y'$ and (ii): $\mathbf{Verify}(C, pp, \pi, x, y) = \mathbf{Verify}(C, pp, \pi', x, y') = 1$.

Correctness, binding, and compactness are natural adaptations of the correctness/binding definitions givens for VCs. A correct polynomial commitment scheme ensures that a verifier accepts a correct proof. A binding PC scheme asserts that proofs showing $p(x) = z$ and $p(x) = z'$ with $z \neq z'$ are both accepted with negligible probability. Compactness requires the sizes of commitments and proofs to be independent of the degree of polynomials being committed.

3.6 Functional Commitment

Definition 3.12 (Functional Commitment). A functional commitment is a tuple of the following algorithms:

$\mathbf{Setup}(1^\lambda n, d, (r))$: Given the security parameter λ, length n of the vectors being committed, and the depth $d = poly(\lambda)$ of the circuits we open to, output public parameters pp. The randomness r is as before.

$\mathbf{Com}(x)$: Generates a commitment C to a vector x of length n as well as some auxiliary information aux.

$\mathbf{Open}(f, aux)$: Using the auxiliary information generated during commitment phase, outputs an opening π to the function/circuit f.

$\mathbf{Verify}(C, f, y, \pi)$ uses π to verify that the functon f applied to the vector commmitted in C yields output y.

Correctness, binding, and compactness are defined in the natural way. In this case, compactness requires for proof sizes to be independent of the circuit depth and for commitment sizes to be independent of the vector length.

3.7 Equivalences

A reader may notice the similarities in the definitions between VCs, PCs, and FCs. Indeed, in most cases we are able to instantiate one type of commitment with the other. In this section, we highlight such (partial) equivalences. Trivially, we see that a functional commitment can instantiate a vector commitment.

Next we observe that polynomial commitments generalize vector commitments. Consider the pairs $(x_1, v_1), \ldots, (x_n, v_n)$ where x_i are any field elements with $x_i \neq x_j$ for $i \neq j$. We can commit to the unique degree $n-1$ polynomial $p(X)$ such that $p(x_i) = v_i$ for $i = 1, \ldots, n$. Note that $p(X)$ can be efficiently calculated through Lagrange interpolation given the pairs $(x_1, v_1) \ldots (x_n, v_n)$. PC binding captures the property that an adversary cannot produce openings for p at an input x showing that $p(x) = y_1$ and $p(x) = y_2$ with $y_1 \neq y_2$, which implies VC binding.

Surprisingly, we can also efficiently instantiate PCs with VCs, but we need the VCs to have some degree of expressivity (although less than that of a full blown functional commitment). Namely, we need to be able to commit to vector $x \in \mathbb{F}_p^n$ and generate proofs with respect to inner products [53]. That is, if we commit to x, we need to be able to generate a proof π_y for a vector $y \in \mathbb{F}_p^n$ attesting to the output of the inner product $\langle x, y \rangle$. To commit to a polynomial $p = \sum_{i=0}^{n-1} a_i X^i$, we commit to the coefficient vector a_i. To generate an evaluation proof at input z, we generate a proof π_z for the inner product opening $[a_0, ..., a_{n-1}] \cdot [1, z, ..., z^{n-1}]$. In the literature, VCs which can open to arbitrary linear maps (not just inner products) are called *linear map commitments* [51] (LMC) or linear map vector commitments [25].

We conclude this section by noting that there exist "dual" functional commitments, where a user commits to a *function* and can open the function at any input. This closely matches the traditional polynomial commitment functionality and can be seen as a generalization of it. Moreover, such dual FCs and the notion of FCs we described are actually equivalent via universal circuits [79] [33].

4 Desirable Properties of VC Schemes

In this section we define VC properties critical for efficiently instantiating the applications discussed in Section 2.

4.1 Hiding

If we are to instantiate a privacy preserving protocol using VCs, then the underlying VC will need to have a hiding property. By a (computationally) hiding vector commitment, we mean one where a polynomially bounded adversary cannot distinguish between a commitment to vector v and a commitment to vector v' (where $v \neq v'$ but they both have the same length).

Definition 4.1 (VC (Standard) Hiding). Let v, v' be any two vectors in $\mathbb{F}_p^n$ and let r, r' be any two sources of randomness. We say a VC scheme is *Hiding* if the distributions of $\mathbf{Com}(v, (r))$ and $\mathbf{Com}(v', (r'))$ are computationally indistinguishable.

Note that a totally deterministic committing procedure can never yield a hiding commitment scheme. Therefore we felt it necessary to explicitly mention the random input (r) in this section. Given a hiding vector commitment, many privacy preserving applications (such as the ones from Section 2) would like us to efficiently prove in zero-knowledge that we have knowledge of a vector v hidden in public commitment C and that vector satisfies a certain relation. This notion captures many variants of VC hiding. For example, recall that position hiding linkability [80] asks for a zero knowledge argument of knowledge for the relation $R_{\text{link}} = (t; f)$ where $t \in F_p^N$ is the public statement and $f \in F_p^m$ with f a subtable of t. When building zk SNARKs in Section 2, we needed a Σ protocol for the relation $R_{\text{linear}} = (P, y; x, (r))$. That is, for a public commitment P and a public output value y, we needed to prove knowledge of a vector x and randomness r such that committing x with randomness r results in P, and that $L(x) = y$ for some public linear map L.

Mercurial VCs must satisfy *mercurial hiding*. That is, an adversary cannot distinguish between hard and soft commitments, or between hard openings and soft openings with a hard flag. Moreover, we can efficiently simulate both hard and soft openings. We refer the reader to [26] for the precise definition. Another notion is *position hiding* [28], where position proofs reveal nothing about the entries at unopened positions in the committed vector.

4.2 Supporting Updates

A crucial property for a VC is to be able to support updates to the committed vector. These update algorithms should be more efficient than simply recomputing the updated commitment/proofs from scratch. Updatability is critical for all VC applications. For example in a stateless blockchain framework, account values are constantly changing with each transaction. Therefore it is important that the scheme used to commit to the account vector can efficiently update its commitment and proofs after a change has occurred in one of the account values. Ideally, we would like an algorithm which can update commitments and proofs after multiple vector value changes. We refer to this property as supporting *batch updates*.

$\mathbf{Update}(C, x_i, x_i', i)$: On input, this takes a commitment C, the ith value in the committed vector x_i, a new value x_i', and a position i. The algorithm outputs a new commitment C' which reflects x_i' being the value at the ith position of the committed vector.

$\mathbf{UpdateProof}(C, \pi_j, x_i', i, U)$: This allows anyone who holds a proof π_j for position j in the committed vector to update their proof when the original commitment C gets changed in the ith position from x_i to x_i'. U is public update information which is part of the public parameters generated during Setup time. Alternatively, U may be part of the auxiliary information generated by running the $\mathbf{Com}$ algorithm on a vector.

We remark that the public update information U is sometimes referred to as an *update key*. Alternatively, U can be used to derive the update keys needed to perform the update operations [60]. If this derivation is efficient, we say the VC scheme is *keyless updatable*. Otherwise, the VC is *key updatable*.

Some authors also make a distinction known as *hint updatability*. A VC is said to be *hint updatable* if the VC update algorithms for commitments and proofs additionally require an opening for the position where the update occurred [60].

4.3 Opening to Subvectors

Next we proceed to defining *subvector commitments* (SVC) [51] [25]. These primitives are captured by the same algorithms **Setup**, **Com**, **Open**, and **Verify** as in the basic vector commitment definition. However, the opening/verification algorithms now allow for opening/proving *subvectors* in addition to individual elements. The Setup and Commitment algorithms are exactly the same as before.

Definition 4.2 (Subvector Commitment Opening [72]). $\mathbf{Open}(x_I, I, aux)$: The inputs are an index set $I \subseteq \{1, \ldots, n\}$, a subvector $x_I = (x_k)_{k \in I}$ of the original committed vector x, and some auxiliary information aux. It outputs a proof π_I convincing a verifier that x_I is the I–subvector of the committed vector.

Moreover, the verification algorithm now can take subvectors and subvector proofs as input. That is, the verification algorithm now looks like

$$\mathbf{Verify}(C, x_I', I, \pi_I).$$

Here, C is the commitment to the vector x, x_I' is the prover's claim of what the I–subvector x_I of x is. As before, the verifer outputs 1 or 0 accordingly.

For a subvector commitment, the compactness requirement is naturally modified. That is, the size of a proof for a subvector x_I must be independent of the total vector length $|x|$ or the subvector length $|I|$.

Among others, subvector commitments have found use cases in SNARK design and decentralized storage. Specifically, Lai and Malavolta [51] show how to combine any subvector commitment scheme along with any Probabilistically Checkable Proof system [71] to build a SNARK compiler in Section 2.1.2. We have also seen that opening to subvectors is a property which lends itself well to decentralized storage applications (see Section 2.1.3).

4.4 Aggregation

Next, we add *aggregatability* to our subvector commitment. An aggregatable subvector commitment allows a user to aggregate proofs for individual positions into a compact proof for a single subvector opening. Note, aggregation is public so anyone can run this algorithm. Formally, an aggregatable subvector commitment consists of the previous algorithms **Setup**, **Com**, **Open**, and **Verify**, along with an extra aggregation algorithm which we now describe. Here, the algorithms **Open** and **Verify** support generating and verifying subvector proofs respectively.

Definition 4.3 (Aggregate Proofs [72]). $\mathbf{AggregateProofs}(I, (\pi_i)_{i \in I})$: Given proofs π_i, for all $i \in I$, outputs a subvector proof for the subvector in the original commited vector indexed by I.

As usual, we require the aggregated proof to be compact, which means its size is independent of the vector length and number of proofs the aggregation algorithm takes as input. This notion is also referred to as *one hop aggregation*. When a user is able to aggregate proofs for *subvectors* and further aggregate on those proofs, we say the scheme is *incrementally aggregatable*. We provide the formal definition for incremental aggregation in section 4.8. Aggregatability is a desirable property for all applications discussed in section 2, since it allows for validation of multiple proofs with a single check.

4.5 Trustless Setup

A commitment scheme has a *trusted setup* procedure if it requires a trusted party to run the **Setup** algorithm. This usually arises because **Setup** generates some values which would compromise the binding of the scheme if known by an adversary. A VC scheme with *trustless setup* has a **Setup** procedure that can be run by anyone.

Instantiating stateless blockchains with VC schemes that require trusted setup is undesirable because the participants must trust the third party to destroy the toxic waste and not use it to generate false proofs. To sidestep the trusted setup issues, efforts have been made to decentralize the trusted setup ceremony using multiparty computation protocols [59], [16]. However using MPC to decentralize the trusted setup ceremony turns out to be quite an expensive procedure.

4.6 Dynamism

The need for a dynamic vector commitment scheme is apparent when considering applications to (updatable) decentralized storage as well as (updatable) zero-knowledge elementary databases. Namely, we require an efficient procedure to update a commitment and proofs reflecting a change in the length of the underlying vector.

Definition 4.4. Let VC = **Setup**, **Com**, **Open**, **Verify** be a vector commitment scheme. We say VC is *dynamic* if there exists an algorithm $\mathtt{Add}(i, C, x)$ where i is a position in vector $v = [v_1, \ldots, v_n]$, C is a commitment to v, and x is a value we would like to insert at position i in the vector. The algorithm $\mathtt{Add}(i, C, x)$ updates the commitment C to a commitment C' which is a commitment to the vector $v' = [v_1, \ldots, v_{i-1}, x, v_i, v_{i+1}, \ldots, v_n]$.

We also require an $\mathtt{AddPf}(C, i, j, \pi_j)$ which allows a user holding a proof for the value at position j to update his proof when v has been changed to v'. Note that when $v = [v_1, \ldots, v_n]$ changes to $v' = [v_1, \ldots, v_{i-1}, x, v_i, v_{i+1}, \ldots, v_n]$, then **AddPf** must be run on all openings for positions $1, \ldots n$ in the old vector v. Therefore, we desire a property similar to the maintainability property discussed in Section 4.7. Namely after an insertion/deletion applied to the committed vector, we would like the VC scheme to be such that updating proofs for all positions is sublinear in the vector length.

We also require analogous $\mathtt{Del}(i, C)$ and $\mathtt{DelPf}(C, i, j, \pi_j)$ algorithms for when a value has been deleted from the vector.

4.7 Maintainability

Maintainability is a stronger notion of updatability. It requires that updating proofs for *all* positions be sublinear in the vector length. With maintainability, updating the proofs of all participants in the network after a block of transactions (and a corresponding change in the blockchain state vector) would be a cheap operation.

Definition 4.5. Let VC be a vector commitment scheme specified by the tuple of algorithms **Setup**, **Com**, **Open**, and **Verify** and let $x = (x_1, ..., x_n) \in \mathbb{F}_p^\ell$ be a vector committed to using VC. We say VC is *maintainable* if there exists an algorithm **UpdateAllProofs** which takes as input a position i of the vector x, an value δ reflecting the amount x_i has changed by after an update, and proofs $\pi_1, ..., \pi_n$. The algorithm returns updated proofs $\pi'_1, ..., \pi'_n$ which verify against the updated vector $(x_1, ..., x_i + \delta, ..., x_n)$.

Srinivasan et al., [69] were the first to achieve both aggregation *and* maintainability by introducing *multilinear* trees. Importantly, the multilinear tree structure eliminates the need to read in all the current proofs as input to

UpdateAllProofs when an entry in the vector changes. The balanceproofs compiler [77] takes any aggregatable VC scheme and produces a new VC scheme which is both maintainable and aggregatable. In this way, the authors obtain a maintainable and aggregatable VC scheme by instantiating their compiler with Tomescu et al's [72] aSVC scheme.

4.8 Incremental Aggregation

Definition 4.6. A VC scheme with subvector openings is called incrementally aggregatable [23] if there exists algorithms **Agg**, **Disagg** working as follows:

Agg$(pp, (I, x_I, \pi_I), (J, x_J, \pi_J))$: takes public parameters and two triples as input. Each input consists of an index set I or J, the corresponding subvector x_I or x_J, and the corresponding subvector proof π_I or π_J. It outputs a proof π_K for the K−subvector of x_K of x, where $K = I \cup J$.

Disagg(pp, I, x_I, π_I, K): takes in public parameters and a triple consisting of an index set I, the corresponding subvector x_I, the corresponding subvector proof π_I, and a set $K \subseteq I$. It outputs a proof π_K for the K−subvector x_K of x.

We have standard correctness requirements for the **Agg** and **Disagg** algorithms. Note that standard aggregation does not require a **DisAgg** algorithm. Indeed, the only scheme we know of which has such functionality is due to Campanelli et al. [23]. The corresponding compactness property states that the length of all proofs produced by aggregation and disaggregation must be bounded by a polynomial in the security parameter (and independent of the number of proofs being aggregated/disaggregated).

4.9 Cross Commitment Aggregation

We can further extend the notion of aggregation to include aggregating proofs over different commitments. Consider vectors $v^{(1)}, \ldots, v^{(\ell)}$, corresponding commitments $C^{(1)}, \ldots, C^{(\ell)}$ and openings $\pi^{(1)}, \ldots, \pi^{(\ell)}$ (where π_i is an opening to an arbitrary set of positions for the vector committed in $C^{(i)}$). Then the cross commitment aggregation algorithm combines the proofs into one proof which can be used to verify any opening proof $\pi^{(i)}$ against the corresponding commitment $C^{(i)}$.[41]:

Definition 4.7. For $j \in [\ell]$, where C_j is a commitment to a vector x_j, S_j is a set of indices, and π_j is the corresponding subvector proof, there exists algorithms **AggregateAcross** and **VerifyAcross** such that

$$\pi \leftarrow \texttt{AggregateAcross}(\{C_j, S_j, m_j[S_j], \pi_j\}_{j\in\ell})$$

$$b = 1/0 \leftarrow \texttt{VerifyAcross}(\{C_j, S_j, m_j[S_j]\}_{j\in[\ell]}, \pi).$$

The value π is a cross-aggregated proof which opens all subvectors corresponding to the (not cross aggregated) proofs $\pi_1 \ldots, \pi_\ell$. Correctness and binding definitions extend naturally from the basic notions discussed for "vanilla" vector commitments.

The compactness requirement for cross commitment aggregation is naturally obtained from the definition given for standard aggregation.

4.10 Homomorphic Proofs and Homomorphic Openings

We now describe homomorphic properties for commitments and openings [25]. Linear map commitments with homomorphic commitments and homomorphic openings can easily be extended to be aggregatable and cross-commitment aggregatable respectively [25](see Section 5.3).

Definition 4.8. Let v be a vector in $\mathbb{F}_p^\ell$ and let π_1, π_2 be openings corresponding to linear functions f_1, f_2. Let α, β be elements of $\mathbb{F}_p$. If $\alpha\pi_1 + \beta\pi_2$ is an opening for v with respect to the commitment C and the linear map $\alpha f_1 + \beta f_2$, we say the linear map commitment scheme has *homomorphic proofs*.

Homomorphic proofs allow for "unbounded" aggregation, which we discuss in section 5.3.

5 Constructing VCs and Achieving Properties

We highlight some general techniques which many schemes seem to follow in order to achieve various properties. We include additional techniques for updates due to Boneh and Tas [70] in Section 5.4.

5.1 The Pairing Approach

Albrecht et al., [3] observe many commitment schemes based on pairings in bilinear groups follow a pattern which we now outline. The *Setup* algorithm generally begins by computing the following public parameters: a triple of pairing friendly groups $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T)$ (all of prime order p) along with the pairing map. The trusted party running the *Setup* algorithm proceeds to generate a random secret v and computes $\{g(v)\}_{g\in\mathcal{G}}$ where $\mathcal{G}$ is some set of polynomials specified by the algorithm. In KZG polynomial commitments for example, the g polynomials simply raise the input to some power. *Setup* discards v and publishes $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, g_1, \{g_2^{g(v)}\}_{g\in\mathcal{G}})$ where $[k]_\lambda$ refers to the discrete logarithm with respect to some generator in the group G_λ ($\lambda = 1, 2, T$).

To generate a proof, a prover computes $\mu = \sum_{g\in\mathcal{G}} c_g \cdot g_2^{g(v)}$. To verify, the pairing equation check looks like $e(g_1, \mu) = \sum_{g\in\mathcal{G}} g_T^{c_g g(v)}$. We stress that this is only a general phenomena which can be loosely observed in many existing constructions (such as the LMC scheme in Appendix 3.4.1), as opposed to a formulaic method to construct commitment schemes. In [3], Albrecht et al., make this observation to motivate their approach for building lattice analogues for pairing based commitments, thus achieving plausible post quantum security. Instead of generating a secret vector, they notice they can achieve analogous schemes by generating secret *trapdoors* and using them to publish preimages (under an inner product operation with a public vector) for the values $\{g(v)\}_{g\in\mathcal{G}}$. We describe this approach and its limitations in Section 5.6 of the Supplementary Material.

5.2 Achieving Hiding

An easy way to turn any VC scheme into a hiding one is to commit to each element in the vector using a standard (hiding) commitment scheme. Then use the VC scheme with the vector of hiding commitments. Another way to obtain a hiding vector commitment scheme and also achieve homomorphic properties [7] is the following: assume we have a binding and hiding commitment scheme $\mathsf{Com}_{\text{single}}$ where we can only commit to single elements in $\mathbb{F}_p$. Furthermore, assume our commitment scheme satisfies

$$\mathsf{Com}_{\text{single}}(x, (r)) + \mathsf{Com}_{\text{single}}(x', (r')) = \mathsf{Com}_{\text{single}}(x + x', (r + r')).$$

Here, r and r' denote implicit randomness in the single value commitment scheme. Assuming such a scheme exists [63], we can commit to a vector $x \in \mathbb{F}_p^n$ in the following way: sample random $a_i \leftarrow \mathbb{F}_p$ and random r_i. Compute

$w_i = \mathsf{Com}_{\text{single}}(a_i, r_i)$ for all $i \in [n]$. Output the w_i as public parameters along with the public parameters generated at setup time from $\mathsf{Com}_{\text{single}}$. Then to commit to x, compute $\sum_{i=1}^{n} x_i w_i + \mathsf{Com}_{\text{single}}(0, r)$. The hiding property then follows from the hiding property of the single value commitment scheme. Note that we do not have to assume the trusted setup uses random r_i, because the committer add their $\mathsf{Com}_{\text{single}}(0, r)$. This is the approach taken in [7] to get a (homomorphic) hiding vector commitment scheme. Note that this scheme has (zk) linear map openings instead of position openings. To reveal the ith entry in the committed vector, we can simply publish a proof showing $L(x) = x_i$, where L is the linear form defined by taking the inner product with the ith basis vector.

5.3 Aggregation Techniques

One aggregation result [25] due to Campanelli et al., is the following: any linear map VC scheme with homomorphic proofs satisfies *unbounded* aggregation.

Unbounded aggregation is similar to incremental aggregation, except one must keep a "history" of previous aggregation operations. We now outline how we can add this property to a linear map VC with homomorphic proofs, following the example given in [25].

For some linear map VC with homomorphic proofs, let v be a vector committed in C and let π_1, π_2, π_3 be openings to linear maps $f_1(v) = y_1, f_2(v) = y_2, f_3(v) = y_3$. To aggregate π_2 and π_3, compute $\pi_1^* = \pi_2 + \gamma_1 \pi_3$ where $\gamma_1 = H(C, \{(f_2, y_2), (f_3, y_3)\})$ and H is a collision resistant hash function. To aggregate π_1 on top of the previous aggregation, compute $\pi_2^* = \pi_1 + \gamma_2 \pi_1^*$ where $\gamma_2 = H(C, (f_1, y_1), \gamma_1)$. This aggregated proof is valid for the function $f^* = f_1 + \gamma_1 \gamma_2 f_2 + \gamma_2 f_3$. That is, aggregating proofs is essentially taking the original proofs and computing some random linear combination of them. To verify, we check if the latest proof verifies against the corresponding random linear combination of random functions. In this example, we check if π_2^* verifies against $f = f_1 + \gamma_1 \gamma_2 f_2 + \gamma_2 f_3$. Note that this strategy can only work if (1). the scheme has the homomorphic proofs property and (2). the verifier knows what the previous γ coefficients are. To ensure property (2)., [25] stores the "aggregation history" in a tree structure [70].

5.4 Update Techniques

In this section, we look at how "algebraic" VCs (i.e. ones based on group and pairing techniques) handle updates. The idea is publish some public update information U after a commitment is updated so that participants holding proofs can efficiently update proofs. We follow Tas and Boneh's [70] exposition, illustrating this phenomenon with KZG based vector commitments.

Let us instantiate a VC through a KZG commitment. That is for a vector $v = [v_1, \ldots, v_n] \in \mathbb{F}_p^n$ for some finite field $\mathbb{F}_p$, we use KZG to commit to the Lagrange polynomial ϕ determined by the entries in v. For $i \in \mathbb{F}_p$, let $L_i(x)$ be the Lagrange basis polynomial with respect to i.

Let C be the commitment generated with respect to v. Next, suppose that we want to update the vector at positions $(i_j)_{j=1}^{k} \subset n$. That is, in the updated vector v_{i_j} becomes some new v'_{i_j}. Boneh and Tas [70] show that the updated commitment C' is $C \prod_{j=1}^{k} C_{ij}^{v'_{i_j} - v_{i_j}}$ where C_{i_j} is the KZG commitment to the Lagrange basis polynomial with respect to $i_j \in \mathbb{F}_p$. Assuming these commitments to the basis polynomials are publicly known, we can update the vector commitment C to C' knowing only the old vector elements, the updated elements, and the indices where the updates occurred. A similar story occurs when trying to update an opening proof at some position i so that the new proof verifies with respect to the updated commitment. Namely, to update proofs, we only need to know the old and new entries, as well as the indices of updates. However, it takes $k \log(\mathcal{M})$ time to obtain the updated proof, where k is the number of elements that have been updated and $\mathcal{M}$ is the space of possible values that can be entries in the vector v.

5.5 Homomorphic Gadget

To open our vector v to some function f (i.e., generate a proof for $f(v)$), the standard technique in the lattice setting is to use a gadget repurposed from GSW homomorphic encryption [40]. We state the following theorem from [33].

Lemma 5.1. *Let $n, q \in \mathbb{N}$ with $\ell = \lceil \log_2 q \rceil$ and $D = \{0, 1\}$ or $D = \mathbb{F}_{p^{n'}}$ for a prime p that divides q and some $n' \leq n$. There is an efficient deterministic robust matrix encoding for any $v \in D^d$ denoted $v^t \otimes g^t \in \mathbb{Z}_q^{n \times dw}$ where $w = n\ell$, and a deterministic polynomial time homomorphic evaluation algorithm* **Eval**, *where for any function family (we need some minor restriction on the families we can consider) $\mathcal{F} = \{f : D^k \to D\}$*

- **Eval***'s input in square brackets is optional, and when it is provided, the additional output (also in square brackets) is also produced. The non-optional output is unaffected whether or not an optional input is provided.*

- **Eval**$(f \in \mathcal{F}, C \in \mathbb{Z}_q^{n \times kw}[, x \in D^k])$ *outputs a matrix $C_f \in \mathbb{Z}_q^{n \times w}$ [and an integral matrix $S_{f,x} \in \mathbb{Z}^{kw \times w}$], where the additional output $S_{f,x}$ satisfies*

$$(C - x^t \otimes g^t) \cdot S_{f,x} = C_f = f(x)^t \otimes g^t$$

In [33], the public parameter is some random matrix C and the *Eval* algorithm is used to generate the functional commitment C_f. To generate proofs, we use *Eval* again with the vector as our optional input to generate the matrix $S_{f,x}$ as our proof. A verifier simply plugs in C_f and $S_{f,x}$ and checks the above equation holds. This homomorphic gadget is also used in a similar way in the functional commitment of [79].

5.6 Pairing to Lattice Translation

The pairing paradigm can be translated [3] into a version for lattices. Instead of $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, g_1, g_2^{g(v)})$, the Setup generates public parameters of the form $(a, v, \mu_{g \in \mathcal{G}})$ a and v are public vectors, $\mathcal{G}$ is as before, and

$$\langle a, \mu_g \rangle = g(v) \pmod{p}.$$

We note that the μ_g vectors can be generated with preimage sampling algorithms using secret lattice trapdoors [56].

To generate a proof, the strategy is to compute $\mu = \sum_{g \in \mathcal{G}} c_g \mu_g$ for some cleverly chosen constants c_g depending on the commitment and/or the vector/function being committed to. Finally, verification reduces to checking

$$\langle a, \mu \rangle = \sum_{g \in \mathcal{G}} c_g \cdot g(v) \pmod{p}.$$

For a concrete example of this translation strategy, we refer to [3]. We also note that [79] uses lattice preimage sampling to generate public parameters, although the structure of their scheme differs significantly from the above paradigm. Unfortunately, such a structure requires new lattice assumptions, since now we have additional information to solve the underlying Short Integer Solutions (see Appendix A.6) problem [2] (namely trapdoors for matrices related to the original SIS matrix). These assumptions do not have a security reduction to standard lattice assumptions as of now. Moreover, the extractability assumption they propose has been shown to most likely be insecure [78], so we cannot use their constructions to build general SNARKs or arguments of knowledge for specific relations.

Table 1. Reference of Symbols Used

Symbol	Meaning
$n = k \cdot m' = t^r$	Vector Length
α	infinity norm bound of vector corresponding to inner product opening
λ	security parameter
$\lvert\mathbb{G}\rvert$	size of a group element
$\lvert\mathbb{F}\rvert$	size of a field element
d	depth of circuits being opened to
w	width of circuits being opened to
$k \cdot \mathbb{P}$	complexity of k pairing operations
$k \cdot \mathbb{E}$	complexity of computing k group exponentiations
$k \cdot GM$	complexity of k group multiplications
$\mathbb{F}$	complexity of a field operation in $\mathbb{F}_p$
$r \cdot PP$	complexity of computing r pairing products
$\mathbb{M}$	complexity of a matrix vector product
H	complexity of running a collision resistant hash function
HCS	complexity of verification for the homomorphic computation scheme (see [33])
ℓ	size (in bits)of a vector entry
ν	constant in $(0, 1)$
S	upper bound on the number of vector entries to be updated
I	index set for a subvector
$\times$	analysis left for future work
DLog	discrete log problem
CDH	Computational Diffie Hellman
SBDH	Strong Bilinear Diffie Hellman
SIS	Short Integer Solutions
GUO	a Group of Unknown Order Assumption
$nw - BDHE^*$	a variant of the weak bilinear Diffie Hellman Problem
AGM	Algebraic Group Model
GGM	Generic Group Model
ROM	Random Oracle Model

6 Comparison

In this section, we compare various VC constructions. We summarize our comparisons between VC constructions in tables 2, 3, 4, and 5. We define all symbols used in Table 1. In Tables 2, 4, we examine security assumptions, dynamism, homomorphism, and hiding properties, and note if an implementation can be found. In Table 3, we look

Table 2. Properties of Commitment Schemes I

Type	Scheme	Security
Vector	Merkle Tree	CRHF
	Attema [6]	RSA
	Caulk [80]	AGM, ROM, n-SBDH
	Catalano [26] - CDH	CDH
	Catalano [26] - RSA	RSA
	Tomescu [72]	n-SBDH
	Boneh [14]	GUO
	Tas [70]	CRHF
	Campanelli [23] (1)	GUO
	Campanelli [23] (2)	GUO
	Gorbunov [41]	nw-BDHE, AGM, ROM*
	Srinivasan [69]	n-SDH
	Krenn [48]	DBP
	Acharya [1]	CRHF (Post-Quantum)
	Papamanthou [62]	CRHF
	Fleichhacker [38]	n-SBDH
	Chen [28]	CDH
	Wang [77]	n-SBDH
	Peikert [64] [1]	SIS
	Libert [54]	q-Diffie Hellman Exponent
Functional	Wee [79]	$\text{BASIS}_{\text{struct}}$ (Post-Quantum)
	de Castro [33]	SIS(Post-Quantum)
	Wang [75]	$\text{BASIS}_{\text{struct}}$
	Balbas [10] [1]	n-HiKer assumption [10]
	Balbas [10] [2]	Twin-k-R-ISIS [10]
	Lipmaa [55]	span uber assumption [55]
Linear Map	Lai [51] (LMC)	GGM
	Campanelli [25](1)	AGM, DLog
	Campanelli [25] (2)	AGM, Dlog
	Fisch [37]	k-R-ISIS [3] (Post-Quantum)
	Chu [32]	GUO
	Libert [53]	Deja Q[53]

at commitment sizes, proof sizes, and public parameter sizes. Finally in Table 5, we examine proving complexities verification complexities. All properties discussed in Section 4 are discussed in our tables, with the exception of opening to subvectors and dynamism (which is a property that none of the schemes possess). Wherever possible, our complexity is with respect to *subvectors*, where the subvector size is denoted by the size $|I|$ of the corresponding index set. That is, our complexity is with respect to generating/verifying a subvector proof instead of a proof for a single position.

Our primary guideline for including a work in our tables was to check if the work proposes a novel VC, linear map commitment, or FC scheme with functionality similar to our definitions. We also include the balanceproofs VC compiler [77] (which turns an aggregatable VC into a maintainable and aggregatable one) instantiated with the aggregatable subvector commitment scheme [72] for analysis.

Our guidelines for excluding a work are as follows:

Table 3. Comparison of Commitments, Proof, and Public Parameter Sizes.

Scheme	$\|C\|$	$\|\pi\|$	$\|pp\|$
Merkle Tree	$\|H\|$	$O(\log n)$	$O(1)$
Attema [6]	$\|\mathbb{G}\|$	$O(\log n)$	$O(n)$
Caulk [80]	$\|\mathbb{G}\|$	$15\|\mathbb{G}\|, 4\|\mathbb{F}\|$	$O(n)$
Catalano [26] - CDH	$\|\mathbb{G}\|$	$\|\mathbb{G}\|$	$O(n^2)$
Catalano [26] - RSA	$\|\mathbb{Z}_{2^\lambda}\|$	$\|\mathbb{Z}_{2^\lambda}\|$	$O(n)$
Tomescu [72]	$\|\mathbb{G}\|$	$\|\mathbb{G}\|$	$O(n)$
Boneh [14]	$\|\mathbb{G}\|$	$\|\mathbb{G}\|$	$O(1)$
Tas [70]	$\|H\|$	$O(\log n)$	$O(1)$
Campanelli [23](1)	$4\|\mathbb{G}\| + 2\|\mathbb{Z}_{2^{2\lambda}}\|$	$3\|\mathbb{G}\|$	$\|\mathbb{G}\|$
Campanelli [23](2)	$\|\mathbb{Z}_{2^\lambda}\|$	$\|\mathbb{Z}_{2^\lambda}\|$	$O(1)$
Lai [51]	$\|\mathbb{G}\|$	$\|\mathbb{G}\|$	$O(n)\|\mathbb{G}\|$
Campanelli [25](1)	$\|\mathbb{G}\|$	$3\|\mathbb{G}\|$	$O(n)$
Campanelli [25] (2)	$\|\mathbb{G}\|$	$3\|\mathbb{G}\|$	$O(n)$
Gorbunov[41]	$\|\mathbb{G}\|$	$\|\mathbb{G}\|$	$O(n)\|\mathbb{G}\|$
Srinivasan [69]	$\|\mathbb{G}\|$	$O(\log(\|I\| \log n))$	$O(n)$
Wee [79]	$O(\lambda(\log\lambda + \log n))$	$O(\lambda(\log^2\lambda + \log^2 n))$	$n^2\mathrm{poly}(\lambda, d, \log n)$
Fisch [37]	$O(\log n + \log\alpha)$	$O(\log n + \log\alpha)$	$O(n)$
Krenn [48]	$\|\mathbb{G}\|$	$\|\mathbb{G}\|$	$O(n)$
Acharya [1]	$\|H\|$	$O(\log n)$	$O(1)$
Wang [75]	$\mathrm{poly}(\lambda, d, \log n)$	$\mathrm{poly}(\lambda, d, \log n)$	$n^2\mathrm{poly}(\lambda, d, \log n)$
Papamanthou [62]	$\|H\|$	$O(\log n)$	* [a]
Fleischhacker [38]	$\|\mathbb{G}\|$	$\|\mathbb{G}\|$	$O(n)$
Chen [28]	$O(1)$	$O(1)$	$O(n)$
Balbas [10] [1]	$\|\mathbb{G}\|$	$O(d)$	$O(S^5)$
Balbas [10] [2]	$O(\log w)$	$O(d\log^2 w)$	$O(w^5)$
Wang [77]	$\|\mathbb{G}\|$	$\|\mathbb{G}\|$	$O(n)$
Chu [32]	$\|\mathbb{G}\|$	$\|\mathbb{G}\|$	$O(1)$
Peikert [64]	$O(r\log t)$	$O(r^3 t\log^2 t)$	$\widetilde{O}(r^2t^2)$
Lipmaa [55]	×	×	×
Libert [54]	$\|\mathbb{G}\|$	$\|\mathbb{G}\|$	$O(n)$
Libert [53]	$\|\mathbb{G}\|$	$\|\mathbb{G}\|$	$O(n)$

[a] This design uses updatable SNARKs to update subvector proofs, so public parameter sizes depend on the choice of SNARK used to instantiate the construction.

(1) Unless the work adds a property we consider in Tables 2 or 4, we do not include a construction whose basic functionalities work in the same manner as a previous construction
(2) We do not include PC constructions
(3) We do not include all but one VC constructions, as their intended opening functionality differs significantly from standard VC opening functionality.
(4) We do not include schemes whose security is proven in nonstandard models (e.g. the online authority model in [64])

Our process for finding constructions to analyze was to examine works concerning VCs, linear map commitments, or FCs, on or after 2013. We chose 2013 as our cutoff year as that was the year of publication for Catalano and Fiore's [26] seminal work on VCs. We also include a standard Merkle Tree for a baseline comparison. We found all of our works by searching through Google Scholar, Eprint, and the Arxiv websites. To find any potential schemes we missed, we also examined citations of the initial papers we found through this process.

One observation we can make from the tables is that a large number of VCs are built from polynomial commitments via the methods discussed in Section 3.7. In many cases, such VCs are equipped with desirable properties. Tomescu et al. [72] construct a (one hop) aggregatable VC by extending VCs via KZG [47] polynomial commitments. The authors propose using their scheme to instantiate a stateless cryptocurrency network, as the scheme enjoys other important properties necessary for this application such as supporting updates or opening to subvectors. They provide an implemenation in Rust. Similarly, Hyperproofs [69] was constructed by extending VCs via PST polynomial commitments [61]. Their scheme is the only construction which is maintainable and aggregatable, although the balanceproofs compiler [77] can tranform any aggregatable VC into a maintainable one. Hyperproofs also opens to subvectors and enjoys homomorphic properties. As such, they also advocate for their scheme to be used for constructing a stateless cryptocurrency as well as provide a sample implementation in Go. One drawback of VCs obtained in this manner is that they lack incremental aggregation and dynamic properties.

Table 2 also illustrates the current shortcomings in post quantum VCs, many of which are lattice based. Building post quantum VCs from lattices seems to be popular, as designers can take advantage of many algebraic tools in lattice based cryptography, such as homomorphic gadgets or preimage sampling mechanisms. In comparison, other post quantum schemes such as those based on hashing do not enjoy those such structural tools. However, none of the lattice based designs enjoy a trustless setup, with the exception of the functional commitment construction due to de Castro et. al [33] which does not support aggregation. The common reference string produced from the trusted setup process is occasionally critical for aggregation properties. This happens because aggregation generates a random linear combination of the input proofs, which can be verified with a corresponding random linear combination of the common reference string elements [79] [25]. Moreover, these schemes are not based on standard lattice assumptions, such as [10], [79], [3], or [75]. Many of the underlying problems have a similar security structure. That is, their security is based on a variant of SIS where trapdoors for matrices related to the SIS instance are additionally generated. Such problems need to be studied in detail and reductions to standard lattice assumptions must be established for users to have confidence in these schemes' security.

From table 3, we can also notice stark differences between classical VC schemes and lattice based ones. In particular, classical constructions tend to be extremely efficient in terms of space, with many commitments being just a single element in a pairing friendly group or in $\mathbb{Z}_{2^\lambda}$. On the other hand, all lattice based commitments have sizes dependent on the vector length. In the case of functional commitments, proof sizes are also dependent on the circuit depth d and/or

Table 4. Properties of Commitment Schemes II

Scheme	Homomorphism	Hiding	Implementation
Merkle Tree	no	no	yes
Attema [6]	yes	yes[a]	no
Caulk [80]	no	position hiding linkability	yes [81]
Catalano [26] - CDH	no	no	no
Catalano [26] - RSA	no	no	no
Tomescu [72]	no	no	no
Boneh [14]	no	no	yes [46]
Tas [70]	no	no	no
Campanelli [23] (1)	no	no	yes [58]
Campanelli [23] (2)	no	no	yes [23]
Lai [51] (LMC)	yes	no	no
Campanelli [25](1)	yes	no	yes [21]
Campanelli [25] (2)	yes	no	no
Gorbunov [41]	no	no	yes [4]
Srinivasan [69]	yes	no	yes [68]
Wee [79]	yes	no	no
de Castro [33]	yes	no	no
Fisch [37]	yes	no	no
Krenn [48]	yes	no	no
Acharya [1]	no	no	no
Wang [75]	yes	mercurial hiding	no
Papamanthou [62]	no	no	yes [50]
Fleichhacker [38]	yes	yes	yes [74]
Chen [28]	yes	position hiding	no
Balbas [10] [1]	yes	yes	no
Balbas [10] [2]	yes	yes	no
Wang [77]	no	no	yes [76]
Chu [32]	no	no	no
Peikert [64] [1]	yes	no	no
Lipmaa [55]	no	yes[b]	no
Libert [54]	yes	mercurial hiding	no
Libert [53]	yes	yes	no

[a] comes with a sigma protocol for knowledge of linear map openings
[b] comes with a zero knowledge protocol for knowledge of position openings

width w which they open to. Finally, public parameters are almost always dependent on the vector dimension. Schemes based on group of unknown order (GUO) assumptions generally enjoy trustless setup, which tends to result in constant size public parameters. Most of these schemes build on top of a GUO based accumulator from Boneh et. al. [14], who show how to translate the accumulator into a VC scheme. However, GUO based schemes are instantiated with class groups or RSA groups. Such constructions are not post quantum solutions and class group instantiations suffer from efficiency bottlenecks in comparison to other classical designs.

Table 4 indicates that a large number of VC schemes are not hiding. However, we remark that any VC scheme can be turned into a hiding one using the transformation described at the beginning of section 5.2. We also note that the lattice based commitment schemes relying on trapdoor preimage sampling in order to generate commitments and proofs [79] [75] generate hiding commitments. This is precisely because such schemes follow a certain paradigm:

(1) The commitment is is sampled in a statistcally close to random manner.
(2) a proof is generated such that it solves a matrix vector equation involving the commitment and a verification key.
(3) This proof is generated using a preimage sampling mechanism, which outputs a short vector satisfying the equation from the previous point.

Since the commitment generated is statistically close to random, the scheme (information theoretically) hides the committed vectors. Therefore, this framework guarantees strong privacy properties at the expense of requiring a trusted setup. Table 4 also shows that lattice based constructions lack implementations. We hope to see more implementations in order to better understand the real world performance of these constructions.

Most classical VCs based on pairings have homomorphic commitments/proofs, as seen in table 4. Such schemes have commitments which take the form $C = \prod_{i=1}^{n} g_i^{x_i}$, where x_i is the $i$the element of the vector and g_i is the ith element in the public parameter set. Then if $C' = \prod_{i=1}^{n} g_i^{x'_i}$, we see that $C \cdot C'$ is a commitment to $(x_i + x'_1, \ldots x_n + x'_n)$. This shows such commitments are homomorphic. Many of these commitment schemes also have homomorphic proofs because the verification process involves a pairing check. Such schemes take advantage of the bilinear properties of the pairing in order to obtain homomorphic proofs. We note that lattice translation of these pairing based VCs 5.6 will continue to be homomorphic, because the pairing is turned into an inner product operation, which is also bilinear. Finally, we note that many of the lattice based functional commitment schemes in table 4 are homomorphic, as they enjoy the properties of the underlying homomorphic gadget (see Section5.5) at the heart of their constructions.

From table 4, we also note that the construction by Wang et. al. [75] yields the first lattice based mercurial VC scheme. The construction also makes heavy use of the trapdoor preimage sampling properties in order to satisfy the hiding requirements necessary for mercurial commitments. Recall that commitments generated in that way are statistically close to uniform random. While it is unclear how to design a scheme with trustless setup in this way, it is clear that the security properties these techniques yield make it attractive from a VC design standpoint.

6.1 Revisiting Applications

For a stateless blockchain, recall that we need a VC supporting subvector openings as well as aggregation. This way validators can verify a large number of transactions by checking a single aggregated proof. We also need to be able to update commitments and proofs efficiently, since the blockchain state vector changes with each published block of transactions. Some proposals which have these properties are outlined in [72], [14], [79], [23], [41], [51], [25]. For SNARKs instantiated through subvector or linear map commitments, we can use any of the above schemes for a

Table 5. Time Complexity of Proof Generation and Verification

Scheme	Proof Complexity	Verification Complexity
Merkle Tree	$O(\log n)$	$O(\log n)H$
Attema [6]	$O(n)$	$1 \cdot GM + \mathbb{M}$
Caulk [80]	$\widetilde{O}(\|I\|^2 + \|I\| \log n)$	$4\mathbb{P}$
Catalano [26] - CDH	$O(n) \cdot \mathbb{E}$	$O(1)\mathbb{P}$
Catalano [26] - RSA	$n \cdot GM, 1 \cdot \mathbb{E}$	$1 \cdot GM$
Tomescu [72]	$O(n \log n) + O(\|I\| \log^2 \|I\|)$	$O(\|I\| \log^2 \|I\|)$
Boneh [14]	$O(\|I\| \cdot n \log n)\mathbb{G}$	$O(\lambda) + O(\ell \cdot n \log n)$
Tas [70]	$O(\lambda \cdot n \log \ell)$	$O(\log n)H$
Campanelli [23](1)	$\|\mathbb{G}\|$	$O(\ell \cdot m \log(\ell n))\|\mathbb{Z}_{2^{2\lambda}}\| + O(\lambda)\mathbb{G}$
Campanelli [23](2)	$O(n) \cdot GM, 1 \cdot \mathbb{E} \pmod{O(\lambda)}$	$O(1)$
Lai [51]	$O(\lambda n \ell^2)$	$O(\lambda n \ell)$
Campanelli [25](1)	$O(k)$	$O(1)\mathbb{P}$
Campanelli [25] (2)	$O(k)$	$4\mathbb{P}$
Gorbunov[41]	$O(\|I\|n)\mathbb{E}$	$O(\|I\|)\mathbb{E} + O(1)PP$
Srinivasan [69]	$O(n \log n)$	$O(\|I\|)$
Wee [79]	$\times$	$\times$
de Castro [33]	$\times$	$\times$
Fisch [37]	$O(n \log n)$	$O(\log(n\alpha) \log\log(n\alpha))$
Krenn [48]	$O(n) \cdot GM$	$1 \cdot PP + \mathbb{P}$
Acharya [1]	$O(\log n)$	$O(\log n)$
Wang [75]	$O(d)$	$\times$
Papamanthou [62]	$O(\log n)$	$O(\log n)$
Fleischhacker [38]	$O(n) \cdot \mathbb{E}, O(n) \cdot GM$	$2\mathbb{P}$
Chen [28]	$O(n) \cdot GM$	$2\mathbb{P}$
Balbas [10] [1]	$\times$	$O(d)$
Balbas [10] [2]	$\times$	$O(\log n)$
Wang [77]	$O(\sqrt{n} \log n)$	$O(\|I\| \log^2 \|I\|)$
Chu [32]	$O(1)$	$O(n) \cdot \mathbb{E}\ O(\lambda n^2 / \log(\lambda n))$
Peikert [64]	$\times$	$O(r)$
Lipmaa [55]	$\times$	$\times$
Libert [54]	$1 \cdot E, O(n) \cdot GM$	$1 \cdot PP + \mathbb{P}$
Libert [53]	$O(n^2) \cdot \mathrm{GM}$	$1 \cdot E + 1 \cdot GM + 1 \cdot \mathbb{P}$

subvector commitment, or [51], [25] for a linear map commitment. For Decentralized Storage, we need a VC with most of the properties mentioned for the stateless blockchain application. Most importantly, we need a VC scheme which is dynamic. We are unaware of a dynamic VC (to the best of our knowledge).

The types of privacy enhancing applications that a VC can support depends on the types of noninteractive zero knowledge (NIZK) [42] proofs it can support. For example, Caulk's [80] position hiding linkability property means that the scheme is equipped with a zero knowledge argument showing that the values hidden in a committed subtable are also values in a larger committed table. This makes Caulk amenable for lookup constructions. On the other hand, the NIZK proof in [6] concerns knowledge of evaluation of the committed vector under a publicly known linear map, from which the authors show how to design zk SNARKs. Finally, the VC presented in [52] has a NIZK showing the committed vector is "short", making it amenable for range proofs.

7 Open Problems

7.1 Dynamism, Incremental Aggregation, and Supporting Updates

To the best of our knowledge, the only VC scheme which supports incremental aggregation and is dynamic is [23]. Dynamism is critical for the decentralized storage application since it can support data coming from new clients. Incremental aggregation will also be useful for fast verification, since a validator/client can check the data from multiple storage nodes at once by aggregating all of their subvector proofs. However, an important feature their schemes miss is supporting updates (and more generally batch updates). To lead to a simple instantiation of verifiable decentralized storage, we believe an important open problem is to build a compact VC scheme supporting (batch) updates, is incrementally aggregatable, and dynamic. For short proofs of retrievability, such a scheme should also support succinct arguments of knowledge of arbitrary subvectors in a committed vector.

7.2 Post Quantum VCs with Extractable Knowledge Openings

A few vector commitments which we considered have knowledge openings for common relations, such as linear maps or subvectors. All such knowledge openings are extractable (i.e. there exists an algorithm with "rewinding access" to the prover's internal state which can recover the witness for the relation instance [71]) under classical assumptions, such as the algebraic group model. It remains to be seen if we can provide arguments of knowledge for such popular relations and guarantee extractability in a post quantum setting. Albrecht et al. attempt to do this by introducing new knowledge assumptions [3]. However their assumptions were shown to most likely be insecure due to Wee and Wu [78].

One potential area in this direction is to design hiding post quantum VCs which support knowledge openings for a large class of circuits. The hiding VCs which we considered only had knowledge openings for a specific type of relation. This made them useful for a small subset of possible privacy preserving applications. It would be nice to have a general purpose post quantum hiding VC which supports NIZKs for multiple types of relations. A VC with knowledge openings for all NP relations would just be a SNARK, but we could try to build a hiding post quantum functional commitment which has openings for a class of relations which are immediately useful for practical applications, such as Lipmaa's classical FC for sparse polynomials [55].

7.3 Lookups from Vector Commitments

Lookup arguments are extremely similar cryptographic primitives when compared with vector commitments and accumulators. At a high level, we commit to a large set in a VC or accumulator and we reveal subsets of our choice.

A VC proof must respect the positions of the revealed entries, while an accumulator proof simply has to respect set membership. Similarly, a lookup argument proof asserts that the prover's vector is a "subvector" of the larger table vector. However in the lookup case, no notion of position is necessary in the proof, and the subtable can also include repeats of values in the larger table. From this perspective, a lookup argument is similar to an accumulator which supports multiset proofs. However the main difference comes from the complexity requirements. With many vector commitment subvector proofs, the complexity is a function of the length of the vector itself. A lookup argument requires the proving complexity to only be dependent on the length of the subvector. This leads us to ask: given the similarity between primitives such VCs, accumulators, and lookups, can we find a method to transform one of these primitives (e.g. a VC or an accumulator) into a lookup argument and vice versa? The authors of the Caulk lookup scheme [80] take some steps toward this, however their scheme is still somewhat dependent on the size of the large table.

8 Conclusion

In this survey, we presented a systemization of the vector commitment literature. We gave a detailed exposition of vector commitments and their properties, and showed where they fit among related primitives, such as polynomial and functional commitments. We proceeded to discuss properties necessary for many decentralized and privacy preserving applications. Next, we compared and contrasted many of the state of the art vector commitment schemes, discussing what applications they would be suited for. Finally, we ended our discussion with some open problems which we believe to be important for improving decentralized and privacy preserving applications instantiated through VCs. With the increase of VC based instantiations of decentralized and privacy preserving mechanisms, new VC properties and security notions will be put forth. Moreover, constructions will evolve and new paradigms involving VCs will be introduced. We hope that our survey helps identification of new applications, facilitates new/improved commitment designs, and allows researchers to establish new connections between VCs and other cryptographic primitives.

References

[1] Ojaswi Acharya, Foteini Baldimtsi, Samuel Dov Gordon, Daniel McVicker, and Aayush Yadav. 2024. Universal Vector Commitments. In *International Conference on Security and Cryptography for Networks*. Springer, 161–181.

[2] Miklós Ajtai. 1996. Generating hard instances of lattice problems. In *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*. 99–108.

[3] Martin R Albrecht, Valerio Cini, Russell WF Lai, Giulio Malavolta, and Sri AravindaKrishnan Thyagarajan. 2022. Lattice-based SNARKs: Publicly verifiable, preprocessing, and recursively composable. In *Annual International Cryptology Conference*. Springer, 102–132.

[4] Algorand. 2020. pointproofs. https://github.com/algorand/pointproofs/graphs/contributors. [Online; accessed 8-February-2022].

[5] Aditya Asgaonkar. 2022. Scaling Blockchains and the Case for Ethereum. In *Handbook on Blockchain*. Springer, 197–213.

[6] Thomas Attema, Ignacio Cascudo, Ronald Cramer, Ivan Damgård, and Daniel Escudero. 2022. Vector commitments over rings and compressed σ-protocols. In *Theory of Cryptography Conference*. Springer, 173–202.

[7] Thomas Attema and Ronald Cramer. 2020. Compressed-protocol theory and practical application to plug & play secure algorithmics. In *Annual International Cryptology Conference*. Springer, 513–543.

[8] Thomas Attema, Ronald Cramer, and Lisa Kohl. 2021. A compressed Σ-protocol theory for lattices. In *Annual International Cryptology Conference*. Springer, 549–579.

[9] Michael Backes, Aniket Kate, and Arpita Patra. 2011. Computational verifiable secret sharing revisited. In *Advances in Cryptology–ASIACRYPT 2011: 17th International Conference on the Theory and Application of Cryptology and Information Security, Seoul, South Korea, December 4-8, 2011. Proceedings 17*. Springer, 590–609.

[10] David Balbás, Dario Catalano, Dario Fiore, and Russell WF Lai. 2023. Chainable functional commitments for unbounded-depth circuits. In *Theory of Cryptography Conference*. Springer, 363–393.

[11] Davi Pedro Bauer. 2022. Filecoin. In *Getting Started with Ethereum: A Step-by-Step Guide to Becoming a Blockchain Developer*. Springer, 97–101.

[12] Alexandre Belling, Azam Soleimanian, and Bogdan Ursu. 2024. Vortex: A List Polynomial Commitment and its Application to Arguments of Knowledge. *Cryptology ePrint Archive* (2024).

[13] Josh Benaloh and Michael De Mare. 1993. One-way accumulators: A decentralized alternative to digital signatures. In *Workshop on the Theory and Application of of Cryptographic Techniques*. Springer, 274–285.

[14] Dan Boneh, Benedikt Bünz, and Ben Fisch. 2019. Batching techniques for accumulators with applications to IOPs and stateless blockchains. In *Advances in Cryptology–CRYPTO 2019: 39th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 18–22, 2019, Proceedings, Part I 39*. Springer, 561–586.

[15] Dan Boneh and Victor Shoup. 2020. A graduate course in applied cryptography. *Draft 0.5* (2020).

[16] Sean Bowe, Ariel Gabizon, and Ian Miers. 2017. Scalable multi-party computation for zk-SNARK parameters in the random beacon model. *Cryptology ePrint Archive* (2017).

[17] Gilles Brassard, David Chaum, and Claude Crépeau. 1988. Minimum disclosure proofs of knowledge. *Journal of computer and system sciences* 37, 2 (1988), 156–189.

[18] Dung Bui, Kelong Cong, and Cyprien Delpech de Saint Guilhem. 2024. Improved All-but-One Vector Commitment with Applications to Post-Quantum Signatures. *Cryptology ePrint Archive* (2024).

[19] Vitalik Buterin et al. 2014. A next-generation smart contract and decentralized application platform. *white paper* 3, 37 (2014), 2–1.

[20] Jan Camenisch and Anna Lysyanskaya. 2001. An efficient system for non-transferable anonymous credentials with optional anonymity revocation. In *Advances in Cryptology—EUROCRYPT 2001: International Conference on the Theory and Application of Cryptographic Techniques Innsbruck, Austria, May 6–10, 2001 Proceedings 20*. Springer, 93–118.

[21] Mateo Campanelli. 2022. lvc-mon-rust. https://github.com/matteocam/lvc-mon-rust. [Online; accessed 8-February-2022].

[22] Matteo Campanelli, Antonio Faonio, Dario Fiore, Tianyu Li, and Helger Lipmaa. 2024. Lookup arguments: improvements, extensions and applications to zero-knowledge decision trees. In *IACR International Conference on Public-Key Cryptography*. Springer, 337–369.

[23] Matteo Campanelli, Dario Fiore, Nicola Greco, Dimitris Kolonelos, and Luca Nizzardo. 2020. Incrementally aggregatable vector commitments and applications to verifiable decentralized storage. In *Advances in Cryptology–ASIACRYPT 2020: 26th International Conference on the Theory and Application of Cryptology and Information Security, Daejeon, South Korea, December 7–11, 2020, Proceedings, Part II 26*. Springer, 3–35.

[24] Matteo Campanelli, Dario Fiore, and Hamidreza Khoshakhlagh. 2022. Witness Encryption for Succinct Functional Commitments and Applications. *Cryptology ePrint Archive* (2022).

[25] Matteo Campanelli, Anca Nitulescu, Carla Ràfols, Alexandros Zacharakis, and Arantxa Zapico. 2022. Linear-Map Vector Commitments and Their Practical Applications. In *International Conference on the Theory and Application of Cryptology and Information Security*. Springer, 189–219.

[26] Dario Catalano and Dario Fiore. 2013. Vector commitments and their applications. In *Public-Key Cryptography–PKC 2013: 16th International Conference on Practice and Theory in Public-Key Cryptography, Nara, Japan, February 26–March 1, 2013. Proceedings 16*. Springer, 55–72.

[27] Melissa Chase, Alexander Healy, Anna Lysyanskaya, Tal Malkin, and Leonid Reyzin. 2005. Mercurial commitments with applications to zero-knowledge sets. In *Advances in Cryptology–EUROCRYPT 2005: 24th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Aarhus, Denmark, May 22-26, 2005. Proceedings 24*. Springer, 422–439.

[28] Long Chen, Hui Guo, Ya-Nan Li, and Qiang Tang. 2023. Efficient Secure Storage with Version Control and Key Rotation. In *International Conference on the Theory and Application of Cryptology and Information Security*. Springer, 168–198.

[29] Alessandro Chiesa, Marcel Dall'Agnol, Ziyi Guan, and Nicholas Spooner. 2023. Concrete Security for Succinct Arguments from Vector Commitments. *Cryptology ePrint Archive* (2023).

[30] Sherman SM Chow, Ziliang Lai, Chris Liu, Eric Lo, and Yongjun Zhao. 2018. Sharding blockchain. In *2018 IEEE international conference on internet of things (iThings) and IEEE Green computing and communications (GreenCom) and IEEE cyber, physical and social computing (CPSCom) and IEEE smart data (SmartData)*. IEEE, 1665–1665.

[31] Miranda Christ, Foteini Baldimtsi, Konstantinos Kryptos Chalkias, Deepak Maram, Arnab Roy, and Joy Wang. 2024. Sok: Zero-knowledge range proofs. *Cryptology ePrint Archive* (2024).

[32] Hien Chu, Dario Fiore, Dimitris Kolonelos, and Dominique Schröder. 2022. Inner product functional commitments with constant-size public parameters and openings. In *International Conference on Security and Cryptography for Networks*. Springer, 639–662.

[33] Leo de Castro and Chris Peikert. 2023. Functional Commitments for All Functions, with Transparent Setup and from SIS. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 287–320.

[34] Irit Dinur. 2007. The PCP theorem by gap amplification. *Journal of the ACM (JACM)* 54, 3 (2007), 12–es.

[35] Liam Eagen, Dario Fiore, and Ariel Gabizon. 2022. cq: Cached quotients for fast lookups. *Cryptology ePrint Archive* (2022).

[36] Amos Fiat and Adi Shamir. 1986. How to prove yourself: Practical solutions to identification and signature problems. In *Conference on the theory and application of cryptographic techniques*. Springer, 186–194.

[37] Ben Fisch, Zeyu Liu, and Psi Vesely. 2023. Orbweaver: succinct linear functional commitments from lattices. In *Annual International Cryptology Conference*. Springer, 106–131.

[38] Nils Fleischhacker, Mathias Hall-Andersen, Mark Simkin, and Benedikt Wagner. 2025. Jackpot: Non-interactive aggregatable lotteries. In *International Conference on the Theory and Application of Cryptology and Information Security*. Springer, 365–397.

[39] Rui Gao, Zhiguo Wan, Yuncong Hu, and Huaqun Wang. 2024. A Succinct Range Proof for Polynomial-based Vector Commitment. *Cryptology ePrint Archive* (2024).

[40] Craig Gentry, Amit Sahai, and Brent Waters. 2013. Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based. In *Advances in Cryptology–CRYPTO 2013: 33rd Annual Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2013. Proceedings,*

Part I. Springer, 75–92.

[41] Sergey Gorbunov, Leonid Reyzin, Hoeteck Wee, and Zhenfei Zhang. 2020. Pointproofs: Aggregating proofs for multiple vector commitments. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*. 2007–2023.

[42] Jens Groth. 2016. On the size of pairing-based non-interactive arguments. In *Advances in Cryptology–EUROCRYPT 2016: 35th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Vienna, Austria, May 8-12, 2016, Proceedings, Part II 35*. Springer, 305–326.

[43] Daira Hopwood, Sean Bowe, Taylor Hornby, Nathan Wilcox, et al. 2016. Zcash protocol specification. *GitHub: San Francisco, CA, USA* 4, 220 (2016), 32.

[44] Yuval Ishai, Eyal Kushilevitz, and Rafail Ostrovsky. 2007. Efficient arguments without short PCPs. In *Twenty-Second Annual IEEE Conference on Computational Complexity (CCC'07)*. IEEE, 278–291.

[45] Yuval Ishai, Eyal Kushilevitz, Rafail Ostrovsky, and Amit Sahai. 2007. Zero-knowledge from secure multiparty computation. In *Proceedings of the thirty-ninth annual ACM symposium on Theory of computing*. 21–30.

[46] Sanjay Kannan, Alan Flores-Lopez, Eddie Wang, and Zhucheng Yu. 2019. accumulator. https://github.com/cambrian/accumulator. [Online; accessed 8-February-2022].

[47] Aniket Kate, Gregory M Zaverucha, and Ian Goldberg. 2010. Constant-size commitments to polynomials and their applications. In *Advances in Cryptology-ASIACRYPT 2010: 16th International Conference on the Theory and Application of Cryptology and Information Security, Singapore, December 5-9, 2010. Proceedings 16*. Springer, 177–194.

[48] Stephan Krenn, Omid Mir, and Daniel Slamanig. 2024. Structure-Preserving Compressing Primitives: Vector Commitments, Accumulators and Applications. *Cryptology ePrint Archive* (2024).

[49] John Kuszmaul. 2019. Verkle trees. *Verkle Trees* 1 (2019), 1.

[50] Lagrange Labs. 2025. reckle-trees. https://github.com/Lagrange-Labs/reckle-trees. [Online; accessed 8-February-2022].

[51] Russell WF Lai and Giulio Malavolta. 2019. Subvector commitments with application to succinct arguments. In *Advances in Cryptology–CRYPTO 2019: 39th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 18–22, 2019, Proceedings, Part I 39*. Springer, 530–560.

[52] Benoît Libert. 2024. Vector Commitments with Proofs of Smallness: Short Range Proofs and More. In *IACR International Conference on Public-Key Cryptography*. Springer, 36–67.

[53] Benoît Libert, Somindu C Ramanna, et al. 2016. Functional commitment schemes: From polynomial commitments to pairing-based accumulators from simple assumptions. In *43rd International Colloquium on Automata, Languages and Programming (ICALP 2016)*.

[54] Benoît Libert and Moti Yung. 2010. Concise mercurial vector commitments and independent zero-knowledge sets with short proofs. In *Theory of Cryptography Conference*. Springer, 499–517.

[55] Helger Lipmaa and Kateryna Pavlyk. 2020. Succinct functional commitment for a large class of arithmetic circuits. In *International Conference on the Theory and Application of Cryptology and Information Security*. Springer, 686–716.

[56] Daniele Micciancio and Chris Peikert. 2012. Trapdoors for lattices: Simpler, tighter, faster, smaller. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 700–718.

[57] Moni Naor. 1989. Bit commitment using pseudo-randomness. In *Conference on the Theory and Application of Cryptology*. Springer, 128–136.

[58] Nicola, Mateo Campanelli, and Friedel Ziegelmayer. 2019. rust-yinyan. https://github.com/nicola/rust-yinyan?tab=readme-ov-file. [Online; accessed 8-February-2022].

[59] Valeria Nikolaenko, Sam Ragsdale, Joseph Bonneau, and Dan Boneh. 2022. Powers-of-tau to the people: Decentralizing setup ceremonies. *Cryptology ePrint Archive* (2022).

[60] Anca Nitulescu. 2023. Sok: Vector commitments. *URL: https://www. di. ens. fr/˜ nitulesc/files/vc-sok. pdf* (2023).

[61] Charalampos Papamanthou, Elaine Shi, and Roberto Tamassia. 2013. Signatures of correct computation. In *Theory of Cryptography Conference*. Springer, 222–242.

[62] Charalampos Papamanthou, Shravan Srinivasan, Nicolas Gailly, Ismael Hishon-Rezaizadeh, Andrus Salumets, and Stjepan Golemac. 2024. Reckle trees: Updatable merkle batch proofs with applications. In *Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security*. 1538–1551.

[63] Torben Pryds Pedersen. 1991. Non-interactive and information-theoretic secure verifiable secret sharing. In *Annual international cryptology conference*. Springer, 129–140.

[64] Chris Peikert, Zachary Pepin, and Chad Sharp. 2021. Vector and functional commitments from lattices. In *Theory of Cryptography: 19th International Conference, TCC 2021, Raleigh, NC, USA, November 8–11, 2021, Proceedings, Part III 19*. Springer, 480–511.

[65] Jim Posen and Assimakis A Kattis. 2022. Caulk+: Table-independent lookup arguments. *Cryptology ePrint Archive* (2022).

[66] Claus-Peter Schnorr. 1990. Efficient identification and signatures for smart cards. In *Advances in Cryptology—CRYPTO'89 Proceedings 9*. Springer, 239–252.

[67] Adi Shamir. 1979. How to share a secret. *Commun. ACM* 22, 11 (1979), 612–613.

[68] Shravan Srinivasan. 2025. hyperproofs-go. https://github.com/hyperproofs/hyperproofs-go. [Online; accessed 8-February-2022].

[69] Shravan Srinivasan, Alexander Chepurnoy, Charalampos Papamanthou, Alin Tomescu, and Yupeng Zhang. 2022. Hyperproofs: Aggregating and maintaining proofs in vector commitments. In *31st USENIX Security Symposium (USENIX Security 22)*. 3001–3018.

[70] Ertem Nusret Tas and Dan Boneh. 2023. Vector Commitments with Efficient Updates. *arXiv preprint arXiv:2307.04085* (2023).

[71] Justin Thaler et al. 2022. Proofs, arguments, and zero-knowledge. *Foundations and Trends® in Privacy and Security* 4, 2–4 (2022), 117–660.

[72] Alin Tomescu, Ittai Abraham, Vitalik Buterin, Justin Drake, Dankrad Feist, and Dmitry Khovratovich. 2020. Aggregatable subvector commitments for stateless cryptocurrencies. In *International Conference on Security and Cryptography for Networks*. Springer, 45–64.
[73] Paul Valiant. 2008. Incrementally verifiable computation or proofs of knowledge imply time/space efficiency. In *Theory of Cryptography: Fifth Theory of Cryptography Conference, TCC 2008, New York, USA, March 19-21, 2008. Proceedings 5*. Springer, 1–18.
[74] Benedict Wagner. 2023. jackpot. https://github.com/b-wagn/jackpot. [Online; accessed 8-February-2022].
[75] Hongxiao Wang, Siu-Ming Yiu, Yanmin Zhao, Zoe L Jiang, and Min Xie. 2024. Lattice-Based Succinct Mercurial Functional Commitment for Circuits: Definitions and Constructions. *Cryptology ePrint Archive* (2024).
[76] Nick Wang. 2023. balanceproofs-go. https://github.com/wangnick2017/balanceproofs-go. [Online; accessed 8-February-2022].
[77] Weijie Wang, Annie Ulichney, and Charalampos Papamanthou. 2023. {BalanceProofs}: Maintainable Vector Commitments with Fast Aggregation. In *32nd USENIX Security Symposium (USENIX Security 23)*. 4409–4426.
[78] Hoeteck Wee and David J Wu. 2023. Lattice-based functional commitments: Fast verification and cryptanalysis. In *International Conference on the Theory and Application of Cryptology and Information Security*. Springer, 201–235.
[79] Hoeteck Wee and David J Wu. 2023. Succinct vector, polynomial, and functional commitments from lattices. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 385–416.
[80] Arantxa Zapico, Vitalik Buterin, Dmitry Khovratovich, Mary Maller, Anca Nitulescu, and Mark Simkin. 2022. Caulk: Lookup arguments in sublinear time. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*. 3121–3134.
[81] Zhenfei Zhang, Khovratovich Dmitry, and Mary Maller. 2022. Caulk. https://github.com/caulk-crypto/caulk. [Online; accessed 8-February-2022].

A Security Assumptions

For completeness, we define the popular security assumptions made by the VC schemes we consider.

A.1 Discrete Logarithm Assumption

Definition A.1. Let G be a cyclic group of some prime order q with $g \in G$ as a generator. Consider the following game between an adversary $\mathcal{A}$ and a challenger:

(1). The challenger computes $\alpha \leftarrow \mathbb{Z}_q$ and sets $\mu = g^\alpha$. He sends μ, a description of G, and the generator g to $\mathcal{A}$.

(2). $\mathcal{A}$ outputs some $x \in \mathbb{Z}_q$.

The adversary $\mathcal{A}$ wins the game if $x = \alpha \pmod q$. We define $\mathcal{A}'s$ advantage in solving the discrete logarithm problem on G (write DLAdv$[\mathcal{A}, G]$ as the probability that $\mathcal{A}$ wins the game.

Definition A.2. We say the discrete logarithm assumption holds for the group G if for all efficient adversaries $\mathcal{A}$, the quantity DLAdv$[\mathcal{A}, G]$ is negligible. The quantity g^α is an instance of the discrete logarithm problem on the group G and that α is a solution to this instance. The assumption essentially states that no efficient adversary can solve the discrete logarithm problem.

A.2 Computational Diffie Hellman Assumption

Definition A.3. Let G, g be as before. We again have a game between an adversary $\mathcal{A}$ and a challenger.

(1). The challenger computes $\alpha, \beta \leftarrow \mathbb{Z}_q$ and computes $\mu = g^\alpha, v = g^\beta, w = g^{\alpha\beta}$. He sends μ, v, as well as a description of G and the generator g to $\mathcal{A}$.

(2). $\mathcal{A}$ outputs some $\hat{w} \in G$.

We define $\mathcal{A}$'s advantage (write CDHAdv$[\mathcal{A}, G]$ in solving the Computational Diffie Hellman problem with respect to G as the probability $w = \hat{w}$.

Definition A.4. We say the Computational Diffie Hellman assumption holds for G if for all efficient adversaries $\mathcal{A}$, the quantity CDHAdv$[\mathcal{A}, G]$ is a negligible function in the security parameter.

A.3 Strong Bilinear Diffie Hellman Assumption

Definition A.5. Let G be a group of prime order p and g be a generator. Consider the string $SRS = \{g, g^{\tau}, g^{\tau^2}, ..., g^{\tau^D}\}$ where τ is a random integer chosen from $\{1, ..., p-1\}$. The D Strong Bilinear Diffie Hellman Assumption states that given such a string, there is no efficient algorithm that can output a pair $(z, g^{\frac{1}{\tau - z}})$ except with negligible probability.

A.4 Weak Bilinear Diffie Hellman Assumption

We give the same variant of this problem as defined in Pointproofs [41].

Definition A.6. Let G_1, G_2, G_T be groups of prime order p with a nondegenerate bilinear pairing $e : G_1 \times G_2 \longrightarrow G_T$. Fix generators g_1, g_2, and $g_T := e(g1, g2)$ for the three groups. For some secret exponent α, an adversary is given

$$g_1^{\alpha}, \dots, g_1^{\alpha^{\ell}}$$
$$g_1^{\alpha^{\ell+2}}, \dots, g_1^{\alpha^{3\ell}}$$
$$g_2^{\alpha}, \dots, g_2^{\alpha^{\ell}}.$$

The adversary is asked to output $g_1^{\alpha^{\ell+1}}$.

A.5 Group of Unknown Order Assumptions

Let $\mathsf{GGen}(1^{\lambda})$ be a probabilistic algorithm that generates a group G with order in a range ord_{min}, ord_{max} such that $\frac{1}{ord_{min}}, \frac{1}{ord_{max}}, \frac{1}{ord_{max} - ord_{min}} \in negl(\lambda)$. We say the adaptive root assumption holds for Ggen if for any PPT adversary $(\mathcal{A}_1, \mathcal{A}_2)$

$$Pr\Big[u^{\ell} = w \wedge w \neq 1 : G \leftarrow \mathsf{GGen}(1^{\lambda}),$$
$$(w, state) \leftarrow \mathcal{A}_1(G)$$
$$\ell \leftarrow Primes(\lambda)$$
$$u \leftarrow \mathcal{A}_2(\ell, state)\Big] \leq negl(\lambda)$$

where $Primes(x)$ denotes the set of all primes less than or equal to x.

We say the Strong RSA Assumption holds for GGen if for any PPT adversary $\mathcal{A}$

$$Pr\Big[u^{e} = g \wedge \text{e is prime} : G \leftarrow \mathsf{GGen}(1^{\lambda}),$$
$$g \leftarrow G,$$
$$(u, e) \leftarrow \mathcal{A}(G, g)\Big] \leq negl(\lambda).$$

We say the Strong Distinct Prime Product Root assumption holds for Ggen if for any PPT adversary $\mathcal{A}$,

$$Pr\Big[u^{\prod_k} = g, \wedge e_i \in Primes(\lambda), \wedge e_i \neq e_j \text{ for } i \neq j : G \leftarrow \mathsf{Ggen}(1^{\lambda}),$$
$$g \leftarrow G,$$
$$(u, \{e_i\}_{i \in S})$$
$$g \leftarrow G$$
$$(u, \{e_i\}_{i \in S}) \leftarrow \mathcal{A}(G, g)\Big] \leq negl(\lambda).$$

A.6 Short Integer Solutions

Definition A.7 (Short Integer Solutions). Given a matrix $A \in \mathbb{Z}_q^{n\times m}$, the $SIS(n, m, \beta, q)$ problem asks us to find a *short* and nonzero vector e in the kernel of A. Namely, $0 < ||e||_2 \leq \beta$.

Received 20 February 2007; revised 12 March 2009; accepted 5 June 2009